Simulation Question 4:

Use 80 percent of the CIFAR-10 training data to train your model. This will serve as your baseline model

```
!git clone https://github.com/Ehsanacc/ML_Project.git

Cloning into 'ML_Project'...
remote: Enumerating objects: 34, done.ote: Counting objects: 100%
(34/34), done.ote: Compressing objects: 100% (33/33), done.ote: Total
34 (delta 1), reused 31 (delta 0), pack-reused 0
```

Importing needed libraries

```python
# Importing the libraries
from random import shuffle
import torch.optim as optim
import torch
from torchvision import datasets
import torchvision.transforms as transforms
import os
from torch.utils.data import Dataset, DataLoader, random_split,
TensorDataset
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression as LR
from datetime import datetime
from sklearn.preprocessing import StandardScaler
import pickle
```

Given base model

```python
class CIFAR10Classifier(nn.Module):
    def __init__(self):
        super(CIFAR10Classifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(6272, 64)
        self.fc2 = nn.Linear(64, 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)
        return x
```

Define Load data to get the needed Data

```
def get_data():
    # CIFAR-10 dataset mean and standard deviation
    cifar10_mean = np.array([0.49421428, 0.48513139, 0.45040909])
    cifar10_std = np.array([0.24665252, 0.24289226, 0.26159238])

    # CIFAR-10 dataset transforms
    transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(cifar10_mean, cifar10_std),
    ])

    # CIFAR-10 dataset transforms
    transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(cifar10_mean, cifar10_std),
    ])

    # Unnormalize transform for CIFAR-10 dataset
    unnormalize_transform = transforms.Normalize(-
cifar10_mean/cifar10_std, 1/cifar10_std)

    # CIFAR-10 dataset loading
    cifar10_dataset = datasets.CIFAR10(root='dataset', train=True,
download=True, transform=transform_train)
    train_dataset, val_dataset = random_split(cifar10_dataset, [45000,
5000])
    test_dataset = datasets.CIFAR10(root='dataset', train=False,
download=True, transform=transform_test)
    train_loader = DataLoader(dataset=train_dataset, batch_size=64,
shuffle=True)
    val_loader = DataLoader(dataset=val_dataset, batch_size=64,
shuffle=True)
    test_loader = DataLoader(dataset=test_dataset, batch_size=64,
shuffle=True)
```

```
    return train_loader, val_loader, test_loader

train_loader, val_loader, test_loader = get_data()
print(len(train_loader))
print(len(val_loader))
print(len(test_loader))
```

```
Files already downloaded and verified
Files already downloaded and verified
704
79
157
```

define model_train function

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

def model_train(model, train_loader, val_loader, criterion,
num_epochs, regularization_strength = None, model_name = None):
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    if regularization_strength == None:
        regularization_strength = 0

    train_loss_arr, val_loss_arr = [], []
    train_acc_arr, val_acc_arr = [], []
    for epoch in range(num_epochs):
        train_loss, val_loss = .0, .0
        train_acc, val_acc = .0, .0

        model.train()
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            l2_reg = sum(torch.sum(param ** 2) for param in
model.parameters())
            loss += regularization_strength * l2_reg
            train_loss += loss.item() * images.size(0)
            train_acc += torch.sum(torch.max(outputs, axis=1)[1] ==
labels).cpu().item()
            loss.backward()
            optimizer.step()

        model.eval()
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
```

```
                l2_reg = sum(torch.sum(param ** 2) for param in
model.parameters())
                loss += regularization_strength * l2_reg
                val_loss += loss.item() * images.size(0)
                val_acc += torch.sum(torch.max(outputs, axis=1)[1] ==
labels).cpu().item()

        train_loss /= len(train_loader.dataset)
        val_loss /= len(val_loader.dataset)
        train_acc /= len(train_loader.dataset)
        val_acc /= len(val_loader.dataset)

        train_loss_arr.append(train_loss)
        val_loss_arr.append(val_loss)
        train_acc_arr.append(train_acc)
        val_acc_arr.append(val_acc)

        print(f"[Epoch {epoch}]\t"
            f"[{datetime.now().strftime('%H:%M:%S')}]\t"
            f"Train Loss: {train_loss:.4f}\t"
            f"Train Accuracy: {train_acc:.2f}\t"
            f"Validation Loss: {val_loss:.4f}\t\t"
            f"Validation Accuracy: {val_acc:.2f}")
        if model_name != None:
            torch.save(model.state_dict(), f'{model_name}.pth')
```

Train the base line model using the given data in this section

```
model = CIFAR10Classifier().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
num_epochs = 10

model_train(model, train_loader, val_loader, criterion, num_epochs,
model_name='base_line_target_model')
```

```
[Epoch 0]  [14:44:02] Train Loss: 1.7475    Train Accuracy: 0.36
      Validation Loss: 1.4524          Validation Accuracy: 0.47
[Epoch 1]  [14:44:23] Train Loss: 1.5168    Train Accuracy: 0.45
      Validation Loss: 1.3628          Validation Accuracy: 0.51
[Epoch 2]  [14:44:46] Train Loss: 1.4437    Train Accuracy: 0.48
      Validation Loss: 1.2877          Validation Accuracy: 0.54
[Epoch 3]  [14:45:08] Train Loss: 1.3904    Train Accuracy: 0.50
      Validation Loss: 1.2520          Validation Accuracy: 0.56
[Epoch 4]  [14:45:29] Train Loss: 1.3582    Train Accuracy: 0.51
      Validation Loss: 1.1963          Validation Accuracy: 0.58
[Epoch 5]  [14:45:51] Train Loss: 1.3291    Train Accuracy: 0.52
      Validation Loss: 1.1900          Validation Accuracy: 0.58
[Epoch 6]  [14:46:13] Train Loss: 1.3026    Train Accuracy: 0.53
      Validation Loss: 1.1595          Validation Accuracy: 0.59
```

```
[Epoch 7]  [14:46:35] Train Loss: 1.2855    Train Accuracy: 0.54
      Validation Loss: 1.1380          Validation Accuracy: 0.60
[Epoch 8]  [14:46:57] Train Loss: 1.2698    Train Accuracy: 0.55
      Validation Loss: 1.1033          Validation Accuracy: 0.62
[Epoch 9]  [14:47:19] Train Loss: 1.2507    Train Accuracy: 0.56
      Validation Loss: 1.1241          Validation Accuracy: 0.61
```

Check model accuracy on test_loader

```
# Evaluate on the test set
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total}%")

Test Accuracy: 65.38%
```

Training phase

Simulation Question 5:

Train your baseline model with privacy enhancements. This is your modified model. Ensure that the test accuracy difference between your baseline model and the modified model is less than 15

Methods used for privacy enhancements:

```
rounding output of model to 3 digits
Restriction of prediction vector to top 3 elements
Using regulatization term λ||θ||
```

These methods help the model leak less information when classifying

```
class Private_CIFAR10Classifier(nn.Module):
    def __init__(self):
        super(Private_CIFAR10Classifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.pool = nn.MaxPool2d(2, 2)
```

```python
        # Calculate the input size for the fully connected layer
        self._to_linear = None
        self._get_conv_output_size()

        self.fc1 = nn.Linear(self._to_linear, 64)
        self.fc2 = nn.Linear(64, 10)

    def _get_conv_output_size(self):
        x = torch.rand(1, 3, 32, 32)
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)
        x = self.dropout1(x)
        self._to_linear = x.numel()

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)
        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)
        # only giving top k values
        k = 2
        topk_values, topk_indices = torch.topk(x, k, dim=1)
        mask = torch.zeros_like(x)
        mask.scatter_(1, topk_indices, topk_values)

