Secure Two-party Computation (Secure 2PC) Using Machine Learning

Shilpa Batthineni
Masters in Computer Science
Georgia State University
Georgia, United States
sbatthineni1@student.gsu.edu

Pratyush Reddy Gaggenapalli
Masters in Computer Science
Georgia State University
Georgia, United States
pgaggenapalli1@student.gsu.edu

***Abstract*—Secure two-party computation (Secure 2PC) using machine learning refers to a technique where two parties can perform machine learning computations collaboratively while keeping their data private. This technique is particularly useful when two parties have access to different datasets, but both datasets need to be utilized for machine learning applications. By using cryptographic techniques such as homomorphic encryption, the two parties can jointly compute the result of a machine learning model without revealing any sensitive information to each other.**

***Index Terms*—Secure two-party computation,Secure 2PC, CRYPTEN, homomorphic encryption, Machine learning.**

I. Introduction

Secure two-party computation (Secure 2PC) is a form of secure multi-party computation (MPC) that allows two parties to jointly compute a function while keeping their inputs private.

In other words, it enables a group of parties to perform a computation on their private data without having to reveal that data to each other. This is achieved by using cryptographic protocols that allow the parties to jointly compute a function on their inputs, while ensuring that no party learns anything about the other parties' inputs beyond what can be inferred from the output of the computation. Secure computation has many applications in areas such as privacy-preserving data analysis, secure electronic voting, and secure machine learning.

There is given a set of parties (players, computers, authorities...) who want to do a joint computation but may not trust each other!!!

Example (The millionaire‘s problem): There are 2 millionaires who want to find out how is richer (without of course revealing each other the exact amount of money they own).
In the Millionaire's problem scenario, two millionaires, Alice and Bob, want to determine who is wealthier without disclosing the exact amount of their wealth to each other. A third party, called the trusted party, is not involved in the comparison but can provide a secure protocol for Alice and Bob to carry out the comparison without either party learning the other's actual wealth.

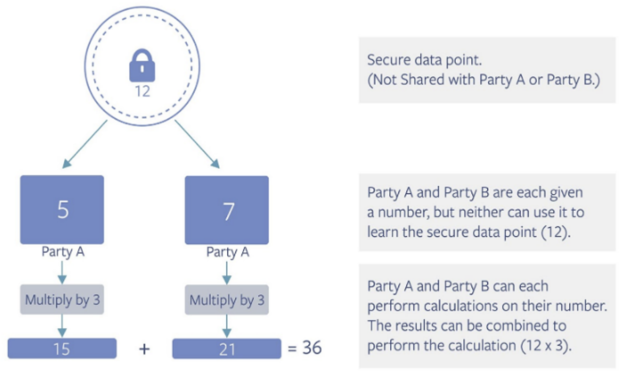
Secure computation can be used to solve this problem by using cryptographic techniques. One approach to solving the Millionaire's problem is using fully homomorphic encryption (FHE), which allows computations to be performed directly on encrypted data. Alice and Bob can encrypt their wealth using FHE, and then perform the comparison on the encrypted values without revealing the actual values to each other or the trusted party.

In this approach a secure computation can be used to solve the Millionaire's problem without revealing their inputs to each other.

A. *Two-Party Compute: An Example*

Secure 2PC encrypts information by dividing data between multiple parties, who can each perform calculations on their share (in this example, 5 and 7) but are not able to read the original data (12).

Each party then computes ("multiply by 3"). When the outputs are combined, the result (36) is identical to the result of performing the calculation on the data directly. Since Party A and Party B do not know the end result (36) they can not deduce the original data point (12).



12
Secure data point.
(Not Shared with Party A or Party B.)
5
Party A
7
Party A
Party A and Party B are each given a number, but neither can use it to learn the secure data point (12).
Multiply by 3
Multiply by 3
15
+
21
= 36
Party A and Party B can each perform calculations on their number. The results can be combined to perform the calculation (12 x 3).

Fig. 1. Secure Two-Party Compute:

II. Motivation

In many applications, it is important to preserve the privacy of individual users. By integrating cryptography with machine learning, models can be trained on encrypted data without revealing information about individual users, ensuring their privacy. We wanted to classify encrypted data using an encrypted model and then validate the encrypted classification. i.e., validate whether this output can be decrypted into meaningful data.

