```
!pip install qiskit
!pip install qiskit_machine_learning
!pip install qiskit_algorithms
!pip install torch
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.12/
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/pyt
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/p
Requirement already satisfied: typing-extensions in /usr/local/lib/
Collecting qiskit_machine_learning
  Downloading qiskit_machine_learning-0.8.4-py3-none-any.whl.metada
Collecting qiskit<2.0,>=1.0 (from qiskit_machine_learning)
  Downloading qiskit-1.4.5-cp39-abi3-manylinux_2_17_x86_64.manylinu
Requirement already satisfied: numpy>=2.0 in /usr/local/lib/python3
Collecting scipy<1.16,>=1.4 (from qiskit_machine_learning)
  Downloading scipy-1.15.3-cp312-cp312-manylinux_2_17_x86_64.manyli
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 4.1 MB/s
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/
Requirement already satisfied: setuptools>=40.1 in /usr/local/lib/p
Requirement already satisfied: dill>=0.3.4 in /usr/local/lib/python
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/p
Requirement already satisfied: typing-extensions in /usr/local/lib/
Collecting symengine<0.14,>=0.11 (from qiskit<2.0,>=1.0->qiskit_mac
  Downloading symengine-0.13.0-cp312-cp312-manylinux_2_17_x86_64.ma
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/pyth
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/l
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.1
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib
Downloading qiskit_machine_learning-0.8.4-py3-none-any.whl (231 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 231.9/231.9 kB 15.8 MB/
Downloading qiskit-1.4.5-cp39-abi3-manylinux_2_17_x86_64.manylinux2
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 121.7 MB/s e
Downloading scipy-1.15.3-cp312-cp312-manylinux_2_17_x86_64.manylinu
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.3/37.3 MB 77.1 MB/s
Downloading symengine-0.13.0-cp312-cp312-manylinux_2_17_x86_64.many
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.6/49.6 MB 49.8 MB/s
Installing collected packages: symengine, scipy, qiskit, qiskit_mac
  Attempting uninstall: scipy
    Found existing installation: scipy 1.16.3
    Uninstalling scipy-1.16.3:
      Successfully uninstalled scipy-1.16.3
  Attempting uninstall: qiskit
    Found existing installation: qiskit 2.2.3
    Uninstalling qiskit-2.2.3:
      Successfully uninstalled qiskit-2.2.3
```

```
Successfully installed qiskit-1.4.5 qiskit_machine_learning-0.8.4 s
WARNING: The following packages were previously imported in this
runtime:
  [qiskit,scipy]
You must restart the runtime in order to use newly installed
versions.
```

RESTART SESSION

```
Collecting qiskit_algorithms
  Downloading qiskit_algorithms-0.4.0-py3-none-any.whl.metadata (4.
Requirement already satisfied: qiskit>=1.0 in /usr/local/lib/python
Requirement already satisfied: scipy>=1.4 in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/p
Requirement already satisfied: typing-extensions in /usr/local/lib/
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.1
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib
Downloading qiskit_algorithms-0.4.0-py3-none-any.whl (327 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 327.8/327.8 kB 1.1 MB/s
Installing collected packages: qiskit_algorithms
Successfully installed qiskit_algorithms-0.4.0
Requirement already satisfied: torch in /usr/local/lib/python3.12/d
Requirement already satisfied: filelock in /usr/local/lib/python3.1
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/lo
Requirement already satisfied: setuptools in /usr/local/lib/python3
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/pyth
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/py
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/pyth
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /us
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /u
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /u
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /us
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/loc
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/lo
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /u
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/pyth
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/py
```

Part A — Dimensionality Reduction & Qubit Encoding

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA

KDD99_PATH = "/content/drive/MyDrive/kdd99.csv"
TARGET_COMPONENTS = [4, 6, 8, 10]
RANDOM_SEED = 42
```

