In [1]:
```
from google.cloud import aiplatform
from google.cloud import bigquery
from google.cloud import storage

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import pandas as pd
from torch.utils.data import Dataset, DataLoader

import time
```

In [21]:
```
# Set your GCP project and location
project = "teak-listener-398917"
location = "us-central1"

# Set your Vertex AI Dataset ID
dataset_id = "1112029567658229760"

# Set your Vertex AI Model Registry display name
model_display_name = "custom-pytorch-model"

# Bucket for saving the model
custom_model_bucket_name = "custom-models-gcp-bucket"
model_filename = "my-custom-pytorch-model.pkl"
model_bucket_path = "gs://custom-models-gcp-bucket"

# Initialize Vertex AI client
aiplatform.init(project=project, location=location)
```

In [3]:
```
# Set up BigQuery client
bq_client = bigquery.Client(project)
```

In [4]:
```
query_sql = "SELECT * FROM `teak-listener-398917.creditcardtransactions.creditcardd

df = bq_client.query(query_sql).to_dataframe()
```

In [5]:
```
df.head()
```

Out[5]:

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.425111 | -0.305729 | -0.133373 | 0.394062 | -0.610187 | 0.752437 | 0.069425 | 0.097934 | -0 |
| **1** | -2.072985 | 2.155252 | -1.937737 | 1.608221 | -2.356588 | -0.643348 | -2.649083 | -0.541339 | -2 |
| **2** | 0.339219 | 0.395395 | -0.719424 | 0.961543 | 0.299822 | -0.052440 | -0.305279 | 0.068382 | -0 |
| **3** | -0.891892 | 0.140255 | -0.472645 | 0.810693 | -0.427782 | -0.145407 | -0.641984 | 0.050744 | 0 |
| **4** | 1.026787 | -0.274050 | 0.260901 | -0.142720 | 0.457773 | 0.360629 | 0.449987 | -0.144174 | 0 |

5 rows × 30 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 30 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   V1      568630 non-null  float64
 1   V2      568630 non-null  float64
 2   V3      568630 non-null  float64
 3   V4      568630 non-null  float64
 4   V5      568630 non-null  float64
 5   V6      568630 non-null  float64
 6   V7      568630 non-null  float64
 7   V8      568630 non-null  float64
 8   V9      568630 non-null  float64
 9   V10     568630 non-null  float64
 10  V11     568630 non-null  float64
 11  V12     568630 non-null  float64
 12  V13     568630 non-null  float64
 13  V14     568630 non-null  float64
 14  V15     568630 non-null  float64
 15  V16     568630 non-null  float64
 16  V17     568630 non-null  float64
 17  V18     568630 non-null  float64
 18  V19     568630 non-null  float64
 19  V20     568630 non-null  float64
 20  V21     568630 non-null  float64
 21  V22     568630 non-null  float64
 22  V23     568630 non-null  float64
 23  V24     568630 non-null  float64
 24  V25     568630 non-null  float64
 25  V26     568630 non-null  float64
 26  V27     568630 non-null  float64
 27  V28     568630 non-null  float64
 28  Amount  568630 non-null  float64
 29  Class   568630 non-null  Int64  
dtypes: Int64(1), float64(29)
memory usage: 130.7 MB
```

In [7]:
```
num_null = df.isnull().sum().sum()

print("Number of null values:", num_null)

if num_null > 0:
    df.dropna(axis=0)
```

```
Number of null values: 0
```

In [8]:
```
df["Amount"].describe()
```

Out[8]:
```
count    568630.000000
mean      12041.957635
std        6919.644449
min          50.010000
25%        6054.892500
50%       12030.150000
75%       18036.330000
max       24039.930000
Name: Amount, dtype: float64
```

In [9]:
```
# Separate features and target variable
X = df.drop("Class", axis=1)
y = df["Class"]
```

In [10]:
```
y.value_counts()
```

Out[10]:
```
1    284315
0    284315
Name: Class, dtype: Int64
```

In [11]:
```
# Perform stratified train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y
```

In [12]:
```
# Convert data to PyTorch tensors with 'long' data type for target labels
X_train_tensor = torch.tensor(X_train.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)  # Use torch.long f
X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)
```

In [13]:
```
# Create DataLoader for training and testing
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

