# Quantum-Classical Hybrid Neural Network for Intrusion Detection

## Objective

The purpose of this project is to:  
- Design and implement a quantum-classical hybrid neural network.  
- Classify entries in the KDDCup dataset into two categories:  
 1. Normal (Good): Representing legitimate activities within the system.  
 2. Intrusion (Bad): Indicating suspicious or malicious activities that may compromise security.  
  
This project utilizes Qiskit for simulating quantum circuits and PyTorch for building and training the classical components of the hybrid model.

## What You Will Learn

1. Intrusion Detection:  
 - Understand the concept of intrusion detection and its importance in cybersecurity.  
 - Explore the types of attacks present in the dataset and how to classify them.  
2. KDDCup Dataset:  
 - Learn the structure of the dataset, including its features and labels.  
 - Understand preprocessing steps like normalization and encoding.  
3. Hybrid Neural Networks:  
 - Combine classical and quantum layers in a single model.  
 - Optimize the hybrid model for binary classification tasks.  
4. Quantum Computing Basics:  
 - Understand quantum gates (RY, Hadamard) and their role in machine learning.  
 - Simulate quantum circuits using Qiskit’s Aer simulator.

## Dataset: KDDCup

The KDDCup dataset is a widely used benchmark dataset for intrusion detection tasks. Each entry represents a network connection, described by various attributes.

### Features

The dataset contains 41 features:  
1. Basic Features:  
 - Protocol type (e.g., TCP, UDP, ICMP).  
 - Duration of the connection.  
 - Service type (e.g., HTTP, FTP).  
2. Content Features:  
 - Number of failed logins.  
 - Successful login indicator.  
3. Traffic Features:  
 - Number of connections to the same host in the past 2 seconds.  
 - Number of SYN errors.

### Class Labels

- Normal (0): Legitimate connections.  
- Intrusion (1): Malicious activities such as DoS or unauthorized access.

## Steps to Complete the Project

### Step 1: Dataset Preprocessing

1. Import the Dataset: Load the kddcup.csv file using pandas.  
2. Handle Missing Values: Replace any missing or invalid data with appropriate values.  
3. Encode Labels: Use LabelEncoder to convert the target column into numeric values (0 for Normal, 1 for Intrusion).  
4. Normalize Features: Scale numerical features using StandardScaler.  
5. Split the Dataset: Divide the data into training (60%), validation (20%), and test (20%) sets.

### Step 2: Quantum Circuit Design

1. Learn About Quantum Gates: Understand how RY gates are parameterized by input features.  
2. Build a Parameterized Quantum Circuit: Use Qiskit to design a quantum circuit with 2 qubits.  
3. Simulate the Circuit: Use Qiskit’s Aer simulator to simulate the quantum circuit.

### Step 3: Hybrid Neural Network Design

1. Define Classical Layers: Use fully connected layers with ReLU activation for initial feature extraction.  
2. Integrate the Quantum Layer: Use the output of the quantum circuit as a layer in the neural network.  
3. Add Output Layer: Use a softmax layer to classify inputs into Normal (0) or Intrusion (1).

### Step 4: Train the Model

1. Define Loss Function: Use Negative Log Likelihood (NLLLoss) to compute the error.  
2. Optimize Parameters: Use the Adam optimizer to adjust weights in classical layers and parameters in the quantum circuit.  
3. Monitor Validation Loss: Evaluate the model on the validation set after each epoch to prevent overfitting.

### Step 5: Evaluate the Model

1. Test the Model: Use the test set to calculate the model’s accuracy.  
2. Analyze Results: Identify areas where the model performs well and where it struggles.  
3. Suggest Improvements: Consider adding more quantum or classical layers, or preprocessing techniques to improve performance.

## Resources

1. Qiskit Documentation: https://qiskit.org/documentation/  
2. PyTorch Documentation: https://pytorch.org/docs/  
3. KDDCup Dataset: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

## Jupyter Notebook

The provided Jupyter Notebook, .ipynb, contains the complete implementation of the Quantum-Classical Hybrid Neural Network. Below is a detailed explanation of what each section in the notebook does.

### Notebook Overview

The notebook demonstrates how to implement a hybrid neural network for binary classification of the KDDCup dataset. It combines classical deep learning techniques with quantum computing using Qiskit and PyTorch.

#### 1. Introduction

This section introduces:  
- The concept of hybrid neural networks.  
- An explanation of quantum computing and its integration with classical models.  
- A brief overview of Qiskit and PyTorch.

#### 2. Installing Dependencies

The notebook installs the required versions of Qiskit and Qiskit-Aer to ensure compatibility for simulating quantum circuits. It also imports all necessary libraries such as PyTorch, pandas, and scikit-learn.

#### 3. Dataset Preprocessing

This section:  
1. Loads the KDDCup dataset into a pandas DataFrame.  
2. Handles missing values and encodes the target column using LabelEncoder.  
3. Normalizes numerical features using StandardScaler.  
4. Splits the dataset into training, validation, and test sets.

#### 4. Quantum Circuit Design

In this section, a parameterized quantum circuit is created using Qiskit. The circuit:  
- Uses `RY` rotation gates, where parameters are updated during training.  
- Includes measurement operations to extract class probabilities.  
- Is simulated using Qiskit’s Aer simulator.

#### 5. Hybrid Neural Network Definition

A PyTorch-based hybrid neural network is implemented. It includes:  
- Classical layers for feature extraction using fully connected layers with ReLU activation.  
- A quantum layer that processes features using the parameterized quantum circuit.  
- A final softmax layer for binary classification.

#### 6. Training and Evaluation

This section trains the hybrid model using the training dataset and evaluates it on the test dataset. Key steps include:  
- Using Negative Log Likelihood (NLLLoss) as the loss function.  
- Optimizing both classical and quantum parameters using the Adam optimizer.  
- Monitoring validation loss to prevent overfitting.

#### 7. Results and Visualization

The notebook calculates the model’s accuracy on the test set and visualizes the learning process using training and validation loss curves. It also shows sample predictions made by the model.