Code 1

|  |
| --- |
| import pennylane as qml  dev = qml.device("default.qubit", wires =1) |

Code 2a

|  |
| --- |
| def qfun():  qml.PauliX(wires=0)  qml.Hadamard(wires=0)  return qml.state() |

Code 2b

|  |
| --- |
| dev = qml.device(name="default.qubit", wires=1) |

Code 2c

|  |
| --- |
| qn = qml.QNode(qfun,dev) |

Code 2d

|  |
| --- |
| print(qml.draw(qn)())  fig, ax = qml.draw\_mpl(qn)() |

Code 3

|  |
| --- |
| def circuit():  qml.PauliX(0)  qml.Hadamard(0)  dev = qml.device(name="default.qubit", wires =1)  @qml.qnode(dev)  def qn\_sv():  circuit()  return qml.state()  @qml.qnode(dev)  def qn\_probs():  circuit()  return qml.probs()  print(qn\_sv())  print(qn\_probs()) |

Code 4

|  |
| --- |
| # Data processing: import numpy as np import pandas as pd  # Utils: import sklearn.decomposition from sklearn.preprocessing import StandardScaler, MinMaxScaler  # Visuals: import seaborn as sns import matplotlib.pyplot as plt |

Code 4a

|  |
| --- |
| !pip uninstall keras  !pip install keras==2.10.0  !pip install --upgrade pip  !pip install --upgrade tensorflow  !pip install --upgrade keras  !pip install pennylane  import pennylane as qml  import tensorflow as tf  import numpy as np  import keras |

Code 5

|  |
| --- |
| # Set random seed: GLOBAL\_SEED = 419514 np.random.seed(GLOBAL\_SEED)  # Computation data format: comp\_dtype = 'float32'  # Increase pandas printing limits pd.set\_option('display.max\_columns', 90) pd.set\_option('display.width', 1000) pd.set\_option('display.max\_rows', 90) |

Code 6

|  |
| --- |
| data\_read\_path = os.path.join(".", "data", "raw", "CSV-01-12", "01-12", "DrDoS\_SSDP.csv") data\_df = pd.read\_csv(data\_read\_path) data\_df.drop(labels=['Unnamed: 0'], axis=1, inplace=True) data\_df.shape |

Code 7

|  |
| --- |
| def first\_unique\_values(column):  unique\_values = column.value\_counts().index.values    if column.isna().sum()>0:  unique\_values = np.append(np.nan, unique\_values)  sorted\_unique\_values = unique\_values[:min(5, unique\_values.shape[0])]    return sorted\_unique\_values  def gen\_basic\_info\_dataframe(df, typical\_values = True):  '''  Making DataFrame with:  -number of unique values  -data type  -percentage of NaN values   -list with max. 5 most common values in the column.   If NaN occur in the column it's first element in this list,   even if it's not the most common value.  '''  basic\_info=pd.DataFrame(df.nunique(), columns=['num\_unique\_values'])   basic\_info['data\_type']= df.dtypes  basic\_info['NaN\_percentage'] = (df.isna().sum())\*100/df.shape[0]  if typical\_values:  basic\_info['typical\_values'] = df.apply(first\_unique\_values, axis=0)    return basic\_infobasic\_info = gen\_basic\_info\_dataframe(data\_df) basic\_info[20:25] |

Code 8

|  |
| --- |
| data\_df.drop(labels=['Flow Bytes/s'], axis=1, inplace=True) |

Code 9

|  |
| --- |
| irrelevant\_columns = [  'Flow ID',  ' Source IP',  ' Source Port',  ' Destination IP',  ' Destination Port',  ' Timestamp',  'SimillarHTTP',  ]  data\_df.drop(labels=irrelevant\_columns, axis=1, inplace=True) |

Code 10

|  |
| --- |
| # get these columns names as a list: one\_value\_columns = basic\_info[basic\_info['num\_unique\_values']==1].index.tolist() print("Columns to drop:\n", one\_value\_columns)  data\_df.drop(labels=one\_value\_columns, axis=1, inplace=True) |

Code 11

|  |
| --- |
| corr\_matrix = data\_df.corr(method='pearson') |

Code 12

|  |
| --- |
| fig, ax = plt.subplots(figsize=(25, 25)) sns.heatmap(corr\_matrix,  square=True,  annot=True,  ax=ax) plt.show() |

Code 13

|  |
| --- |
| threshold = 0.94  filtered\_corr\_matrix = corr\_matrix[np.abs(corr\_matrix)>threshold] unstack\_f\_cm = filtered\_corr\_matrix.unstack().dropna().reset\_index()  # remove self-correlations: corr\_pairs = unstack\_f\_cm[unstack\_f\_cm['level\_0']!=unstack\_f\_cm['level\_1']].reset\_index().drop(labels=['index'], axis=1)  print(f'{len(corr\_pairs)} correlations between columns with coeff. over {threshold}.') # set columns to drop: corr\_columns\_to\_drop = [] for id, row in corr\_pairs.iterrows():  if not (row['level\_0'] in corr\_columns\_to\_drop) and not (row['level\_1'] in corr\_columns\_to\_drop):   corr\_columns\_to\_drop.append(row['level\_0'])  print(f'{len(corr\_columns\_to\_drop)} columns to drop.') data\_df.drop(labels=corr\_columns\_to\_drop, axis=1, |