Machine learning models that are transmitted without using secure methods can be vulnerable to various attacks. For example, Model inversion attacks, Model stealing attacks: In these attacks, an attacker tries to determine whether a particular record was used to train the model i.e., an attacker could try to determine if a particular individual's data was used to train a model that predicts whether someone has a certain disease.

Using cryptography in machine learning, datasets can be trained on sensitive data without exposing that data to unauthorized parties, reducing the risk of data breaches and maintaining privacy. This is especially important for applications in fields such as healthcare, finance, and government, where the data being processed is particularly sensitive and confidentiality is of utmost importance.

III. Secure two way computation using CRYPTEN

To foster the adoption of secure computation techniques in machine learning, CRYPTEN is used to secure the models. CRYPTEN is a software framework designed to promote the adoption of secure computation techniques in machine learning. It provides a flexible platform for modern secure MPC techniques that are accessible to machine learning researchers and developers without a background in cryptography. The framework includes automatic differentiation and a modular neural network package, and it assumes a semi-honest threat model while working for an arbitrary number of parties. To make private training and inference efficient, CRYPTEN offloads computations to the GPU. Performance benchmarks show that CRYPTEN is capable of efficient private evaluation of modern machine learning models under a semi-honest threat model.

A. *Workflow of the Machine Learning Model*

High-level workflow of how CRYPTEN is included and how output is provided.

- Data Preparation: The input data is prepared for processing by the machine learning model. This may include tasks such as normalization, feature selection, encryption and encoding.

- Model Training: The machine learning model is trained on the prepared input data using the CRYPTEN framework. During training, CRYPTEN ensures that the data remains encrypted and secure.
- Encrypted Inference: Once the model is trained, it can be used for encrypted inference on new data. The input data is encrypted using the CRYPTEN framework and the trained model is applied to the encrypted data. The resulting output is also encrypted.
- Decryption: The encrypted output is sent to the party who has the decryption key. This party decrypts the output to obtain the final result. Overall, the workflow involves preparing input data, training the model, performing encrypted inference, and decrypting the output. The CRYPTEN framework ensures that the data remains secure throughout the entire process.

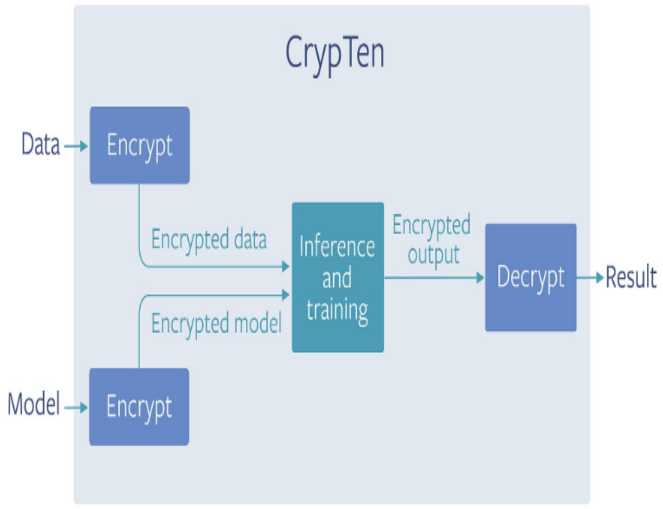


Fig. 2. Secure Computation Workflow:

B. Secure two party computation example using CRYPTEN:

CRYPTEN is a new framework built on PyTorch to facilitate research in secure and privacy-preserving machine learning.

- Privacy preserving: In many applications, it is important to preserve the privacy of individual users. By using CrypTen, machine learning models can be trained on encrypted data without revealing information about individual users, ensuring their privacy.
- Secure Computation: CrypTen uses homomorphic encryption and secure two-party computation to enable computation on encrypted data. This ensures that computation can be performed securely and privately, even in distributed environments.

CRYPTEN lowers the barrier for machine learning researchers by integrating with the common PyTorch API. It was originally developed to support secure two-party computations for machine learning applications, although it has since been expanded to support more parties as well.