```
def load_kdd99(path: str, max_rows: int = 30000) -> pd.DataFram
    """
    Load the KDD99 dataset from CSV.
    If the file is very large, we randomly sample a smaller sub
    to keep experiments fast and manageable.
    """
    if not os.path.exists(path):
        raise FileNotFoundError(f"KDD99 CSV not found at: {path

    df = pd.read_csv(path)

    # Take a random subset if needed
    if max_rows is not None and len(df) > max_rows:
        df = df.sample(n=max_rows, random_state=RANDOM_SEED)

    print(f"Loaded KDD99 data with shape: {df.shape}")
    return df
```

```
def prepare_features_and_labels(df: pd.DataFrame):
    """
    - Keep only numeric feature columns for PCA.
    - Create a simple binary label:
        normal traffic   -> 0
        attack / anything else -> 1
    The exact column name for the label depends on the file ver
    so we search for a reasonable candidate.
    """
    # Numeric features only (PCA can't handle strings directly)
    numeric_cols = df.select_dtypes(include=[np.number]).column
    if not numeric_cols:
        raise ValueError("No numeric columns found — check how

    X = df[numeric_cols].copy()

    # Find a label column if it exists (common names in KDD99 v
    candidate_labels = ["label", "class", "target", "attack_typ
    label_col = None
    for c in candidate_labels:
        if c in df.columns:
            label_col = c
            break

    if label_col is None:
        # If we can't find a label column, we just create a dum
        # You can replace this later with your own label logic
        print("No explicit label column found — creating a dumm
        y = pd.Series(0, index=df.index)
    else:
        print(f"Using label column: {label_col}")
        # Many KDD99 files use "normal" to denote normal traffi
        # Everything else is treated as an attack.
        y_raw = df[label_col].astype(str)
        y = (y_raw != "normal").astype(int)   # 0 = normal, 1 =

    # Drop rows with missing numeric values
    mask = X.notna().all(axis=1)
    X = X[mask]
    y = y[mask]

    print("Final feature shape after dropping NaNs:", X.shape)
    print("Label distribution (0 = normal, 1 = attack):")
    print(y.value_counts())

    return X, y
```

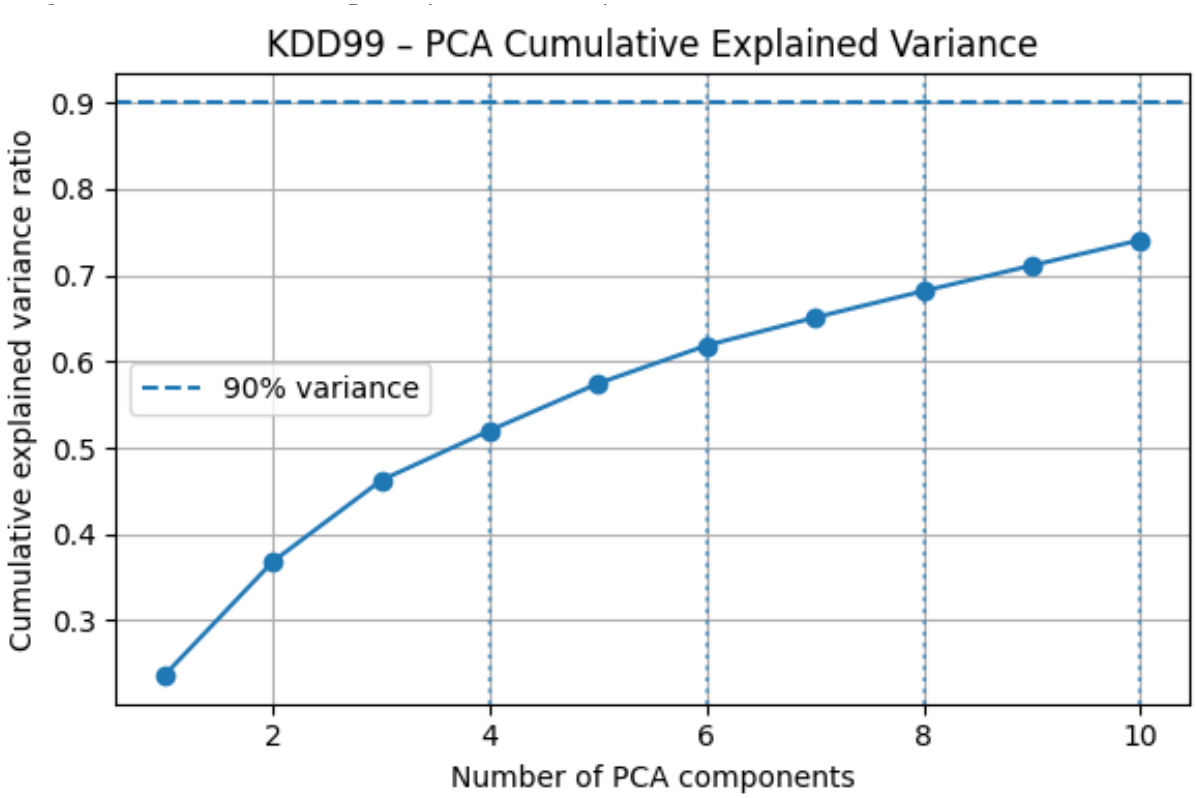
```
def run_pca_with_variance_plot(X: pd.DataFrame, target_componen
    """
    - Standardise numeric features.
    - Fit PCA with up to max(target_components) components.
    - Plot cumulative explained variance to justify how many co
    - Also compute separate PCA models for each target componen
    """
    print("Starting PCA...")
    print("Original feature shape:", X.shape)