        # Round the values in the mask to 3 decimal places
        rounded_mask = torch.round(mask * 100) / 100

        return mask
```

Start training phase for the new model created as private model from the same data had from earlier

```python
model = Private_CIFAR10Classifier().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
regularization_strength = 2e-3
num_epochs = 10
```

```
model_train(model, train_loader, val_loader, criterion, num_epochs,
regularization_strength=regularization_strength,
model_name='private_target_model')
```

```
[Epoch 0]  [14:47:43] Train Loss: 2.1538    Train Accuracy: 0.27
     Validation Loss: 2.0033         Validation Accuracy: 0.38
[Epoch 1]  [14:48:05] Train Loss: 2.0052    Train Accuracy: 0.35
     Validation Loss: 1.9529         Validation Accuracy: 0.38
[Epoch 2]  [14:48:27] Train Loss: 1.9572    Train Accuracy: 0.38
     Validation Loss: 1.8589         Validation Accuracy: 0.44
[Epoch 3]  [14:48:49] Train Loss: 1.9169    Train Accuracy: 0.41
     Validation Loss: 1.8330         Validation Accuracy: 0.44
[Epoch 4]  [14:49:11] Train Loss: 1.9007    Train Accuracy: 0.42
     Validation Loss: 1.8208         Validation Accuracy: 0.46
[Epoch 5]  [14:49:32] Train Loss: 1.8629    Train Accuracy: 0.44
     Validation Loss: 1.7448         Validation Accuracy: 0.49
[Epoch 6]  [14:49:55] Train Loss: 1.8539    Train Accuracy: 0.44
     Validation Loss: 1.7579         Validation Accuracy: 0.49
[Epoch 7]  [14:50:17] Train Loss: 1.8316    Train Accuracy: 0.45
     Validation Loss: 1.7146         Validation Accuracy: 0.50
[Epoch 8]  [14:50:39] Train Loss: 1.8265    Train Accuracy: 0.45
     Validation Loss: 1.7326         Validation Accuracy: 0.49
[Epoch 9]  [14:51:01] Train Loss: 1.8183    Train Accuracy: 0.46
     Validation Loss: 1.7107         Validation Accuracy: 0.51
```

Check private model accuracy on the test_loader

```
# Evaluate on the test set
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total}%")
```

```
Test Accuracy: 55.72%
```

Simulation Question 6.

Train two Attacker Models based on MIA techniques learned in Phase 0, one for the baseline model and one for the modified model. Compare the MIA accuracy of these two attacker models. Use 80 percent of the training data as your seen data, and the remaining training data along with the test data as your unseen data

Create a shadow model to mimic the target model

```
class ShadowModel(nn.Module):
    def __init__(self):
        super(ShadowModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(6272, 64)
        self.fc2 = nn.Linear(64, 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)
        return F.softmax(x, dim=1)
```

create private shadow model to mimic private target model

```
class Private_ShadowModel(nn.Module):
    def __init__(self):
        super(Private_ShadowModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.pool = nn.MaxPool2d(2, 2)

        # Calculate the input size for the fully connected layer
        self._to_linear = None
        self._get_conv_output_size()

        self.fc1 = nn.Linear(self._to_linear, 64)
        self.fc2 = nn.Linear(64, 10)

    def _get_conv_output_size(self):
        x = torch.rand(1, 3, 32, 32)
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
```

```
        x = self.pool(x)
        x = self.dropout1(x)
        self._to_linear = x.numel()

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)
        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)
        # only giving top k values
        k = 2
        topk_values, topk_indices = torch.topk(x, k, dim=1)
        mask = torch.zeros_like(x)
        mask.scatter_(1, topk_indices, topk_values)

        # Round the values in the mask to 3 decimal places
        rounded_mask = torch.round(mask * 100) / 100

        return mask
```

Train shadow model in this section for base line model

```
# Train 10 different shadow models, each with its corresponding label
dataset
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
shadow_model = ShadowModel().to(device)
num_epochs = 5
criterion = nn.CrossEntropyLoss()


print(f"Training shadow model ...")
model_train(shadow_model, train_loader, val_loader, criterion,
num_epochs)

# Save the shadow models
torch.save(shadow_model.state_dict(), f'shadow_model_Q6.pth')

# Example: Print summary of one shadow model
print(shadow_model)
```

```
Training shadow model ...
[Epoch 0]  [14:51:26] Train Loss: 2.1694    Train Accuracy: 0.28
      Validation Loss: 2.0998         Validation Accuracy: 0.35
[Epoch 1]  [14:51:48] Train Loss: 2.1151    Train Accuracy: 0.34
      Validation Loss: 2.0583         Validation Accuracy: 0.40
[Epoch 2]  [14:52:10] Train Loss: 2.0855    Train Accuracy: 0.37
      Validation Loss: 2.0496         Validation Accuracy: 0.40
[Epoch 3]  [14:52:31] Train Loss: 2.0655    Train Accuracy: 0.39
      Validation Loss: 2.0129         Validation Accuracy: 0.44
[Epoch 4]  [14:52:53] Train Loss: 2.0519    Train Accuracy: 0.40
      Validation Loss: 2.0001         Validation Accuracy: 0.46
ShadowModel(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=6272, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

Create datas suitable for tranining the Attacker model these datas are created by using main datas on shadow models that mimic the target model

We get the outputs of shadow model and concat them with the true label.

We use this pair as inputs of the attacker model and the outputs of attacker model would be in or out

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Function to combine inputs and outputs into new dataset entries
def create_combined_data_of_simple_shadow(shadow_models,
loaders_by_label, mode):

    shadow_model = shadow_models[0]
    shadow_model.eval()

    with torch.no_grad():
        for images, true_outputs in loaders_by_label:
            images = images.to(device)
            outputs = shadow_model(images).cpu()
            for true_output, output in zip(true_outputs, outputs):
                true_output, output = true_output.to(device),
output.to(device)
                # Ensure true_output and output have the same number
of dimensions
                if true_output.dim() == 0:  # Handle scalar tensor
case
                    true_output = true_output.unsqueeze(0)
                if output.dim() > 1:  # Handle higher-dimensional
```

```
output tensor
                    output = output.squeeze()  # Adjust dimensions as
needed
                combined_output = torch.cat((true_output, output),
dim=0)
                yield (combined_output, mode)
            torch.cuda.empty_cache()

# Initialize the dictionary for the shadow models
shadow_models = {}

model_dir = './ML_Project/'

# Load shadow models
model_path = os.path.join(model_dir, f'shadow_model_Q6.pth')
shadow_model = ShadowModel().to(device)
shadow_model.load_state_dict(torch.load(model_path,
map_location=device))
shadow_model.eval()  # Set the model to evaluation mode
shadow_models[0] = shadow_model

# Example: Print the keys of the shadow_models dictionary to verify
print("Loaded shadow models:", shadow_models.keys())

# Create seen data
combined_in_data1 =
list(create_combined_data_of_simple_shadow(shadow_models,
train_loader, 1))
combined_in_data2 =
list(create_combined_data_of_simple_shadow(shadow_models, val_loader,
1))
combined_in_data = combined_in_data1 + combined_in_data2

# Create unseen data
combined_out_data =
list(create_combined_data_of_simple_shadow(shadow_models, test_loader,
-1))
```

```
Loaded shadow models: dict_keys([0])
```

Print a few data samples to see their looks and shapes

```
# Example: Print first few entries of each data
for i in range(5):
    input_data, label = combined_in_data[i]
    print(f"Input: {input_data}, Label: {label}")

for i in range(5):
    input_data, label = combined_out_data[i]
```

```
    print(f"Input: {input_data}, Label: {label}")

print(len(combined_in_data))
print(len(combined_out_data))