Code 14

|  |
| --- |
| n\_samples = 10000 |

Code 15

|  |
| --- |
| # choose indices from both classes: benign\_ids = data\_df[data\_df[' Label']=='BENIGN'].index ddos\_ids = data\_df[data\_df[' Label']=='DrDoS\_SSDP'].index  ddos\_samples\_ids = np.random.choice(ddos\_ids, (n\_samples-763)) benign\_samples\_ids = np.array(benign\_ids) # merge chosen indices: samples\_ids = np.concatenate((ddos\_samples\_ids, benign\_samples\_ids))  selected\_data\_df = data\_df.iloc[samples\_ids] selected\_data\_df[' Label'].value\_counts()DrDoS\_SSDP 9237 BENIGN 763 Name: Label, dtype: int64 |

Code 16

|  |
| --- |
| y = np.array(selected\_data\_df[' Label'].astype('category').cat.codes.astype(int)) |

Code 17

|  |
| --- |
| n\_features = 2  # Standardize all the features: x = StandardScaler().fit\_transform(np.array(selected\_data\_df.drop(columns=[' Label'], inplace=False)))  # PCA fitting and transforming the data: pca = sklearn.decomposition.PCA(n\_components=n\_features) pca.fit(x) x = pca.transform(x)  # Normalize the output to the range (-1, +1): minmax\_scale = MinMaxScaler((-1, 1)).fit(x) x = minmax\_scale.transform(x) x = x.astype(comp\_dtype)  # Plot results: for k in range(0, 2):  x\_axis\_data = x[np.array(y) == k, 0]  y\_axis\_data = x[np.array(y) == k, 1]  label = 'Benign' if k == 0 else 'DDoS'  print(f'{label}: {x\_axis\_data.shape}')  plt.scatter(x\_axis\_data, y\_axis\_data, label=label)  plt.title("DDoS\_SSDP Dataset (Dimensionality Reduced With PCA)") plt.legend() plt.show()Benign: (763,) DDoS: (9237,) |

Code 18

|  |
| --- |
| # QML import pennylane as qml  # ML import tensorflow as tf from tensorflow.keras.callbacks import ReduceLROnPlateau  # Data processing: import numpy as np  # Utils from sklearn.model\_selection import train\_test\_split from sklearn.metrics import classification\_report, recall\_score, confusion\_matrix, ConfusionMatrixDisplay from sklearn.metrics import f1\_score, accuracy\_score, precision\_score, make\_scorer  # Visuals: import matplotlib.pyplot as plt import matplotlib.ticker as mtick |

Code 19

|  |
| --- |
| # Set random seed: GLOBAL\_SEED = 419514 np.random.seed(GLOBAL\_SEED) tf.random.set\_seed(GLOBAL\_SEED) # Computation data format: comp\_dtype = 'float32' |

Code 20

|  |
| --- |
| # set ratio: train\_val\_test\_split = [0.7, 0.15, 0.15] test\_val\_size = sum(train\_val\_test\_split[1:]) val\_size = train\_val\_test\_split[2]/test\_val\_size  # split dataset: trainX, testX, trainy, testy = train\_test\_split(x, y, stratify=y, test\_size=test\_val\_size, random\_state=GLOBAL\_SEED) testX, valx, testy, valy = train\_test\_split(testX, testy, stratify=testy, test\_size=val\_size, random\_state=GLOBAL\_SEED)  # One-hot encode: trainy = tf.one\_hot(trainy, depth=n\_features, dtype=comp\_dtype) testy = tf.one\_hot(testy, depth=n\_features, dtype=comp\_dtype)  valy = tf.one\_hot(valy, depth=n\_features, dtype=comp\_dtype) |

Code 21

|  |
| --- |
| n\_qubits = 2 layers = 1 data\_dimension = n\_features  batch\_size = 5 n\_epochs = 1 |

Code 22

|  |
| --- |
| dev = qml.device("default.qubit", wires=n\_qubits)  @qml.qnode(dev) def qnode(inputs, weights):  qml.templates.AngleEmbedding(np.pi\*inputs, wires=range(n\_qubits))  qml.templates.StronglyEntanglingLayers(weights, wires=range(n\_qubits))    return [qml.expval(qml.PauliZ(i)) for i in range(n\_qubits)]   weight\_shapes = {"weights": (layers,n\_qubits,3)}  inputs = np.random.rand(n\_qubits).astype(comp\_dtype) weights = np.random.rand(layers, n\_qubits, 3).astype(comp\_dtype)  plt.figure(figsize=(10,10)) qml.draw\_mpl(qnode)(inputs, weights) plt.show() |