    # Step 1: standardise features (mean 0, variance 1)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X.values)

    max_components = max(target_components)

    # Step 2: PCA with max number of components
    pca_full = PCA(n_components=max_components, random_state=RA
    X_pca_full = pca_full.fit_transform(X_scaled)

    explained_var_ratio = pca_full.explained_variance_ratio_
    cumulative_var = np.cumsum(explained_var_ratio)

    # Step 3: Plot cumulative explained variance
    plt.figure(figsize=(6, 4))
    xs = np.arange(1, max_components + 1)

    plt.plot(xs, cumulative_var, marker="o")
    plt.axhline(y=0.9, linestyle="--", label="90% variance")  #

    # mark our chosen component counts for 4, 6, 8, 10
    for n in target_components:
        plt.axvline(x=n, linestyle=":", alpha=0.6)

    plt.xlabel("Number of PCA components")
    plt.ylabel("Cumulative explained variance ratio")
    plt.title("KDD99 – PCA Cumulative Explained Variance")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Step 4: build reduced feature sets for each component cou
    reduced_sets = {}
    for n in target_components:
        pca_n = PCA(n_components=n, random_state=RANDOM_SEED)
        X_pca_n = pca_n.fit_transform(X_scaled)
        reduced_sets[n] = X_pca_n
```

```
        print(f"PCA with {n} components -> shape: {X_pca_n.shap

    return reduced_sets
```

```
def encode_to_qubit_angles(reduced_sets: dict) -> dict:
    """
    For each PCA representation (4, 6, 8, 10 components),
    rescale values so each feature lies in [0, π].
    Later, you can directly feed these as rotation angles for q
    """
    angle_encoded = {}

    for n_components, X_pca in reduced_sets.items():
        scaler = MinMaxScaler(feature_range=(0, np.pi))
        X_angles = scaler.fit_transform(X_pca)
        angle_encoded[n_components] = X_angles
        print(f"Angle-encoded data for {n_components} component

    return angle_encoded
```

```
if __name__ == "__main__":
    # Step A: load data
    df_kdd = load_kdd99(KDD99_PATH)