Input: tensor([9.0000e+00, 4.2485e-08, 4.7822e-08, 2.8395e-08,
2.8028e-03, 4.3366e-05,
        2.8054e-05, 3.4315e-01, 1.3451e-01, 8.7503e-09, 5.1946e-01],
       device='cuda:0'), Label: 1
Input: tensor([8.0000e+00, 7.0774e-14, 1.0000e+00, 4.7829e-33,
5.0776e-30, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 9.8396e-41, 3.7384e-12, 2.8850e-06],
       device='cuda:0'), Label: 1
Input: tensor([6.0000e+00, 1.9649e-17, 6.3153e-13, 1.8519e-10,
3.1942e-06, 4.1173e-07,
        3.2729e-08, 1.0000e+00, 1.6778e-07, 5.4525e-16, 1.3804e-08],
       device='cuda:0'), Label: 1
Input: tensor([9.0000e+00, 1.7009e-15, 1.7859e-06, 7.6702e-08,
6.3544e-05, 1.7880e-07,
        8.0279e-06, 9.9993e-01, 1.9776e-09, 6.8186e-14, 4.0539e-08],
       device='cuda:0'), Label: 1
Input: tensor([2.0000e+00, 1.0000e+00, 9.0694e-16, 5.8849e-25,
7.2715e-18, 4.9520e-31,
        1.6459e-23, 4.1276e-34, 3.3222e-24, 3.7914e-09, 1.4709e-19],
       device='cuda:0'), Label: 1
Input: tensor([4.0000e+00, 1.8438e-03, 3.3926e-03, 1.6443e-03,
4.1506e-01, 1.2991e-05,
        5.1745e-01, 1.3443e-06, 5.9776e-02, 2.8087e-08, 8.1262e-04],
       device='cuda:0'), Label: -1
Input: tensor([4.0000e+00, 2.6370e-07, 1.7773e-09, 1.8782e-05,
6.1057e-12, 1.8857e-03,
        4.2476e-11, 4.4047e-13, 9.9810e-01, 3.1444e-18, 1.0676e-09],
       device='cuda:0'), Label: -1
Input: tensor([5.0000e+00, 7.3882e-08, 9.2633e-13, 3.7857e-04,
1.7943e-06, 9.1044e-04,
        9.2498e-05, 2.6615e-13, 9.9862e-01, 1.9710e-15, 4.4814e-11],
       device='cuda:0'), Label: -1
Input: tensor([2.0000e+00, 4.6353e-10, 5.1917e-10, 7.0469e-05,
2.3946e-02, 1.7676e-03,
        9.7385e-01, 2.7273e-04, 7.3435e-06, 8.4338e-05, 3.5823e-08],
       device='cuda:0'), Label: -1
Input: tensor([3.0000e+00, 4.3024e-18, 2.6709e-11, 4.8469e-09,
1.8405e-07, 7.2679e-07,
        1.2800e-09, 1.0000e+00, 3.8598e-09, 7.4103e-16, 4.3825e-10],
       device='cuda:0'), Label: -1
50000
10000
```

Shuffle Attacker model data

```
# Step 1: Combine the seen and unseen data
def get_shadow_datasets(combined_in_data, combined_out_data):
    # print(len(combined_in_data))
    # print(len(combined_out_data))
    combined_dataset = combined_in_data + combined_out_data
    shuffle(combined_dataset)
    # print(len(combined_dataset))

    # Step 2: Split 60,000 data into 50,000 (train/validation) and
10,000 (test)
    train_data = combined_dataset[:50000]
    test_data = combined_dataset[50000:]

    # Create DataLoaders
    batch_size = 64

    train_shadow_loader = DataLoader(train_data,
batch_size=batch_size, shuffle=True)
    test_shadow_loader = DataLoader(test_data, batch_size=batch_size,
shuffle=True)
    return train_shadow_loader, test_shadow_loader

train_shadow_loader, test_shadow_loader =
get_shadow_datasets(combined_in_data, combined_out_data)

# Example: Print sizes to verify
# print(f"Total Combined Dataset Size: {len(combined_dataset)}")
print(f"Training Data Size: {len(train_shadow_loader)}")
print(f"Test Data Size: {len(test_shadow_loader)}")

Training Data Size: 782
Test Data Size: 157
```

Prepare datas for linear regression classifier

```
# Function to extract data and labels from DataLoader
def extract_data_and_labels(loader):
    data = []
    labels = []
    for x,y in loader:
        data.append(x.cpu().numpy())  # Assuming DataLoader returns
PyTorch tensors
        labels.append(y.cpu().numpy())
    data = np.concatenate(data, axis=0)
    labels = np.concatenate(labels, axis=0)
    # normalizing
    scaler = StandardScaler()
    scaler.fit(data)
    scaled_data = scaler.transform(data)
    return scaled_data, labels
```

```
X_train, y_train = extract_data_and_labels(train_shadow_loader)
X_test, y_test = extract_data_and_labels(test_shadow_loader)
```

Use linear regression as attcker model (binary classifier)

```
lr = LR()
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)
with open('basic_attack_model.pkl', 'wb') as file:
    pickle.dump(lr, file)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 83.83%
```

Train private shadow model in this section to mimic private model

```
# Train 10 different shadow models, each with its corresponding label
dataset
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
private_shadow_model = Private_ShadowModel().to(device)
num_epochs = 5
criterion = nn.CrossEntropyLoss()


print(f"Training shadow model ...")
model_train(private_shadow_model, train_loader, val_loader, criterion,
num_epochs)

# Save the shadow models
torch.save(private_shadow_model.state_dict(),
f'private_shadow_model_Q6.pth')

# Example: Print summary of one shadow model
print(private_shadow_model)
```

```
Training shadow model ...
[Epoch 0]  [14:54:46] Train Loss: 2.0675    Train Accuracy: 0.28
     Validation Loss: 1.7886         Validation Accuracy: 0.42
[Epoch 1]  [14:55:09] Train Loss: 1.8235    Train Accuracy: 0.39
     Validation Loss: 1.6567         Validation Accuracy: 0.48
[Epoch 2]  [14:55:30] Train Loss: 1.7397    Train Accuracy: 0.42
     Validation Loss: 1.6131         Validation Accuracy: 0.49
[Epoch 3]  [14:55:52] Train Loss: 1.6844    Train Accuracy: 0.44
     Validation Loss: 1.5294         Validation Accuracy: 0.52
[Epoch 4]  [14:56:14] Train Loss: 1.6478    Train Accuracy: 0.45
     Validation Loss: 1.5000         Validation Accuracy: 0.52
```

```
Private_ShadowModel(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=6272, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

Create datas suitable for tranining the Attacker model these datas are created by using main datas on private shadow models that mimic the private target model

We get the outputs of private shadow model and concat them with the true label.

