Start coding or generate with AI.

```
!pip install pennylane
```

```
Collecting pennylane
  Downloading PennyLane-0.40.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy<2.1 in /usr/local/lib/python3.11/dist-packages (from pennylane) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from pennylane) (1.14.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from pennylane) (3.4.2)
Collecting rustworkx>=0.14.0 (from pennylane)
  Downloading rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (10 kB)
Requirement already satisfied: autograd in /usr/local/lib/python3.11/dist-packages (from pennylane) (1.7.0)
Collecting tomlkit (from pennylane)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Collecting appdirs (from pennylane)
  Downloading appdirs-1.4.4-py2.py3-none-any.whl.metadata (9.0 kB)
Collecting autoray>=0.6.11 (from pennylane)
  Downloading autoray-0.7.1-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: cachetools in /usr/local/lib/python3.11/dist-packages (from pennylane) (5.5.2)
Collecting pennylane-lightning>=0.40 (from pennylane)
  Downloading PennyLane_Lightning-0.40.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (27 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from pennylane) (2.32.3)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from pennylane) (4.13.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from pennylane) (24.2)
Collecting diastatic-malt (from pennylane)
  Downloading diastatic_malt-2.15.2-py3-none-any.whl.metadata (2.6 kB)
Collecting scipy-openblas32>=0.3.26 (from pennylane-lightning>=0.40->pennylane)
  Downloading scipy_openblas32-0.3.29.0.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (56 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.1/56.1 kB 2.6 MB/s eta 0:00:00
Requirement already satisfied: astunparse in /usr/local/lib/python3.11/dist-packages (from diastatic-malt->pennylane) (1.6.3)
Requirement already satisfied: gast in /usr/local/lib/python3.11/dist-packages (from diastatic-malt->pennylane) (0.6.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.11/dist-packages (from diastatic-malt->pennylane) (2.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->pennylane)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->pennylane) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->pennylane) (2.3.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->pennylane) (2025.
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse->diastatic-malt-
Requirement already satisfied: six<2.0,>=1.6.1 in /usr/local/lib/python3.11/dist-packages (from astunparse->diastatic-malt->pe
Downloading PennyLane-0.40.0-py3-none-any.whl (2.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 20.2 MB/s eta 0:00:00
Downloading autoray-0.7.1-py3-none-any.whl (930 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 930.8/930.8 kB 16.3 MB/s eta 0:00:00
Downloading PennyLane_Lightning-0.40.0-cp311-cp311-manylinux_2_28_x86_64.whl (2.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.4/2.4 MB 20.3 MB/s eta 0:00:00
Downloading rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 40.3 MB/s eta 0:00:00
Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Downloading diastatic_malt-2.15.2-py3-none-any.whl (167 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 167.9/167.9 kB 9.8 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading scipy_openblas32-0.3.29.0.0-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.6/8.6 MB 25.0 MB/s eta 0:00:00
Installing collected packages: appdirs, tomlkit, scipy-openblas32, rustworkx, autoray, diastatic-malt, pennylane-lightning, pe
Successfully installed appdirs-1.4.4 autoray-0.7.1 diastatic-malt-2.15.2 pennylane-0.40.0 pennylane-lightning-0.40.0 rustworkx
```

```python
# Convert to Torch tensor (Ensure float64 for compatibility with Pennylane)
X_train_torch = torch.tensor(X_train, dtype=torch.float64)
y_train_torch = torch.tensor(y_train, dtype=torch.float64)
X_test_torch = torch.tensor(X_test, dtype=torch.float64)