It uses advanced cryptographic techniques, such as secure ***homomorphic*** encryption, to enable parties to jointly compute on their private data without revealing it to other parties. This makes it possible to perform secure computations on sensitive data without compromising privacy

Suppose Alice and Bob each have a secret number, and they want to determine who has the larger number without revealing their actual values to each other. They can use Crypten to securely compute the maximum of their two values.

- Step 1: **Set up the environment -** Both Alice and Bob need to set up the Crypten environment by installing the necessary packages and libraries. They also need to agree on the protocol to use for the computation. For this example, let's assume they will use the Paillier cryptosystem.
- Step 2: **Initialize the parties -** Each party ini-

tializes their instance of Crypten and generates their public and private keys for the Paillier cryptosystem.

- Step 3: **Input the data -** Alice and Bob input their respective secret values into their Crypten instances.
- Step 4: **Encrypt the data -** Both Alice and Bob encrypt their input data using the other party's public key. This ensures that each party only has access to the encrypted version of the other party's input.
- Step 5: **Compute the maximum -** Using Crypten's secure computation primitives, Alice and Bob perform a comparison operation on their encrypted input data to determine which value is greater. The result of this computation is encrypted and shared between the two parties.
- Step 6: **Decrypt the result -** Alice and Bob decrypt the result of the computation using their own private keys. They can then compare the decrypted result to determine which party had the larger input value.
  Step 7: **Output the result -** Finally, Alice and Bob output the result of the computation, without revealing their actual input values to each other.

  In summary, the secure two-party computation using Crypten involved initializing the parties, inputting the data, encrypting the data, computing the maximum, decrypting the result, and outputting the result. Throughout the process, the security of the computation was maintained by using encryption and secure computation primitives.

C. Homomorphic Encryption

CRYPTEN uses homomorphic encryption as its encryption mechanism. Homomorphic encryption is a type of encryption that enables computation to be performed on encrypted data without decrypting it first. This means that sensitive data can be kept encrypted while still allowing for computation to be performed on it, which is important in privacy-preserving machine learning applications.

Homomorphic encryption can be used in different ways, such as-

- **Fully homomorphic encryption** enables arbitrary computations to be performed on encrypted data, but it is computationally expensive and not practical for many applications.
- **Partially homomorphic encryption**, on the other hand, enables only specific computations to be performed on encrypted data, but it is more efficient than fully homomorphic encryption and can be used in many applications.There are two main types of PHE:
  - Additive Homomorphic Encryption (AHE): This type of PHE allows for addition operations to be performed on encrypted data. For example, if we have encrypted data x and y, we can perform an addition operation on the encrypted values and obtain the encryption of the sum (x + y).
  - Multiplicative Homomorphic Encryption (MHE): This type of PHE allows for multiplication operations to be performed on encrypted data. For example, if we have encrypted data x and y, we can perform a multiplication operation on the encrypted values and obtain the encryption of the product (x * y).
- **Fixed-Point Homomorphic Encryption (FPHE):** This is a type of PHE that is specifically designed for efficient computation on floating-point numbers, which are commonly used in machine learning applications. FPHE

enables the computation of machine learning models on encrypted data without significant computational overhead.

CRYPTEN uses a specific variant of homomorphic encryption called Additive homomorphic encryption (AHE), which is a type of partially homomorphic encryption. AHE is well-suited to machine learning applications because it allows for efficient computation on encrypted data, making it possible to train machine learning models on encrypted data without significant computational overhead.

Lets see an example of how to use AHE to encrypt the numbers 24 and 12, add them together, and decrypt the result:

Choose public and private keys for the encryption scheme. For simplicity, we'll use the same keys for both numbers:

Public key: n = 33, g = 2

Private key: lambda $(\lambda) = 10, \mu(mu) = 23$

Encrypt the number 24:

- Choose a random number r = 4 (which is relatively prime to n)
- Calculate the ciphertext as c = g$^24 * r^n mod n^2 = 2^24 * 4^33 mod 33^2 = 1984$

Encrypt the number 12:

- Choose a random number s = 5 (which is relatively prime to n)
- Calculate the ciphertext as d = g$^12 * s^n mod n^2 = 2^12 * 5^33 mod 33^2 = 1216$

Add the encrypted numbers together:

Compute the product of the ciphertexts as

e = c*d mod n^2

= 1984 * 1216 mod 33$^2 = 255360$

Note that this value is still encrypted and cannot be directly read as the sum of 24 and 12.