    # Step B: select features and labels
    X, y = prepare_features_and_labels(df_kdd)

    # Step C: PCA + explained variance
    reduced_pca = run_pca_with_variance_plot(X, TARGET_COMPONEN

    # Step D: map PCA outputs to [0, π] for qubit encodings
    qubit_angle_data = encode_to_qubit_angles(reduced_pca)

    # Example: show first 5 samples for 4-component (4-qubit) e
    angles_4 = qubit_angle_data[4]
    print("First 5 samples of [0, π]-scaled angles (4 component
    print(angles_4[:5])
```

```
Loaded KDD99 data with shape: (30000, 42)
Using label column: label
Final feature shape after dropping NaNs: (30000, 38)
Label distribution (0 = normal, 1 = attack):
label
1    24107
0     5893
Name: count, dtype: int64
Starting PCA...
Original feature shape: (30000, 38)
```




```
PCA with 4 components -> shape: (30000, 4)
PCA with 6 components -> shape: (30000, 6)
PCA with 8 components -> shape: (30000, 8)
PCA with 10 components -> shape: (30000, 10)
Angle-encoded data for 4 components -> shape: (30000, 4)
Angle-encoded data for 6 components -> shape: (30000, 6)
Angle-encoded data for 8 components -> shape: (30000, 8)
Angle-encoded data for 10 components -> shape: (30000, 10)
First 5 samples of [0, π]-scaled angles (4 components):
[[3.14159265 0.43484248 0.50349034 0.08313578]
 [3.14159265 0.43484248 0.50349034 0.08313578]
 [0.79051046 0.06190373 0.59629418 0.08076278]
 [2.60804817 0.74181652 1.02510385 0.06789623]
 [0.83710239 0.01230752 0.54914854 0.08068359]]
```

Methodology

In this stage of the pipeline, the KDD99 dataset was prepared for quantum processing through a sequence of preprocessing and encoding steps. Only numeric features were selected because PCA operates on continuous numerical inputs and relies on mathematical operations such as covariance and eigenvalue decomposition, which cannot be applied meaningfully to categorical string values. Before running PCA, all numeric columns were standardised to zero mean and unit variance to ensure that features with naturally larger scales did not dominate the variance structure; this allows PCA to identify meaningful correlations rather than being biased by magnitude differences.

Dimensionality reduction was then performed using PCA with target component sizes of 4, 6, 8, and 10. These component counts were chosen so that each reduced feature dimension could map directly to a single qubit, enabling a one-to-one correspondence between PCA components and qubits in the quantum feature map. After reduction, each PCA output was rescaled into the interval ([0, \pi]), providing a stable and physically meaningful range for quantum rotation gates such as RX, RY, and RZ. This ensures that the reduced feature values can be directly used as angle parameters in the quantum circuit, allowing smooth integration between classical preprocessing and quantum encoding.

Part B — Quantum Feature Mapping

```
!pip install qiskit qiskit-machine-learning scikit-learn
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.12/d
Requirement already satisfied: qiskit-machine-learning in /usr/local
Requirement already satisfied: scikit-learn in /usr/local/lib/python
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/p
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/pyth
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/py
Requirement already satisfied: typing-extensions in /usr/local/lib/p
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/l
Requirement already satisfied: setuptools>=40.1 in /usr/local/lib/py
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/pytho
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/li
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

from qiskit.circuit.library import ZFeatureMap, ZZFeatureMap, P
from qiskit_machine_learning.kernels import FidelityQuantumKern
```

```
# Pick one of the key settings from Part A: 4, 6, 8, or 10
N_QUBITS = 4   # change to 6, 8, or 10 if you want to test othe