We use this pair as inputs of the attacker model and the outputs of attacker model would be in or out

```
# Initialize the dictionary for the shadow models
private_shadow_models = {}

model_dir = './ML_Project/'

# Load shadow models
model_path = os.path.join(model_dir, f'private_shadow_model_Q6.pth')
private_shadow_model = Private_ShadowModel().to(device)
private_shadow_model.load_state_dict(torch.load(model_path,
map_location=device))
private_shadow_model.eval()  # Set the model to evaluation mode
private_shadow_models[0] = private_shadow_model

# Example: Print the keys of the shadow_models dictionary to verify
print("Loaded private shadow models:", private_shadow_models.keys())

# Create seen data
private_combined_in_data1 =
list(create_combined_data_of_simple_shadow(private_shadow_models,
train_loader, 1))
private_combined_in_data2 =
list(create_combined_data_of_simple_shadow(private_shadow_models,
val_loader, 1))
private_combined_in_data = private_combined_in_data1 +
private_combined_in_data2

# Create unseen data
private_combined_out_data =
list(create_combined_data_of_simple_shadow(private_shadow_models,
test_loader, -1))
```

```
Loaded private shadow models: dict_keys([0])
```

shuffle attacker model datas

```
private_train_shadow_loader, private_test_shadow_loader =
get_shadow_datasets(private_combined_in_data,
private_combined_out_data)

# Example: Print sizes to verify
# print(f"Total Combined Dataset Size: {len(combined_dataset)}")
print(f"Training Data Size: {len(private_train_shadow_loader)}")
print(f"Test Data Size: {len(private_test_shadow_loader)}")

print(f'sample train data: {private_train_shadow_loader.dataset[0]}')

Training Data Size: 782
Test Data Size: 157
sample train data: (tensor([9.0000, 0.0000, 0.0000, 0.0000, 0.5511,
0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.5516], device='cuda:0'), 1)
```

prepare data for linear regression attacker model

```
# Extract data and labels from DataLoader
private_X_train, private_y_train =
extract_data_and_labels(private_train_shadow_loader)
private_X_test, private_y_test =
extract_data_and_labels(private_test_shadow_loader)

lr = LR()
lr.fit(private_X_train, private_y_train)
private_y_pred = lr.predict(private_X_test)
with open('basic_private_attack_model.pkl', 'wb') as file:
    pickle.dump(lr, file)

accuracy = accuracy_score(private_y_test, private_y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))

Accuracy: 82.80%
```

Train shadow models and save them

```
# Train 10 different shadow models, each with its corresponding label
dataset
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
shadow_models = {}
num_epochs = 5
num_shadows = 10
criterion = nn.CrossEntropyLoss()
```

```python
for s in range(num_shadows):
    shadow_model = ShadowModel().to(device)
    print(f"Training shadow model number {s}...")
    model_train(shadow_model, train_loader, val_loader, criterion,
num_epochs)
    shadow_models[s] = shadow_model

# Save the shadow models
for s in range(10):
    torch.save(shadow_models[s].state_dict(), f'shadow_model_{s}.pth')

# Example: Print summary of one shadow model
print(shadow_models[0])
```

```
Training shadow model number 0...
[Epoch 0]  [14:57:30] Train Loss: 2.1786    Train Accuracy: 0.27
      Validation Loss: 2.0934         Validation Accuracy: 0.36
[Epoch 1]  [14:57:52] Train Loss: 2.1075    Train Accuracy: 0.35
      Validation Loss: 2.0593         Validation Accuracy: 0.39
[Epoch 2]  [14:58:13] Train Loss: 2.0825    Train Accuracy: 0.37
      Validation Loss: 2.0271         Validation Accuracy: 0.43
[Epoch 3]  [14:58:35] Train Loss: 2.0617    Train Accuracy: 0.39
      Validation Loss: 2.0043         Validation Accuracy: 0.45
[Epoch 4]  [14:58:56] Train Loss: 2.0455    Train Accuracy: 0.41
      Validation Loss: 1.9929         Validation Accuracy: 0.46
Training shadow model number 1...
[Epoch 0]  [14:59:18] Train Loss: 2.1661    Train Accuracy: 0.28
      Validation Loss: 2.0994         Validation Accuracy: 0.35
[Epoch 1]  [14:59:39] Train Loss: 2.1027    Train Accuracy: 0.35
      Validation Loss: 2.0624         Validation Accuracy: 0.39
[Epoch 2]  [15:00:01] Train Loss: 2.0790    Train Accuracy: 0.37
      Validation Loss: 2.0295         Validation Accuracy: 0.43
[Epoch 3]  [15:00:22] Train Loss: 2.0585    Train Accuracy: 0.40
      Validation Loss: 2.0135         Validation Accuracy: 0.44
[Epoch 4]  [15:00:44] Train Loss: 2.0414    Train Accuracy: 0.41
      Validation Loss: 1.9882         Validation Accuracy: 0.47
Training shadow model number 2...
[Epoch 0]  [15:01:05] Train Loss: 2.1573    Train Accuracy: 0.29
      Validation Loss: 2.0857         Validation Accuracy: 0.37
[Epoch 1]  [15:01:27] Train Loss: 2.0887    Train Accuracy: 0.37
      Validation Loss: 2.0540         Validation Accuracy: 0.40
[Epoch 2]  [15:01:48] Train Loss: 2.0648    Train Accuracy: 0.39
      Validation Loss: 2.0150         Validation Accuracy: 0.44
[Epoch 3]  [15:02:10] Train Loss: 2.0465    Train Accuracy: 0.41
      Validation Loss: 2.0085         Validation Accuracy: 0.45
[Epoch 4]  [15:02:31] Train Loss: 2.0349    Train Accuracy: 0.42
      Validation Loss: 1.9946         Validation Accuracy: 0.46
Training shadow model number 3...
```

```
[Epoch 0]  [15:02:53] Train Loss: 2.1555    Train Accuracy: 0.29
      Validation Loss: 2.0732          Validation Accuracy: 0.38
[Epoch 1]  [15:03:15] Train Loss: 2.0821    Train Accuracy: 0.37
      Validation Loss: 2.0711          Validation Accuracy: 0.38
[Epoch 2]  [15:03:36] Train Loss: 2.0559    Train Accuracy: 0.40
      Validation Loss: 2.0170          Validation Accuracy: 0.43
[Epoch 3]  [15:03:58] Train Loss: 2.0381    Train Accuracy: 0.42
      Validation Loss: 1.9965          Validation Accuracy: 0.46
[Epoch 4]  [15:04:19] Train Loss: 2.0204    Train Accuracy: 0.44
      Validation Loss: 1.9814          Validation Accuracy: 0.48
Training shadow model number 4...
[Epoch 0]  [15:04:41] Train Loss: 2.1603    Train Accuracy: 0.29
      Validation Loss: 2.1060          Validation Accuracy: 0.34
[Epoch 1]  [15:05:03] Train Loss: 2.0913    Train Accuracy: 0.36
      Validation Loss: 2.0421          Validation Accuracy: 0.41
[Epoch 2]  [15:05:24] Train Loss: 2.0692    Train Accuracy: 0.38
      Validation Loss: 2.0307          Validation Accuracy: 0.43
[Epoch 3]  [15:05:46] Train Loss: 2.0496    Train Accuracy: 0.41
      Validation Loss: 2.0070          Validation Accuracy: 0.44
[Epoch 4]  [15:06:08] Train Loss: 2.0356    Train Accuracy: 0.42
      Validation Loss: 1.9805          Validation Accuracy: 0.48
Training shadow model number 5...
[Epoch 0]  [15:06:29] Train Loss: 2.1674    Train Accuracy: 0.28
      Validation Loss: 2.0981          Validation Accuracy: 0.36
[Epoch 1]  [15:06:51] Train Loss: 2.1022    Train Accuracy: 0.35
      Validation Loss: 2.0508          Validation Accuracy: 0.40
[Epoch 2]  [15:07:12] Train Loss: 2.0764    Train Accuracy: 0.38
      Validation Loss: 2.0329          Validation Accuracy: 0.43
[Epoch 3]  [15:07:34] Train Loss: 2.0561    Train Accuracy: 0.40
      Validation Loss: 2.0161          Validation Accuracy: 0.44
[Epoch 4]  [15:07:55] Train Loss: 2.0432    Train Accuracy: 0.41
      Validation Loss: 1.9988          Validation Accuracy: 0.46
Training shadow model number 6...
[Epoch 0]  [15:08:17] Train Loss: 2.1597    Train Accuracy: 0.29
      Validation Loss: 2.0712          Validation Accuracy: 0.38
[Epoch 1]  [15:08:38] Train Loss: 2.0865    Train Accuracy: 0.37
      Validation Loss: 2.0306          Validation Accuracy: 0.42
[Epoch 2]  [15:09:00] Train Loss: 2.0580    Train Accuracy: 0.40
      Validation Loss: 2.0123          Validation Accuracy: 0.44
[Epoch 3]  [15:09:21] Train Loss: 2.0379    Train Accuracy: 0.42
      Validation Loss: 1.9866          Validation Accuracy: 0.47
[Epoch 4]  [15:09:43] Train Loss: 2.0244    Train Accuracy: 0.43
      Validation Loss: 1.9722          Validation Accuracy: 0.48
Training shadow model number 7...
[Epoch 0]  [15:10:04] Train Loss: 2.1706    Train Accuracy: 0.28
      Validation Loss: 2.0993          Validation Accuracy: 0.36
[Epoch 1]  [15:10:26] Train Loss: 2.1048    Train Accuracy: 0.35
      Validation Loss: 2.0592          Validation Accuracy: 0.40
[Epoch 2]  [15:10:48] Train Loss: 2.0786    Train Accuracy: 0.38
```

```
      Validation Loss: 2.0313         Validation Accuracy: 0.43
[Epoch 3]  [15:11:09] Train Loss: 2.0607    Train Accuracy: 0.39
      Validation Loss: 2.0034         Validation Accuracy: 0.45
[Epoch 4]  [15:11:31] Train Loss: 2.0425    Train Accuracy: 0.41
      Validation Loss: 1.9897         Validation Accuracy: 0.47
Training shadow model number 8...
[Epoch 0]  [15:11:52] Train Loss: 2.1721    Train Accuracy: 0.27
      Validation Loss: 2.0916         Validation Accuracy: 0.36
[Epoch 1]  [15:12:13] Train Loss: 2.1034    Train Accuracy: 0.35
      Validation Loss: 2.0553         Validation Accuracy: 0.40
[Epoch 2]  [15:12:35] Train Loss: 2.0794    Train Accuracy: 0.38
      Validation Loss: 2.0287         Validation Accuracy: 0.43
[Epoch 3]  [15:12:56] Train Loss: 2.0627    Train Accuracy: 0.39
      Validation Loss: 2.0218         Validation Accuracy: 0.44
[Epoch 4]  [15:13:17] Train Loss: 2.0467    Train Accuracy: 0.41
      Validation Loss: 2.0221         Validation Accuracy: 0.43
Training shadow model number 9...
[Epoch 0]  [15:13:39] Train Loss: 2.1656    Train Accuracy: 0.28
      Validation Loss: 2.0992         Validation Accuracy: 0.35
[Epoch 1]  [15:14:00] Train Loss: 2.1067    Train Accuracy: 0.35
      Validation Loss: 2.0569         Validation Accuracy: 0.40
[Epoch 2]  [15:14:22] Train Loss: 2.0809    Train Accuracy: 0.37
      Validation Loss: 2.0404         Validation Accuracy: 0.42
[Epoch 3]  [15:14:43] Train Loss: 2.0630    Train Accuracy: 0.39
      Validation Loss: 2.0177         Validation Accuracy: 0.44
[Epoch 4]  [15:15:05] Train Loss: 2.0491    Train Accuracy: 0.41
      Validation Loss: 2.0038         Validation Accuracy: 0.45
ShadowModel(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=6272, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

Create datas suitable for tranining the Attacker model these datas are created by using main datas on shadow models that mimic the target model

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Function to combine inputs and outputs into new dataset entries
def create_combined_data(shadow_models, loader, mode):
    for s in range(len(shadow_models)):
        shadow_model = shadow_models[s]
        shadow_model.eval()

        with torch.no_grad():
            for images, true_outputs in loader:
```

```
                images = images.to(device)
                outputs = shadow_model(images).cpu()
                for true_output, output in zip(true_outputs, outputs):
                    true_output, output = true_output.to(device), 
output.to(device)
                    # Ensure true_output and output have the same 
number of dimensions
                    if true_output.dim() == 0:  # Handle scalar tensor 
case
                        true_output = true_output.unsqueeze(0)
                    if output.dim() > 1:  # Handle higher-dimensional 
output tensor
                        output = output.squeeze()  # Adjust dimensions 
as needed
                    combined_output = torch.cat((true_output, output), 
dim=0)
                    yield (combined_output, mode)
                torch.cuda.empty_cache()

# Initialize the dictionary for the shadow models
shadow_models = {}
num_shodow_models = 10
model_dir = './ML_Project/shadow_models/'

# Load shadow models
for i in range(num_shodow_models):
    model_path = os.path.join(model_dir, f'shadow_model_{i}.pth')
    shadow_model = ShadowModel().to(device)
    shadow_model.load_state_dict(torch.load(model_path, 
map_location=device))
    shadow_model.eval()  # Set the model to evaluation mode
    shadow_models[i] = shadow_model

# Example: Print the keys of the shadow_models dictionary to verify
print("Loaded shadow models:", shadow_models.keys())

# Create seen data
combined_in_data1 = list(create_combined_data(shadow_models, 
train_loader, 1))
combined_in_data2 = list(create_combined_data(shadow_models, 
val_loader, 1))
combined_in_data = combined_in_data1 + combined_in_data2

# Create unseen data
combined_out_data = list(create_combined_data(shadow_models, 
test_loader, -1))
```

```
Loaded shadow models: dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Print a few sample datas to see their looks