Decrypt the result:

Use the private key to decrypt the result as

$$m = L(e^\lambda mod n^2) * \mu mod n \tag{1}$$

= L(255360$^\lambda mod 33^2) * 23 mod 33 = 36$

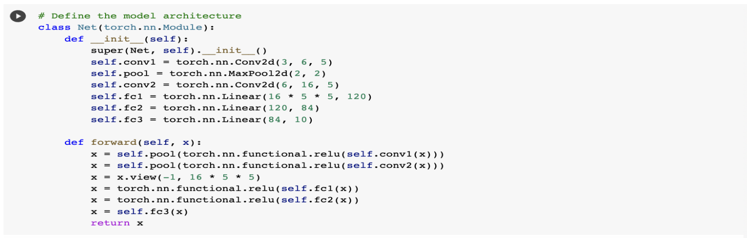
The decrypted value is 36, which is the sum of the original plaintexts 24 and 12. So, the result of adding the encrypted numbers 24 and 12 is 36 after decrypting the encrypted sum.

IV. Implementation

Steps to perform secure two-party computation using CrypTen:

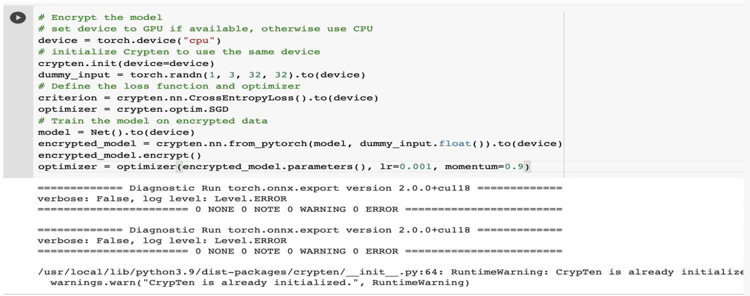
- **Define two parties:** In secure two-party computation, two parties (often referred to as Alice and Bob) perform computations on their private inputs while keeping them secret from each other. In Crypten, these two parties are represented by two different devices or processes, and each party holds their own data.
- **Initialize Crypten on both parties:** Before you can perform secure two-party computation, both parties need to initialize Crypten. Instead of using encryption keys, you can use *crypten.mpc.provider.MultiProcessBatchedSharedTensor Provider()* to securely share data between the two parties. This provider allows each party to share their local data with the other party, while ensuring that the data remains confidential throughout the computation.
- **Split the dataset:** Split the dataset into two parts and distribute them between the two parties. Each party holds their own part of the dataset and uses it to compute the gradients on their local data.
- **Encrypt the model:** Once the dataset is split, encrypt the model using the *crypten.nn.from_pytorch* method.
  This method takes the PyTorch model as input and returns an encrypted Crypten model that can be used for secure computation.
- **Compute the forward pass:** In each iteration of the training loop, each party computes