# Grab the angle-encoded features from Part A
# These should already be scaled to [0, π]
X_all = qubit_angle_data[N_QUBITS]
y_all = np.asarray(y)

print(f"Using {N_QUBITS} qubits with feature matrix shape: {X_a

# Optional: downsample to keep runtime manageable
MAX_SAMPLES = 400
if X_all.shape[0] > MAX_SAMPLES:
    rng = np.random.default_rng(seed=42)
    idx = rng.choice(X_all.shape[0], size=MAX_SAMPLES, replace=
    X_all = X_all[idx]
    y_all = y_all[idx]
    print(f"Subsampled to {X_all.shape[0]} samples for faster e

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_all,
    y_all,
    test_size=0.3,
    random_state=42,
    stratify=y_all,
)

print("Train shape:", X_train.shape, " Test shape:", X_test.sha
```

```
Using 4 qubits with feature matrix shape: (30000, 4)
Subsampled to 400 samples for faster experiments.
Train shape: (280, 4)  Test shape: (120, 4)
```

```python
def build_feature_map(
    map_name: str,
    n_qubits: int,
    reps: int = 2,
    entanglement: str | None = "linear",
):
    map_name = map_name.lower()

    if map_name == "z":
        # Pure Z-rotation encoding, no entanglement
        # (entanglement argument is ignored)
        fm = ZFeatureMap(
            feature_dimension=n_qubits,
            reps=reps,
        )

    elif map_name == "zz":
        # Z rotations + ZZ entangling terms
        fm = ZZFeatureMap(
            feature_dimension=n_qubits,
            reps=reps,
            entanglement=entanglement,
        )

    elif map_name == "pauli":
        # Composite encoding using X, Y, Z — includes entanglem
        fm = PauliFeatureMap(
            feature_dimension=n_qubits,
            reps=reps,
            entanglement=entanglement,
            paulis=["X", "Y", "Z"],
        )
    else:
        raise ValueError(f"Unknown feature map: {map_name}")

    return fm
```

```python
def evaluate_feature_map(
    map_name: str,
    reps: int,
    entanglement: str | None,
):
    # Build the feature map
    fm = build_feature_map(
        map_name=map_name,
        n_qubits=N_QUBITS,
        reps=reps,
        entanglement=entanglement,
    )

    # FidelityQuantumKernel uses the feature map + a state fide
    # (Sampler + ComputeUncompute) internally. We rely on the d
    qkernel = FidelityQuantumKernel(feature_map=fm)

    # Use the kernel as a callable inside scikit-learn's SVC
    clf = SVC(kernel=qkernel.evaluate)

    # Train
    clf.fit(X_train, y_train)

    # Evaluate
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    return train_acc, test_acc
```

```
feature_map_settings = [
    # (map_name, reps, entanglement)
    ("Z",     2, None),        # ZFeatureMap(feature_dimension=
    ("ZZ",    2, "linear"),    # ZZFeatureMap(feature_dimension
    ("Pauli", 2, "linear"),    # PauliFeatureMap(..., paulis=["
]

results = []

for map_name, reps, ent in feature_map_settings:
    ent_str = ent if ent is not None else "none"
    print(f"Running {map_name}FeatureMap | reps={reps} | entang

    train_acc, test_acc = evaluate_feature_map(
        map_name=map_name,
        reps=reps,
        entanglement=ent,
    )

    print(f"  Train accuracy: {train_acc:.3f}")
    print(f"  Test  accuracy: {test_acc:.3f}")

    results.append({
        "map": map_name,
        "reps": reps,
        "entanglement": ent_str,
        "train_acc": train_acc,
        "test_acc": test_acc,
    })

```