```
# Example: Print first few entries of each data
for i in range(5):
    input_data, label = combined_in_data[i]
    print(f"Input: {input_data}, Label: {label}")

for i in range(5):
    input_data, label = combined_out_data[i]
    print(f"Input: {input_data}, Label: {label}")

print(len(combined_in_data))
print(len(combined_out_data))
```

```
Input: tensor([1.0000e+00, 5.7000e-12, 9.9997e-01, 1.3633e-10,
1.3945e-11, 1.7988e-17,
        2.9970e-17, 1.3210e-15, 8.0076e-18, 7.0411e-07, 2.4437e-05],
       device='cuda:0'), Label: 1
Input: tensor([9.0000e+00, 7.3594e-11, 7.5609e-06, 5.1034e-12,
6.0145e-07, 1.1050e-13,
        7.1933e-10, 9.3258e-08, 3.8030e-05, 1.1856e-08, 9.9995e-01],
       device='cuda:0'), Label: 1
Input: tensor([7.0000e+00, 7.6815e-04, 1.5189e-02, 6.7124e-02,
5.4361e-01, 5.5002e-02,
        5.8696e-02, 2.2467e-01, 2.0750e-02, 1.8769e-03, 1.2312e-02],
       device='cuda:0'), Label: 1
Input: tensor([8.0000e+00, 8.4091e-01, 1.1952e-02, 9.3768e-05,
8.8409e-02, 1.0779e-06,
        5.8708e-04, 4.3390e-08, 2.5747e-05, 5.2637e-02, 5.3849e-03],
       device='cuda:0'), Label: 1
Input: tensor([0.0000, 0.1549, 0.0007, 0.0689, 0.0381, 0.1775, 0.0926,
0.0026, 0.4606,
        0.0006, 0.0035], device='cuda:0'), Label: 1
Input: tensor([7.0000e+00, 1.5993e-08, 3.2558e-14, 1.3142e-10,
6.8014e-15, 4.1251e-10,
        4.0823e-10, 1.8324e-13, 1.0000e+00, 2.0293e-25, 1.6176e-13],
       device='cuda:0'), Label: -1
Input: tensor([3.0000e+00, 6.6074e-23, 3.2279e-09, 2.8033e-15,
2.9297e-06, 2.2770e-14,
        1.1411e-13, 1.0000e+00, 1.0047e-11, 5.5602e-19, 5.8257e-08],
       device='cuda:0'), Label: -1
Input: tensor([2.0000e+00, 8.4727e-09, 3.1985e-05, 5.3729e-05,
1.1257e-04, 3.9741e-04,
        5.6721e-05, 9.9350e-01, 5.7841e-03, 4.8504e-10, 5.8702e-05],
       device='cuda:0'), Label: -1
Input: tensor([2.0000e+00, 9.1457e-14, 2.1005e-20, 2.2176e-06,
7.0072e-04, 2.4825e-09,
        9.9930e-01, 1.8728e-16, 1.1753e-09, 4.9784e-16, 6.7300e-21],
       device='cuda:0'), Label: -1
Input: tensor([3.0000e+00, 3.1744e-04, 2.0695e-10, 2.2470e-05,
```