In [14]:
```
# Define a GRU model
class GRUModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(GRUModel, self).__init__()
        self.gru = nn.GRU(input_size, hidden_size, num_layers=3, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()
```

```
    def forward(self, x):
        _, h_n = self.gru(x)
        x = h_n[-1, :, :]  # Take the hidden state from the last time step
        x = self.fc(x)
        x = self.sigmoid(x)
        return x

# Instantiate the model with desired sizes and move to GPU
input_size = X_train.shape[1]
hidden_size = 256  # You can adjust this
output_size = 1  # For binary classification
model = GRUModel(input_size, hidden_size, output_size)

# Define loss function and optimizer
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
criterion.to(device)


training_start = time.time()

# Training loop
num_epochs = 10

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    correct_predictions = 0
    total_samples = 0

    for inputs, labels in train_loader:
        optimizer.zero_grad()

        # Assuming each row is considered a sequence (sequence length = 1)
        inputs = inputs.unsqueeze(1)

        outputs = model(inputs)
        loss = criterion(outputs.squeeze(), labels.float())
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

        _, preds = torch.round(outputs).long().squeeze(), labels
        correct_predictions += torch.sum(preds == labels).item()
        total_samples += labels.size(0)

    # Calculate and print average loss and accuracy for the epoch
    average_loss = total_loss / len(train_loader)
    accuracy = correct_predictions / total_samples
    print(f"Epoch {epoch + 1}/{num_epochs} | Loss: {average_loss:.4f} | Accuracy: {

training_end = time.time()
```

```
print("\n\nTotal training time (s):", (training_end - training_start))
```

```
Epoch 1/10 | Loss: 0.6034 | Accuracy: 100.00%
Epoch 2/10 | Loss: 0.3966 | Accuracy: 100.00%
Epoch 3/10 | Loss: 0.3168 | Accuracy: 100.00%
Epoch 4/10 | Loss: 0.2945 | Accuracy: 100.00%
Epoch 5/10 | Loss: 0.2809 | Accuracy: 100.00%
Epoch 6/10 | Loss: 0.2958 | Accuracy: 100.00%
Epoch 7/10 | Loss: 0.2664 | Accuracy: 100.00%
Epoch 8/10 | Loss: 0.2666 | Accuracy: 100.00%
Epoch 9/10 | Loss: 0.2536 | Accuracy: 100.00%
Epoch 10/10 | Loss: 0.2363 | Accuracy: 100.00%


Total training time (s): 1508.1906161308289
```

In [15]:

```
# Evaluation
model.eval()
all_preds = []
true_labels = []

with torch.no_grad():
    for inputs, labels in test_loader:
        # Assuming each row is considered a sequence (sequence length = 1)
        inputs = inputs.unsqueeze(1)

        outputs = model(inputs)
        preds = torch.round(outputs).long().squeeze()
        all_preds.extend(preds.tolist())
        true_labels.extend(labels.tolist())

# Convert predictions back to original labels if needed
# predicted_labels = label_encoder.inverse_transform(all_preds)

# Calculate accuracy
accuracy = accuracy_score(true_labels, all_preds)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Create confusion matrix
conf_matrix = confusion_matrix(true_labels, all_preds)
print("Confusion Matrix:")
print(conf_matrix)

# Create classification report
class_report = classification_report(true_labels, all_preds, target_names=["Class 0
print("Classification Report:")
print(class_report)
```

```
Test Accuracy: 94.11%
Confusion Matrix:
[[56618   245]
 [ 6451 50412]]
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.90      1.00      0.94     56863
     Class 1       1.00      0.89      0.94     56863

    accuracy                           0.94    113726
   macro avg       0.95      0.94      0.94    113726
weighted avg       0.95      0.94      0.94    113726
```

In [22]:
```
# Save model
torch.save(model.state_dict(), model_filename)
```

In [23]:
```
# Upload the model to Cloud Storage

client = storage.Client()
bucket = client.bucket(custom_model_bucket_name)
blob = bucket.blob(model_filename)
blob.upload_from_filename(model_filename)
```

In [ ]:
```

```