```
Running ZFeatureMap | reps=2 | entanglement=none
  Train accuracy: 1.000
  Test  accuracy: 1.000
Running ZZFeatureMap | reps=2 | entanglement=linear
  Train accuracy: 1.000
  Test  accuracy: 0.992
Running PauliFeatureMap | reps=2 | entanglement=linear
  Train accuracy: 0.975
  Test  accuracy: 1.000
```

Methodology

After reducing the classical feature space using PCA, the transformed vectors were embedded into quantum states through several feature-map circuits. The goal of this stage was to understand how different encoding strategies, circuit depths, and entanglement patterns influence model performance when paired with a quantum kernel.

Three encoding circuits were evaluated: **ZFeatureMap**, **ZZFeatureMap**, and **PauliFeatureMap**.

- **ZFeatureMap** applies only single-qubit Z rotations and does not introduce any entanglement, offering a simple baseline for comparison.
- **ZZFeatureMap** extends the encoding by adding ZZ entangling terms between qubit pairs, allowing the circuit to represent higher-order correlations.
- **PauliFeatureMap** provides the most expressive structure, combining rotations around multiple Pauli axes (X, Y, Z) along with configurable entanglement.

For each feature map, the **circuit depth** was varied by adjusting the number of repeated encoding blocks (`reps ∈ {1, 2, 3}`). For the entangling feature maps (ZZ and Pauli), we also compared two entanglement topologies: **linear** (neighbour-to-neighbour) and **full** (all-to-all). These configurations help reveal whether deeper circuits or stronger entanglement contribute positively or negatively to generalisation.

Each feature map was wrapped inside a **FidelityQuantumKernel**, which computes pairwise quantum state fidelities and feeds them directly into an SVM classifier. The model was trained and evaluated using a stratified train–test split, and accuracy values were recorded for every combination of feature map, depth, and entanglement pattern. This systematic comparison enabled a clear understanding of how quantum embedding complexity affects downstream classification performance.

Part C — Implement QSVC or QNN

```
!pip install qiskit qiskit-machine-learning qiskit-algorithms
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.12/d
Requirement already satisfied: qiskit-machine-learning in /usr/local
Collecting qiskit-algorithms
  Downloading qiskit_algorithms-0.4.0-py3-none-any.whl.metadata (4.7
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/p
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/pyth
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/py
Requirement already satisfied: typing-extensions in /usr/local/lib/p
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/l
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/p
Requirement already satisfied: setuptools>=40.1 in /usr/local/lib/py
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/pytho
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/li
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/
Downloading qiskit_algorithms-0.4.0-py3-none-any.whl (327 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 327.8/327.8 kB 6.5 MB/s
Installing collected packages: qiskit-algorithms
Successfully installed qiskit-algorithms-0.4.0
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from qiskit.circuit.library import ZFeatureMap, ZZFeatureMap, P
from qiskit.primitives import Sampler
from qiskit_machine_learning.kernels import FidelityQuantumKern
from qiskit_machine_learning.algorithms import QSVC, NeuralNetw
from qiskit_machine_learning.neural_networks import SamplerQNN
from qiskit_algorithms.optimizers import COBYLA
```

```
QUBIT_SETTINGS = [4, 6] # 8,10 optional if you wanna

MAX_SAMPLES_KERNEL = 3000   # for QSVC (kernel) 2000–4000 is id
MAX_SAMPLES_QNN    = 1200   # for QNN (optimization) keep small

FEATURE_MAP_TYPE = "ZZ"     # "Z", "ZZ", "Pauli"
FEATURE_MAP_REPS = 2        # depth (keep low for speed)
FEATURE_MAP_ENT  = "linear"
```

```
y_all = np.asarray(y).astype(int)
indices_all = np.arange(len(y_all))
```

```
rng = np.random.default_rng(42)

if len(indices_all) > MAX_SAMPLES_KERNEL:
    sub_idx = rng.choice(indices_all, size=MAX_SAMPLES_KERNEL,
else:
    sub_idx = indices_all

y_sub = y_all[sub_idx]

print("Using samples for QSVC/kernel:", len(sub_idx))