```
8.1231e-01, 3.4953e-09,
        1.8735e-01, 4.6857e-13, 2.1038e-06, 7.5421e-07, 1.3674e-09],
       device='cuda:0'), Label: -1
500000
100000
```

create final dataset for the attacker model

```
# Step 1: Combine the seen and unseen data
def get_shadow_datasets(combined_in_data, combined_out_data):
    # print(len(combined_in_data))
    # print(len(combined_out_data))
    combined_dataset = combined_in_data + combined_out_data
    shuffle(combined_dataset)
#     print(len(combined_dataset))

    # Create DataLoaders
    batch_size = 64

    train_shadow_loader = DataLoader(combined_dataset[0:48000],
batch_size=batch_size, shuffle=True)
    test_shadow_loader = DataLoader(combined_dataset[48000:60000],
batch_size=batch_size, shuffle=True)
#     print(len(train_shadow_loader))
#     print(test_shadow_loader.dataset[0])
    return train_shadow_loader, test_shadow_loader

train_shadow_loader, test_shadow_loader =
get_shadow_datasets(combined_in_data, combined_out_data)

# Example: Print sizes to verify
# print(f"Total Combined Dataset Size: {len(combined_dataset)}")
print(f"Training Data Size: {len(train_shadow_loader)}")
print(f"Test Data Size: {len(test_shadow_loader)}")

Training Data Size: 750
Test Data Size: 188
```

see a sample data

```
print(f'sample train data: {train_shadow_loader.dataset[0]}')

sample train data: (tensor([9.0000e+00, 6.6451e-08, 1.0000e+00,
7.0550e-14, 1.4830e-12, 1.4231e-24,
        1.7990e-18, 2.1469e-22, 1.5359e-19, 1.2996e-09, 1.1756e-06],
       device='cuda:0'), 1)
```

preprocess data to train linear regression model as attacker model

```
def extract_data_and_labels(loader):
    data = []
    labels = []
    for x,y in loader:
#         print(x.cpu().numpy().shape)
        data.append(x.cpu().numpy())  # Assuming DataLoader returns
PyTorch tensors
        labels.append(y.cpu().numpy())
    data = np.concatenate(data, axis=0)
    labels = np.concatenate(labels, axis=0)
    # normalizing
    scaler = StandardScaler()
    scaler.fit(data)
    scaled_data = scaler.transform(data)
    return scaled_data, labels

X_train, y_train = extract_data_and_labels(train_shadow_loader)
X_test, y_test = extract_data_and_labels(test_shadow_loader)
print(X_train.shape)
print(y_train.shape)

(48000, 11)
(48000,)
```

simple logistic regression model as attacker

```
lr = LR()
lr.fit(X_train, y_train)
# Evaluate the model
y_pred = lr.predict(X_test)
with open('advanced_attack_model.pkl', 'wb') as file:
    pickle.dump(lr, file)
```

check attacker model accuracy

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))

Accuracy: 83.48%
```

