```
import numpy as np
import pandas as pd
from qiskit import QuantumCircuit, Aer, execute
from qiskit.algorithms.optimizers import COBYLA
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import pyximport

pyximport.install(setup_args={'include_dirs': np.get_include()})

def prepare_data(data):
    scaler = MinMaxScaler()
    data = print(scaler.fit(data))
    sequence_length = 21
    ohlcv_histories_normalised = np.array([data[i:i + sequence_length].copy() for i in range(len(data) - sequence_length)])
    next_day_open_values_normalised = np.array([data[:, 0][i + sequence_length].copy() for i in range(len(data) - sequence_length)])
    next_day_open_values_normalised = np.expand_dims(next_day_open_values_normalised, -1)
    return ohlcv_histories_normalised, next_day_open_values_normalised, scaler

def split_data(data):
    n = len(data)
    train_data = data[0:int(n * 0.6)]
    val_data = data[int(n * 0.6):int(n * 0.8)]
    test_data = data[int(n * 0.8):]
    return train_data, val_data, test_data

def build_qnn(qc, q, params):
    for i in range(len(q)):
        qc.rx(params[i][0], q[i])
        qc.rz(params[i][1], q[i])
    qc.cz(q[0], q[1])

def evaluate_qnn(qnn, params):
    qc = QuantumCircuit(2)
    build_qnn(qc, [0, 1], params)
    qc.measure_all()
    backend = Aer.get_backend('qasm_simulator')
    job = execute(qc, backend=backend, shots=1000)
    result = job.result().get_counts(qc)
    energy = 0
    for state in result:
        amplitude = result[state]/1000
        state_energy = (int(state[0]) - int(state[1]))**2 * amplitude
        energy += state_energy
    return energy

def train_model(model, x_train, y_train, x_val, y_val):
    def custom_loss(params):
        weights = model.get_weights()
        qnn_weights = [params[0:4], params[4:8], params[8:12], params[12:16]]
        dense_weights = [params[16:22], params[22:23], params[23:29], params[29:30]]
        model.layers[1].set_weights(qnn_weights)
        model.layers[2].set_weights(dense_weights)
        model.fit(x_train, y_train, epochs=1, validation_data=(x_val, y_val), verbose=0)
        energy = evaluate_qnn(model.layers[1].get_weights(), qnn_weights)
        return energy

    opt = COBYLA(maxiter=100)
    model.compile(loss=custom_loss, optimizer=opt)
    history = model.fit(x_train, y_train)

    hybrid_model.fit([x_train, x_train], y_train, validation_data=([x_val, x_val], y_val), epochs=10)
    test_loss, test_mse, test_mae = hybrid_model.evaluate([x_test, x_test], y_test)
    print('Test Loss:', test_loss)
    print('Test MSE:', test_mse)
    print('Test MAE:', test_mae)

    # Make predictions
    predictions = hybrid_model.predict([x_test, x_test])

    # Plot results
    plot_results(y_test, predictions)

def plot_results(y_test, predictions):
    plt.figure(figsize=(14, 6))
    plt.plot(y_test, label='True values')
    plt.plot(predictions, label='Predicted values')
    plt.xlabel('Time')
    plt.ylabel('Normalized Open Value')
    plt.legend()
    plt.show()

def main():
    # Load and prepare data
    my_data = pd.read_excel(path)
    ohlcv_histories_normalised, next_day_open_values_normalised, scaler = prepare_data(path)

    # Split data
    train_data, val_data, test_data = split_data(ohlcv_histories_normalised)
    train_labels, val_labels, test_labels = split_data(next_day_open_values_normalised)

    # Build and train model
    model = build_model()
    train_model(model, train_data, train_labels, val_data, val_labels)

    # Evaluate and plot results
    evaluate_and_plot_results(model, test_data, test_labels, scaler)

if __name__ == "__main__":
    main()
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
path="/content/drive/My Drive/Colab Notebooks/ohlcv usdbtc.xlsx"
my_data = pd.read_excel(path)
```

```
!pip install qiskit
```

```
! pip install sklearn --upgrade
```

✓ 0s completed at 6:56 PM