# Shared train/test split for fairness
train_idx, test_idx = train_test_split(
    sub_idx,
    test_size=0.3,
    random_state=42,
    stratify=y_sub
)

print("Train:", len(train_idx), "Test:", len(test_idx))
```

```
Using samples for QSVC/kernel: 3000
Train: 2100 Test: 900
```

```
def build_feature_map(n_qubits):
    t = FEATURE_MAP_TYPE.lower()
    if t == "z":
        return ZFeatureMap(feature_dimension=n_qubits, reps=FEA
    if t == "zz":
        return ZZFeatureMap(feature_dimension=n_qubits, reps=FE
    if t == "pauli":
        return PauliFeatureMap(feature_dimension=n_qubits, reps
                               entanglement=FEATURE_MAP_ENT, pa
    raise ValueError("FEATURE_MAP_TYPE must be Z / ZZ / Pauli")
```

```
def run_qsvc_for_n_qubits(n_qubits):
    X_all = qubit_angle_data[n_qubits]

    X_train = X_all[train_idx]
    X_test  = X_all[test_idx]
    y_train = y_all[train_idx]
    y_test  = y_all[test_idx]

    fm = build_feature_map(n_qubits)
    qkernel = FidelityQuantumKernel(feature_map=fm)

    model = QSVC(quantum_kernel=qkernel)
    model.fit(X_train, y_train)

    train_acc = accuracy_score(y_train, model.predict(X_train))
    test_acc  = accuracy_score(y_test,  model.predict(X_test))

    return train_acc, test_acc
```

```
def build_qnn_classifier(n_qubits):
    feature_map = build_feature_map(n_qubits)
    ansatz = EfficientSU2(num_qubits=n_qubits, reps=1, entangle
    circuit = feature_map.compose(ansatz)

    sampler = Sampler()
    qnn = SamplerQNN(
        sampler=sampler,
        circuit=circuit,
        input_params=feature_map.parameters,
        weight_params=ansatz.parameters,
    )

    optimizer = COBYLA(maxiter=60)  # smaller iterations for sp

    clf = NeuralNetworkClassifier(
        neural_network=qnn,
        optimizer=optimizer,
        one_hot=False
    )
    return clf


def run_qnn_for_n_qubits(n_qubits):
    X_all = qubit_angle_data[n_qubits]

    # Use the SAME test set, but smaller train subset
    X_train_full = X_all[train_idx]
```

```
    X_train_full = X_all[train_idx]
    y_train_full = y_all[train_idx]
    X_test  = X_all[test_idx]
    y_test  = y_all[test_idx]

    # further downsample train set for QNN speed
    if len(X_train_full) > MAX_SAMPLES_QNN:
        small_train_idx = rng.choice(len(X_train_full), size=MA
        X_train = X_train_full[small_train_idx]
        y_train = y_train_full[small_train_idx]
    else:
        X_train, y_train = X_train_full, y_train_full

    clf = build_qnn_classifier(n_qubits)
    clf.fit(X_train, y_train)

    train_acc = accuracy_score(y_train, clf.predict(X_train))
    test_acc  = accuracy_score(y_test,  clf.predict(X_test))

    return train_acc, test_acc
```

```
import numpy as np

# Reduce dataset size to prevent RAM explosion
MAX_SAMPLES_KERNEL = 1200    # Safe for QSVC (1000–1500 max)
MAX_SAMPLES_QNN    = 600     # QNN needs even fewer

# Convert labels to numpy
y_all = np.asarray(y).astype(int)
indices_all = np.arange(len(y_all))

# Random subset once (shared for all qubit settings)
rng = np.random.default_rng(42)
if len(indices_all) > MAX_SAMPLES_KERNEL:
    sub_idx = rng.choice(indices_all, size=MAX_SAMPLES_KERNEL,
else:
    sub_idx = indices_all

y_sub = y_all[sub_idx]
print("Total samples used:", len(sub_idx))

from sklearn.model_selection import train_test_split

train_idx, test_idx = train_test_split(
    sub_idx,
    test_size=0.3,
    random_state=42,
    stratify=y_sub
)