Code 23

|  |
| --- |
| q\_layer = qml.qnn.KerasLayer(qnode, weight\_shapes, output\_dim=n\_qubits, dtype=comp\_dtype) q\_layer.build(2)  q\_model = tf.keras.models.Sequential() q\_model.add(tf.keras.layers.Dense(n\_qubits, activation='sigmoid', input\_dim=data\_dimension)) q\_model.add(q\_layer) q\_model.add(tf.keras.layers.Dense(data\_dimension, activation='softmax'))  q\_model.summary() |

Code 24

|  |
| --- |
| %%time  opt = tf.keras.optimizers.Adam(learning\_rate=0.05) q\_model.compile(loss='categorical\_crossentropy', optimizer=opt,metrics=["accuracy"])  q\_history = q\_model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=n\_epochs, batch\_size=batch\_size) |

Code 25

|  |
| --- |
| # predicition: valpredy = q\_model.predict(valx) valpredy\_round = np.round(valpredy)  # metrics calculation: q\_classification = classification\_report(valy[:,1], valpredy\_round[:,1]) q\_confusion = confusion\_matrix(valy[:,1], valpredy\_round[:,1])   q\_accuracy = round(accuracy\_score(valy[:,1], valpredy\_round[:,1])\*100,5) q\_recall = round(recall\_score(valy[:,1], valpredy\_round[:,1], average='macro')\*100,5) q\_precision = round(precision\_score(valy[:,1], valpredy\_round[:,1], average='weighted')\*100,5) q\_f1 = round(f1\_score(valy[:,1], valpredy\_round[:,1], average='weighted')\*100,5)  print(f'Accuracy:\t {q\_accuracy:.2f}%') print(f'Recall:\t\t {q\_recall:.2f}%') print(f'Precision:\t {q\_precision:.2f}%') print(f'F1:\t\t {q\_f1:.2f}%') |

|  |
| --- |
| disp = ConfusionMatrixDisplay(confusion\_matrix=q\_confusion) disp.plot() plt.show() |

Code 26

|  |
| --- |
| model = tf.keras.models.Sequential() model.add(tf.keras.layers.Dense(n\_qubits, activation='relu', input\_dim=data\_dimension)) model.add(tf.keras.layers.Dense(data\_dimension, activation='relu')) model.add(tf.keras.layers.Dense(data\_dimension, activation='softmax'))  model.summary() |

Code 27

|  |
| --- |
| %%time  opt = tf.keras.optimizers.Adam(learning\_rate=0.02) model.compile(loss='categorical\_crossentropy', optimizer=opt,metrics=["accuracy"])  history = model.fit(trainX, trainy, validation\_data=(testX, testy), epochs=n\_epochs, batch\_size=batch\_size) |

|  |
| --- |
| # prediction: valpredy = model.predict(valx) valpredy\_round = np.round(valpredy)  # metrics calculation: classification = classification\_report(valy[:,1], valpredy\_round[:,1]) confusion = confusion\_matrix(valy[:,1], valpredy\_round[:,1])   accuracy = round(accuracy\_score(valy[:,1], valpredy\_round[:,1])\*100,5) recall = round(recall\_score(valy[:,1], valpredy\_round[:,1], average='macro')\*100,5) precision = round(precision\_score(valy[:,1], valpredy\_round[:,1], average='weighted')\*100,5) f1 = round(f1\_score(valy[:,1], valpredy\_round[:,1], average='weighted')\*100,5)  print(f'Accuracy:\t {accuracy:.2f}%') print(f'Recall:\t\t {recall:.2f}%') print(f'Precision:\t {precision:.2f}%') print(f'F1:\t\t {f1:.2f}%') |

|  |
| --- |
| # plotting confusion matrix: disp = ConfusionMatrixDisplay(confusion\_matrix=confusion) disp.plot() plt.show() |

Code 28

|  |
| --- |
| c\_results = np.asarray([accuracy, recall, precision, f1]) q\_results = np.asarray([q\_accuracy, q\_recall, q\_precision, q\_f1]) results\_description = ['Accuracy', 'Recall', 'Precision', 'F1']  # plot preparation: fig, ax = plt.subplots(figsize = (10,6)) idx = np.asarray([i for i in range(4)]) width = 0.4  # plotting: q\_bars = ax.bar(idx-width/2, q\_results, width=width, label='H-QNN') c\_bars = ax.bar(idx+width/2, c\_results, width=width, label='DNN')  # setting ticks: ax.set\_xticks(idx) ax.set\_title('Models results comparison:') ax.set\_xticklabels(results\_description, rotation=65) fmt = '%.2f%%' yticks = mtick.FormatStrFormatter(fmt) ax.yaxis.set\_major\_formatter(yticks)  # add bars labels: ax.bar\_label(c\_bars, fmt=fmt) ax.bar\_label(q\_bars, fmt=fmt)  # set y-axis limits ax.set\_ylim(90, 100.5)  ax.legend(loc=4) plt.grid()  plt.show() |