y_test_torch = torch.tensor(y_test, dtype=torch.float64)
```

```python
import numpy as np
import pennylane as qml
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Generate Z2 x Z2 symmetric dataset
def generate_symmetric_data(n_samples=200):
    x1 = np.random.uniform(-1, 1, n_samples // 4)
    x2 = np.random.uniform(-1, 1, n_samples // 4)

    data = []
    labels = []

    for xi, yi in zip(x1, x2):
        base_point = np.array([xi, yi])
        transformed_points = [
            base_point,
```

```
            [-xi, yi],
            [xi, -yi],
            [-xi, -yi]
        ]
        class_label = int(xi * yi > 0)

        for point in transformed_points:
            data.append(point)
            labels.append(class_label)

    return np.array(data), np.array(labels)

# Prepare data
data, labels = generate_symmetric_data()
scaler = StandardScaler()
data = scaler.fit_transform(data)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

# Define quantum device
dev = qml.device("default.qubit", wires=2)

def standard_qnn(weights, x):
    qml.AngleEmbedding(x, wires=[0, 1])
    qml.BasicEntanglerLayers(weights, wires=[0, 1])
    return qml.expval(qml.PauliZ(0))

@qml.qnode(dev, interface='torch')
def equivariant_qnn(weights, x):
    qml.RY(x[0], wires=0)
    qml.RY(x[1], wires=1)
    qml.CNOT(wires=[0, 1])
    qml.RY(weights[0], wires=0)
    qml.RY(weights[1], wires=1)
    qml.CNOT(wires=[0, 1])
    return qml.expval(qml.PauliZ(0))

# Convert to Torch tensor (Ensure float64 for compatibility with Pennylane)
# Convert to Torch tensor (Ensure float32 for compatibility with PyTorch)
X_train_torch = torch.tensor(X_train, dtype=torch.float32)
y_train_torch = torch.tensor(y_train, dtype=torch.float32)
X_test_torch = torch.tensor(X_test, dtype=torch.float32)
y_test_torch = torch.tensor(y_test, dtype=torch.float32)

class QuantumModel(nn.Module):
    def __init__(self, quantum_circuit, num_layers=1, num_qubits=2):
        super().__init__()
        # For the equivariant model, the shape should be (num_qubits,)
        # as it has a single layer and expects a 1D tensor for weights.
        if quantum_circuit == equivariant_qnn:  # Check if it's the equivariant model
            self.q_weights = nn.Parameter(0.01 * torch.randn(num_qubits, dtype=torch.float32))  # Shape (num_qubits,)
        else:
            self.q_weights = nn.Parameter(0.01 * torch.randn(num_layers, num_qubits, dtype=torch.float32))  # Shape (num_layers, num
        self.q_circuit = quantum_circuit

    def forward(self, x):
        # Apply the quantum circuit to each data point in the batch
        results = [self.q_circuit(self.q_weights, data_point) for data_point in x]
        # Stack the results into a single tensor
        output = torch.stack(results)
        return torch.sigmoid(output).type(torch.float32)

# Train and evaluate both models
def train_model(qnn, X_train, y_train, X_test, y_test):
    optimizer = optim.Adam(qnn.parameters(), lr=0.1)
    loss_fn = nn.BCELoss()

    for epoch in range(100):
        optimizer.zero_grad()
        y_pred = qnn(X_train).squeeze()
        loss = loss_fn(y_pred, y_train)
        loss.backward()
        optimizer.step()

    with torch.no_grad():
        y_pred_test = qnn(X_test).squeeze().round()
    acc = accuracy_score(y_test, y_pred_test.numpy())
    return acc

# Train standard QNN
standard_model = QuantumModel(qml.qnode(dev)(standard_qnn), num_layers=1, num_qubits=2)
acc_standard = train_model(standard_model, X_train_torch, y_train_torch, X_test_torch, y_test_torch)
# Train equivariant QNN
```

```
equivariant_model = QuantumModel(equivariant_qnn)
acc_equivariant = train_model(equivariant_model, X_train_torch, y_train_torch, X_test_torch, y_test_torch)

print(f"Accuracy of Standard QNN: {acc_standard:.2f}")
print(f"Accuracy of Equivariant QNN: {acc_equivariant:.2f}")
```

```
Accuracy of Standard QNN: 0.55
Accuracy of Equivariant QNN: 0.45
```

Start coding or generate with AI.