the forward pass on their local data by calling *encrypted_model.forward*. The input data is encrypted using crypten.cryptensor, which encrypts the data using secure multi-party computation techniques. The output of this computation is also encrypted, ensuring that neither party can see the intermediate values.



```
# Define the model architecture
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Conv2d(3, 6, 5)
        self.pool = torch.nn.MaxPool2d(2, 2)
        self.conv2 = torch.nn.Conv2d(6, 16, 5)
        self.fc1 = torch.nn.Linear(16 * 5 * 5, 120)
        self.fc2 = torch.nn.Linear(120, 84)
        self.fc3 = torch.nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(torch.nn.functional.relu(self.conv1(x)))
        x = self.pool(torch.nn.functional.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



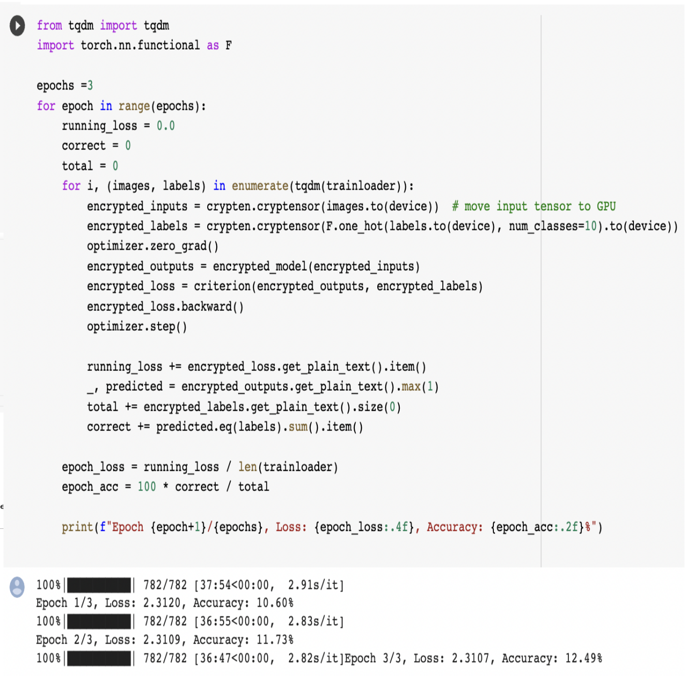
```
# Encrypt the model
# set device to GPU if available, otherwise use CPU
device = torch.device("cpu")
# initialize Crypten to use the same device
crypten.init(device=device)
dummy_input = torch.randn(1, 3, 32, 32).to(device)
# Define the loss function and optimizer
criterion = crypten.nn.CrossEntropyLoss().to(device)
optimizer = crypten.optim.SGD
# Train the model on encrypted data
model = Net().to(device)
encrypted_model = crypten.nn.from_pytorch(model, dummy_input.float()).to(device)
encrypted_model.encrypt()
optimizer = optimizer(encrypted_model.parameters(), lr=0.001, momentum=0.9)
```

```
============= Diagnostic Run torch.onnx.export version 2.0.0+cu118 =============
verbose: False, log level: Level.ERROR
======================= 0 NONE 0 NOTE 0 WARNING 0 ERROR ========================

============= Diagnostic Run torch.onnx.export version 2.0.0+cu118 =============
verbose: False, log level: Level.ERROR
======================= 0 NONE 0 NOTE 0 WARNING 0 ERROR ========================

/usr/local/lib/python3.9/dist-packages/crypten/__init__.py:64: RuntimeWarning: CrypTen is already initialize
  warnings.warn("CrypTen is already initialized.", RuntimeWarning)
```

- **Compute the loss and gradients:** After computing the forward pass, each party computes the loss and gradients on their local data. The loss is computed using the *crypten.nn.CrossEntropyLoss* method, and the gradients are computed using the backward method.
- **Combine the gradients:** After each party has computed the gradients on their local data, the gradients need to be combined to update the model weights. In Crypten, you can use the crypten.stack method to stack the gradients computed by each party, and then sum them along the first dimension using crypten.sum.
- **Update the model weights:** Finally, the updated gradients are used to update the model weights. This is done by calling optimizer.step() on the encrypted model.
- **Decrypt the output:** Once the model has been updated, the encrypted output is decrypted using *get_plain_text()* to get the plain-text output. The accuracy is then computed on this plain-text output. Repeat the training loop: The training loop is repeated for a fixed number of epochs, and the model weights are updated in each iteration until the model converges.



```
from tqdm import tqdm
import torch.nn.functional as F

epochs =3
for epoch in range(epochs):
    running_loss = 0.0
    correct = 0
    total = 0
    for i, (images, labels) in enumerate(tqdm(trainloader)):
        encrypted_inputs = crypten.cryptensor(images.to(device))  # move input tensor to GPU
        encrypted_labels = crypten.cryptensor(F.one_hot(labels.to(device), num_classes=10).to(device))
        optimizer.zero_grad()
        encrypted_outputs = encrypted_model(encrypted_inputs)
        encrypted_loss = criterion(encrypted_outputs, encrypted_labels)
        encrypted_loss.backward()
        optimizer.step()

        running_loss += encrypted_loss.get_plain_text().item()
        _, predicted = encrypted_outputs.get_plain_text().max(1)
        total += encrypted_labels.get_plain_text().size(0)
        correct += predicted.eq(labels).sum().item()

    epoch_loss = running_loss / len(trainloader)
    epoch_acc = 100 * correct / total

    print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}, Accuracy: {epoch_acc:.2f}%")
```