create private shadow models to mimic private model

```
# Train 10 different shadow models, each with its corresponding label
dataset
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
private_shadow_models = {}
regularization_strength = 2e-3
num_epochs = 5
criterion = nn.CrossEntropyLoss()
```

```python
for label in range(10):
    private_shadow_model = Private_ShadowModel().to(device)
    print(f"Training shadow model for label {label}...")
    model_train(private_shadow_model, train_loader, val_loader,
criterion, num_epochs, regularization_strength)
    private_shadow_models[label] = private_shadow_model

# Save the shadow models
for label in range(10):
    torch.save(private_shadow_models[label].state_dict(),
f'private_shadow_model_{label}.pth')

# Example: Print summary of one shadow model
print(private_shadow_models[0])
```

```
Training shadow model for label 0...
[Epoch 0]  [16:09:50] Train Loss: 2.1372    Train Accuracy: 0.29
      Validation Loss: 1.9093          Validation Accuracy: 0.41
[Epoch 1]  [16:10:12] Train Loss: 1.9859    Train Accuracy: 0.37
      Validation Loss: 1.8262          Validation Accuracy: 0.44
[Epoch 2]  [16:10:34] Train Loss: 1.9207    Train Accuracy: 0.41
      Validation Loss: 1.7706          Validation Accuracy: 0.47
[Epoch 3]  [16:10:55] Train Loss: 1.8877    Train Accuracy: 0.43
      Validation Loss: 1.7127          Validation Accuracy: 0.49
[Epoch 4]  [16:11:17] Train Loss: 1.8601    Train Accuracy: 0.44
      Validation Loss: 1.6945          Validation Accuracy: 0.50
Training shadow model for label 1...
[Epoch 0]  [16:11:39] Train Loss: 2.1989    Train Accuracy: 0.23
      Validation Loss: 2.0859          Validation Accuracy: 0.32
[Epoch 1]  [16:12:01] Train Loss: 2.0627    Train Accuracy: 0.33
      Validation Loss: 1.8970          Validation Accuracy: 0.42
[Epoch 2]  [16:12:23] Train Loss: 1.9674    Train Accuracy: 0.38
      Validation Loss: 1.8254          Validation Accuracy: 0.45
[Epoch 3]  [16:12:44] Train Loss: 1.9084    Train Accuracy: 0.41
      Validation Loss: 1.7758          Validation Accuracy: 0.48
[Epoch 4]  [16:13:06] Train Loss: 1.8745    Train Accuracy: 0.43
      Validation Loss: 1.7083          Validation Accuracy: 0.50
Training shadow model for label 2...
[Epoch 0]  [16:13:28] Train Loss: 2.1786    Train Accuracy: 0.25
      Validation Loss: 2.0067          Validation Accuracy: 0.36
[Epoch 1]  [16:13:50] Train Loss: 2.0369    Train Accuracy: 0.33
      Validation Loss: 1.8584          Validation Accuracy: 0.41
[Epoch 2]  [16:14:11] Train Loss: 1.9654    Train Accuracy: 0.38
      Validation Loss: 1.8268          Validation Accuracy: 0.44
[Epoch 3]  [16:14:33] Train Loss: 1.9321    Train Accuracy: 0.40
      Validation Loss: 1.7750          Validation Accuracy: 0.46
[Epoch 4]  [16:14:55] Train Loss: 1.8941    Train Accuracy: 0.42
      Validation Loss: 1.6903          Validation Accuracy: 0.49
Training shadow model for label 3...
```

```
[Epoch 0]  [16:15:17] Train Loss: 2.1837    Train Accuracy: 0.24
     Validation Loss: 1.9609         Validation Accuracy: 0.38
[Epoch 1]  [16:15:38] Train Loss: 2.0241    Train Accuracy: 0.35
     Validation Loss: 1.8912         Validation Accuracy: 0.41
[Epoch 2]  [16:16:00] Train Loss: 1.9459    Train Accuracy: 0.39
     Validation Loss: 1.7790         Validation Accuracy: 0.46
[Epoch 3]  [16:16:22] Train Loss: 1.9075    Train Accuracy: 0.41
     Validation Loss: 1.7438         Validation Accuracy: 0.48
[Epoch 4]  [16:16:44] Train Loss: 1.8833    Train Accuracy: 0.42
     Validation Loss: 1.7376         Validation Accuracy: 0.48
Training shadow model for label 4...
[Epoch 0]  [16:17:06] Train Loss: 2.1432    Train Accuracy: 0.28
     Validation Loss: 1.9221         Validation Accuracy: 0.39
[Epoch 1]  [16:17:27] Train Loss: 1.9894    Train Accuracy: 0.36
     Validation Loss: 1.8075         Validation Accuracy: 0.44
[Epoch 2]  [16:17:49] Train Loss: 1.9207    Train Accuracy: 0.40
     Validation Loss: 1.7316         Validation Accuracy: 0.49
[Epoch 3]  [16:18:11] Train Loss: 1.8790    Train Accuracy: 0.42
     Validation Loss: 1.7218         Validation Accuracy: 0.48
[Epoch 4]  [16:18:33] Train Loss: 1.8531    Train Accuracy: 0.44
     Validation Loss: 1.7076         Validation Accuracy: 0.50
Training shadow model for label 5...
[Epoch 0]  [16:18:55] Train Loss: 2.1931    Train Accuracy: 0.24
     Validation Loss: 2.0049         Validation Accuracy: 0.36
[Epoch 1]  [16:19:17] Train Loss: 2.0276    Train Accuracy: 0.34
     Validation Loss: 1.8440         Validation Accuracy: 0.43
[Epoch 2]  [16:19:38] Train Loss: 1.9561    Train Accuracy: 0.38
     Validation Loss: 1.7668         Validation Accuracy: 0.45
[Epoch 3]  [16:20:00] Train Loss: 1.9048    Train Accuracy: 0.40
     Validation Loss: 1.7617         Validation Accuracy: 0.47
[Epoch 4]  [16:20:22] Train Loss: 1.8740    Train Accuracy: 0.42
     Validation Loss: 1.7026         Validation Accuracy: 0.51
Training shadow model for label 6...
[Epoch 0]  [16:20:44] Train Loss: 2.1431    Train Accuracy: 0.28
     Validation Loss: 1.9864         Validation Accuracy: 0.37
[Epoch 1]  [16:21:06] Train Loss: 2.0197    Train Accuracy: 0.36
     Validation Loss: 1.8945         Validation Accuracy: 0.44
[Epoch 2]  [16:21:27] Train Loss: 1.9529    Train Accuracy: 0.40
     Validation Loss: 1.7807         Validation Accuracy: 0.46
[Epoch 3]  [16:21:49] Train Loss: 1.8992    Train Accuracy: 0.42
     Validation Loss: 1.7481         Validation Accuracy: 0.48
[Epoch 4]  [16:22:11] Train Loss: 1.8673    Train Accuracy: 0.43
     Validation Loss: 1.7341         Validation Accuracy: 0.52
Training shadow model for label 7...
[Epoch 0]  [16:22:33] Train Loss: 2.1520    Train Accuracy: 0.27
     Validation Loss: 1.9443         Validation Accuracy: 0.37
[Epoch 1]  [16:22:55] Train Loss: 2.0081    Train Accuracy: 0.35
     Validation Loss: 1.8844         Validation Accuracy: 0.39
[Epoch 2]  [16:23:17] Train Loss: 1.9596    Train Accuracy: 0.37
```

```
      Validation Loss: 1.8433         Validation Accuracy: 0.42
[Epoch 3]  [16:23:38] Train Loss: 1.9296    Train Accuracy: 0.39
      Validation Loss: 1.8060         Validation Accuracy: 0.44
[Epoch 4]  [16:24:00] Train Loss: 1.9191    Train Accuracy: 0.39
      Validation Loss: 1.7793         Validation Accuracy: 0.45
Training shadow model for label 8...
[Epoch 0]  [16:24:22] Train Loss: 2.1545    Train Accuracy: 0.26
      Validation Loss: 2.0240         Validation Accuracy: 0.32
[Epoch 1]  [16:24:44] Train Loss: 2.0493    Train Accuracy: 0.32
      Validation Loss: 1.9292         Validation Accuracy: 0.39
[Epoch 2]  [16:25:06] Train Loss: 1.9973    Train Accuracy: 0.35
      Validation Loss: 1.8964         Validation Accuracy: 0.40
[Epoch 3]  [16:25:27] Train Loss: 1.9826    Train Accuracy: 0.37
      Validation Loss: 1.8665         Validation Accuracy: 0.41
[Epoch 4]  [16:25:49] Train Loss: 1.9543    Train Accuracy: 0.37
      Validation Loss: 1.8253         Validation Accuracy: 0.42
Training shadow model for label 9...
[Epoch 0]  [16:26:11] Train Loss: 2.1566    Train Accuracy: 0.27
      Validation Loss: 1.9817         Validation Accuracy: 0.35
[Epoch 1]  [16:26:33] Train Loss: 2.0362    Train Accuracy: 0.34
      Validation Loss: 1.9336         Validation Accuracy: 0.40
[Epoch 2]  [16:26:54] Train Loss: 1.9821    Train Accuracy: 0.37
      Validation Loss: 1.8442         Validation Accuracy: 0.43
[Epoch 3]  [16:27:16] Train Loss: 1.9381    Train Accuracy: 0.40
      Validation Loss: 1.7855         Validation Accuracy: 0.47
[Epoch 4]  [16:27:38] Train Loss: 1.9015    Train Accuracy: 0.42
      Validation Loss: 1.7814         Validation Accuracy: 0.47
Private_ShadowModel(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.25, inplace=False)
  (dropout2): Dropout(p=0.5, inplace=False)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc1): Linear(in_features=6272, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

read saved private shadow models and prepare data for attacker model

```
# Initialize the dictionary for the shadow models
private_shadow_models = {}

model_dir = './ML_Project/private_shadow_models/'

# Load shadow models
for i in range(10):
    model_path = os.path.join(model_dir,
f'private_shadow_model_{i}.pth')
```

```
    private_shadow_model = Private_ShadowModel().to(device)
    private_shadow_model.load_state_dict(torch.load(model_path,
map_location=device))
    private_shadow_model.eval()  # Set the model to evaluation mode
    private_shadow_models[i] = private_shadow_model

# Example: Print the keys of the shadow_models dictionary to verify
print("Loaded private shadow models:", private_shadow_models.keys())

# Create seen data
private_combined_in_data1 =
list(create_combined_data(private_shadow_models, train_loader, 1))
private_combined_in_data2 =
list(create_combined_data(private_shadow_models, val_loader, 1))
private_combined_in_data = private_combined_in_data1 +
private_combined_in_data2

# Create unseen data
private_combined_out_data =
list(create_combined_data(private_shadow_models, test_loader, -1))
```

```
Loaded private shadow models: dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8,
9])
```

see samples of final data

```
# Example: Print first few entries of each data
for i in range(5):
    input_data, label = private_combined_in_data[i]
    print(f"Input: {input_data}, Label: {label}")

for i in range(5):
    input_data, label = private_combined_out_data[i]
    print(f"Input: {input_data}, Label: {label}")

print(len(private_combined_in_data))
print(len(private_combined_out_data))
```

```
Input: tensor([9.0000, 0.0000, 2.4825, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000,
        0.0000, 4.0832], device='cuda:0'), Label: 1
Input: tensor([2.0000, 0.0000, 0.0000, 0.0000, 1.0437, 0.0000, 1.2690,
0.0000, 0.0000,
        0.0000, 0.0000], device='cuda:0'), Label: 1
Input: tensor([7.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 1.0462,
        0.0000, 1.6653], device='cuda:0'), Label: 1
Input: tensor([7.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7650, 0.8787,
0.0000, 0.0000,
```