```
100%|██████████| 782/782 [37:54<00:00,  2.91s/it]
Epoch 1/3, Loss: 2.3120, Accuracy: 10.60%
100%|██████████| 782/782 [36:55<00:00,  2.83s/it]
Epoch 2/3, Loss: 2.3109, Accuracy: 11.73%
100%|██████████| 782/782 [36:47<00:00,  2.82s/it]Epoch 3/3, Loss: 2.3107, Accuracy: 12.49%
```

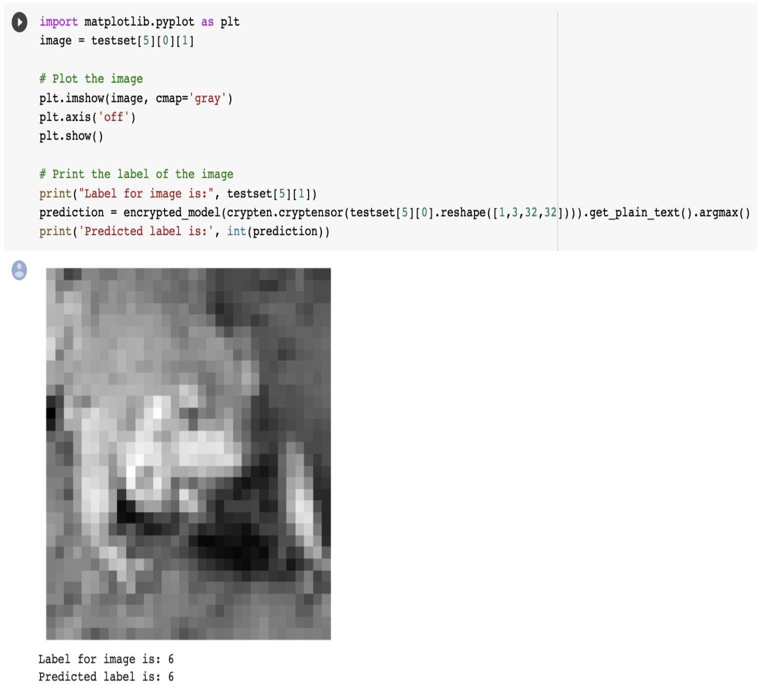
By using crypten.cryptensor instead of encryption keys, you can perform secure two-party computation using Crypten without having to manually generate and agree on encryption keys. Crypten handles all the necessary cryptographic operations to ensure that the data remains confidential throughout the computation.

V. Evaluation

In order to assess the performance of our model in a secure and privacy-preserving manner, we utilized secure two-party computation (STPC) techniques. The following steps were taken to evaluate the accuracy of the model on the encrypted test data:

- We encrypted the test data using Crypten's encryption capabilities, which enabled us to perform computations on the encrypted data without revealing any information about the underlying data.

- The encrypted test data was passed through the trained model to obtain the predictions. To decrypt the predictions, we used Crypten's decryption capabilities.



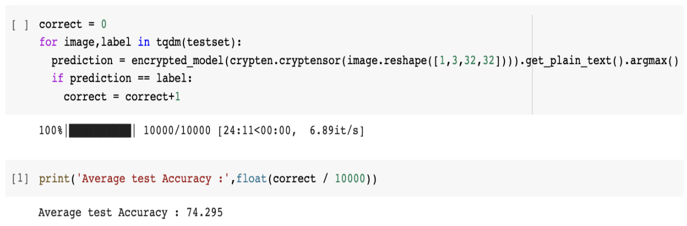
```
import matplotlib.pyplot as plt
image = testset[5][0][1]

# Plot the image
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.show()

# Print the label of the image
print("Label for image is:", testset[5][1])
prediction = encrypted_model(crypten.cryptensor(testset[5][0].reshape([1,3,32,32]))).get_plain_text().argmax()
print('Predicted label is:', int(prediction))
```

```
Label for image is: 6
Predicted label is: 6
```

- We computed the accuracy of the model by comparing the decrypted predictions with the actual labels of the test data.
- The average test accuracy achieved on the encrypted test data was 74.295% for a test set of 10,000 images.



```
[ ] correct = 0
    for image,label in tqdm(testset):
      prediction = encrypted_model(crypten.cryptensor(image.reshape([1,3,32,32]))).get_plain_text().argmax()
      if prediction == label:
        correct = correct+1

    100%|██████████| 10000/10000 [24:11<00:00,  6.89it/s]

[1] print('Average test Accuracy :',float(correct / 10000))

    Average test Accuracy : 74.295
```

Overall, the use of STPC techniques allowed us to evaluate the accuracy of our model in a secure and private manner, without compromising the confidentiality of the underlying data. This is especially important when dealing with sensitive data, as it ensures that the privacy of the individuals involved is protected.

VI. Challenges of using CrypTen

There are several challenges of using CrypTen, including:

- **Increased computational overhead:** Performing computations on encrypted data using homomorphic encryption can be computationally expensive, which can slow down the training and inference of machine learning models. This overhead can be especially significant when working with large datasets or complex models.
- **Communication overhead**: Secure multiparty computation requires significant communication between parties, which can lead to increased latency and network overhead. CrypTen tries to address this challenge by using efficient communication protocols, but it is still an area of active research.
- **Limited support for complex operations:** While CrypTen supports a variety of operations on encrypted data, it may not support all the operations required for certain machine learning tasks. For example, CrypTen does not support division, which can make certain types of models difficult to train using this framework.
- **Key management:** Homomorphic encryption requires the use of encryption keys to protect the privacy and security of the data. Managing these keys securely can be a challenge, especially in distributed settings where multiple parties may be involved.
- **Limited community support:** CrypTen is a relatively new framework, and its user base and community support are still growing. This can make it difficult to find resources, documentation, and support when using the framework.

Overall, while CrypTen has the potential to enable secure machine learning on private data sets, there are still several challenges that need to be addressed to make it more accessible and practical for real-world use cases.

VII. Conclusion

In conclusion, secure two-party computation (2PC) using machine learning has great potential for enabling collaborations between parties while keeping their private data confidential. Crypten is a flexible software framework that aims to make modern secure MPC techniques accessible to machine learning researchers and developers without a background in cryptography.

The advantages of using Crypten for secure 2PC include the ability to perform computations on encrypted data, ensuring the privacy of sensitive information, and the ability to offload computations to GPUs, making it more efficient. Additionally, Crypten provides features such as automatic differentiation and a modular neural network package, making it easier to train and evaluate machine learning models.

However, there are also potential downsides to the adoption of secure computation in machine learning, such as the difficulty in doing quality control of AI systems implemented in Crypten due to the inability to inspect intermediate activations and model outputs unless all parties agree to reveal those values.

Despite these challenges, secure 2PC using machine learning has the potential to lead to the development of more private and secure AI systems. By leveraging the benefits of both secure computation and machine learning, we can enable collaborations between parties with confidential data, and develop AI systems that preserve privacy and security.

References

[1] D. Demmler, T. Schneider, and M. Zohner. ABY – A framework for efficient mixed-protocol secure two-party computation. In NDSS, 2015.

[2] CI. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. https:// eprint.iacr.org/2011/535.

[3] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta. CRYPTEN: Secure Multi-Party Computation Meets Machine Learning https://openreview.net/pdf?id=dwJyEMPZ04I

[4] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi. EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning. In IEEE European Symposium on Security and Privacy, 2019.

[5] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High throughput semi-honest secure three-party computation with an honest majority. In ACM CCS, pages 805–817, 2016.

[6] https://crypten.ai/.

[7] D. W. Archer, D. Bogdanov, L. Kamm, Y. Lindell, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright. From keys to databases – real-world applications of secure multi-party computation. Computer Journal, 61(12):1749–1771, 2018.

[8] V. Haralampieva, D. Rueckert, and J. Passerat-Palmbach. A systematic comparison of encrypted machine learning solutions for image classification. In Proceedings of the Workshop on PrivacyPreserving Machine Learning in Practice, pages 55–59, 2020.

[9] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. SoK: general-purpose compilers for secure multi-party computation. In 2019 IEEE Symposium on Security and Privacy (SP), 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[11] A. S. Householder. The Numerical Treatment of a Single Nonlinear Equation. 1970.

[12] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (Part III: Fundamental Electronic Science), 72: 56–64, 1989.