```
        0.0000, 0.0000], device='cuda:0'), Label: 1
Input: tensor([5.0000, 3.9677, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000,
        2.5984, 0.0000], device='cuda:0'), Label: 1
Input: tensor([4.0000, 0.0000, 0.0000, 0.0000, 1.4472, 0.0000, 1.6715,
0.0000, 0.0000,
        0.0000, 0.0000], device='cuda:0'), Label: -1
Input: tensor([5.0000, 0.0000, 0.0000, 0.0000, 1.4383, 0.0000, 2.0437,
0.0000, 0.0000,
        0.0000, 0.0000], device='cuda:0'), Label: -1
Input: tensor([9.0000, 0.0000, 3.6093, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000,
        0.0000, 3.6882], device='cuda:0'), Label: -1
Input: tensor([3.0000, 0.0000, 0.0000, 0.0000, 1.9907, 0.0000, 2.7556,
0.0000, 0.0000,
        0.0000, 0.0000], device='cuda:0'), Label: -1
Input: tensor([8.0000, 2.7107, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000,
        4.6987, 0.0000], device='cuda:0'), Label: -1
500000
100000
```

combine in and out datas

```
private_train_shadow_loader, private_test_shadow_loader =
get_shadow_datasets(private_combined_in_data,
private_combined_out_data)

# Example: Print sizes to verify
# print(f"Total Combined Dataset Size: {len(combined_dataset)}")
print(f"Training Data Size: {len(private_train_shadow_loader)}")
print(f"Test Data Size: {len(private_test_shadow_loader)}")

Training Data Size: 750
Test Data Size: 188
```

private attacker model sample train data

```
print(f'sample train data: {private_train_shadow_loader.dataset[0]}')

sample train data: (tensor([9.0000, 0.0000, 1.1647, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 1.7250], device='cuda:0'), -1)
```

Seperate parts of data for linear regression model

```
# Extract data and labels from DataLoader
private_X_train, private_y_train =
extract_data_and_labels(private_train_shadow_loader)
```

```
private_X_test, private_y_test =
extract_data_and_labels(private_test_shadow_loader)
```

Train linear regression attacker model for private dataset

```
lr = LR()
lr.fit(private_X_train, private_y_train)
# Evaluate the model
private_y_pred = lr.predict(private_X_test)
with open('advanced_private_attack_model.pkl', 'wb') as file:
    pickle.dump(lr, file)

# # Create an instance of the SVM classifier with a linear kernel
# clf = SVC(kernel='linear', C=10)

# # Train the SVM classifier
# clf.fit(X_train, y_train)

# # Make predictions with the trained model
# predictions = clf.predict(X_test)
```

check accuracy of private attacker model

```
accuracy = accuracy_score(private_y_test, private_y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
# accuracy = accuracy_score(y_test, predictions)
# print("Accuracy:", accuracy)

Accuracy: 82.88%
```

3 Membership Inference Attack

Simulation Question 8.

Attempt to train an attacker model for the given private model (private_model.pth). We will test it on our dataset during the online presentation session. A competitive bonus point is available for the best performance.

Given code

```
from torchvision import models
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
```

```python
class CIFAR10Classifier(nn.Module):
  def __init__(self):
    super(CIFAR10Classifier, self).__init__()
    self.conv1 = nn.Conv2d(3, 16, 3, 1)
    self.conv2 = nn.Conv2d(16, 32, 3, 1)
    self.dropout1 = nn.Dropout2d(0.25)
    self.dropout2 = nn.Dropout2d(0.5)
    self.fc1 = nn.Linear(6272, 64)
    self.fc2 = nn.Linear(64, 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)
    return x
```

Going to complete this cell

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.datasets import CIFAR10
from torchvision import transforms
from torch.utils.data import Subset, DataLoader, TensorDataset
from sklearn.metrics import confusion_matrix, precision_score,
recall_score ,f1_score
from sklearn.linear_model import LogisticRegression

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = CIFAR10Classifier()
state_dict = torch.load("model_state_dict.pth", map_location=device)
new_state_dict = {key.replace('_module.', ''): value for key, value in
state_dict.items()}
model.load_state_dict(new_state_dict)
model.to(device)
model.eval()

transform = transforms.Compose([
```

```
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

DATA_ROOT = '../cifar10'
BATCH_SIZE = 64

# Load the indices from list.txt
indices_file = 'list.txt' ############
with open(indices_file, 'r') as f:
    indices = [int(line.strip()) for line in f]

full_train_dataset = CIFAR10(root=DATA_ROOT, train=True,
download=True, transform=transform)
test_dataset = CIFAR10(root=DATA_ROOT, train=False, download=True,
transform=transform)

train_indices_set = set(indices)
all_indices = set(range(len(full_train_dataset)))
other_indices = list(all_indices - train_indices_set)

train_dataset = Subset(full_train_dataset, indices[:len(indices)//2])
###########
other_dataset = Subset(full_train_dataset, other_indices)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=False)
other_loader = DataLoader(other_dataset, batch_size=BATCH_SIZE,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False)

# Create labels
train_labels = torch.ones(len(train_dataset)).to(device)
other_labels = torch.zeros(len(other_dataset)).to(device)
test_labels = torch.zeros(len(test_dataset)).to(device)
####################################
#if you have an attacker model for each class, modify the above code.
####################################

def extract_features(model, dataloader):
    model.eval()
    features = []
    with torch.no_grad():
        for data in dataloader:
            inputs, _ = data
            inputs = inputs.to(device)
            outputs = model(inputs)
            features.append(outputs)
```

```
    return torch.cat(features).to(device)

train_features = extract_features(model, train_loader)
other_features = extract_features(model, other_loader)
test_features = extract_features(model, test_loader)


combined_features = torch.cat((train_features, other_features,
test_features))
combined_labels = torch.cat((train_labels, other_labels, test_labels))


new_dataset = TensorDataset(combined_features, combined_labels)
new_loader = DataLoader(new_dataset, batch_size=BATCH_SIZE,
shuffle=True)

#load your attacker model
#############################################
# attackers created in question 6
with open('basic_attack_model.pkl', 'rb') as file:
    basic_attack_model = pickle.load(file)
with open('basic_private_attack_model.pkl', 'rb') as file:
    basic_private_attack_model = pickle.load(file)
# attackers created in question 7
with open('advanced_attack_model.pkl', 'rb') as file:
    advanced_attack_model = pickle.load(file)
with open('advanced_private_attack_model.pkl', 'rb') as file:
    advanced_private_attack_model = pickle.load(file)

# Calculate training accuracy, confusion matrix, precision, and recall
binary_classifier.eval()
all_labels = []
all_predicted = []
correct = 0
total = 0

with torch.no_grad():
    for features, labels in new_loader:
        features, labels = features.to(device), labels.to(device)
        outputs = attacker(features).squeeze()
        predicted = (outputs > 0.5).float()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_labels.extend(labels.cpu().numpy())
        all_predicted.extend(predicted.cpu().numpy())

accuracy = correct / total
print(f'Training Accuracy: {accuracy:.4f}')

cm = confusion_matrix(all_labels, all_predicted)
```

```
precision = precision_score(all_labels, all_predicted)
recall = recall_score(all_labels, all_predicted)
f1 = f1_score(all_labels, all_predicted)

print(f'Confusion Matrix:\n{cm}')
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```