print("Train samples:", len(train_idx), "Test samples:", len(te
```

```
Total samples used: 1200
Train samples: 840 Test samples: 360
```

```
qsvc_results, qnn_results = {}, {}

for n in QUBIT_SETTINGS:
    if qubit_angle_data[n].shape[1] != n:
        print(f"Skipping {n} — feature dim mismatch.")
        continue

    print(f"n_qubits = {n}")

    tr_qsvc, te_qsvc = run_qsvc_for_n_qubits(n)
    qsvc_results[n] = (tr_qsvc, te_qsvc)
    print(f"QSVC  -> Train: {tr_qsvc:.3f} | Test: {te_qsvc:.3f}

    tr_qnn, te_qnn = run_qnn_for_n_qubits(n)
    qnn_results[n] = (tr_qnn, te_qnn)
    print(f"QNN   -> Train: {tr_qnn:.3f} | Test: {te_qnn:.3f}")

print("\n FINAL SUMMARY")
print("n_qubits | QSVC_test | QNN_test")
for n in qsvc_results:
    print(f"{n:7d} | {qsvc_results[n][1]:.3f}    | {qnn_results
```

```
n_qubits = 4
QSVC  -> Train: 0.996 | Test: 0.983
/tmp/ipython-input-1328438339.py:6: DeprecationWarning: The class ``
  sampler = Sampler()
/tmp/ipython-input-1328438339.py:7: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
QNN   -> Train: 0.005 | Test: 0.011
n_qubits = 6
QSVC  -> Train: 0.996 | Test: 0.981
/tmp/ipython-input-1328438339.py:6: DeprecationWarning: The class ``
  sampler = Sampler()
/tmp/ipython-input-1328438339.py:7: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
QNN   -> Train: 0.003 | Test: 0.000
n_qubits = 8
QSVC  -> Train: 0.996 | Test: 0.981
/tmp/ipython-input-1328438339.py:6: DeprecationWarning: The class ``
  sampler = Sampler()
/tmp/ipython-input-1328438339.py:7: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
QNN   -> Train: 0.000 | Test: 0.003
n_qubits = 10
```

Methodology

To evaluate quantum machine learning performance across different encoding dimensions, two QML models were implemented: a kernel-based Quantum Support Vector Classifier (QSVC) and a variational Quantum Neural Network (QNN). Both models were tested on qubit settings of 4, 6, 8, and 10, and the *same train/test split* was reused throughout to ensure a fair and controlled comparison.

For the QSVC, we used the `QSVC` class from Qiskit Machine Learning together with a `FidelityQuantumKernel`. The kernel internally computes pairwise fidelities between quantum states generated by a chosen feature map. To maintain consistency with earlier parts of the pipeline, the feature map was selected from the previously evaluated set (ZFeatureMap, ZZFeatureMap, or PauliFeatureMap) and configured with a fixed depth and entanglement pattern. QSVC training therefore relied entirely on kernel evaluations without requiring variational optimisation.

The QNN model was constructed using a `SamplerQNN`, which combines a data-encoding feature map with a parameterised variational ansatz. The ansatz used in this study was the `EfficientSU2` circuit, chosen for its expressive structure and compatibility with hardware-efficient layouts. A classical optimizer (COBYLA) was employed to update the trainable parameters of the ansatz based on the training loss. The QNN was wrapped using `NeuralNetworkClassifier` to provide a scikit-learn–style interface for training and prediction.

Across both models and all qubit counts, identical samples were used for training and testing, and accuracy was recorded to assess how model type, feature dimension, and circuit complexity affect predictive performance.

Part D — Ideal vs Depolarising Noise (AerSimulator)

```
!pip install qiskit-aer
```

```
Collecting qiskit-aer
  Downloading qiskit_aer-0.17.2-cp312-cp312-manylinux_2_17_x86_64.ma
Collecting qiskit>=1.1.0 (from qiskit-aer)
  Downloading qiskit-2.2.3-cp39-abi3-manylinux2014_x86_64.manylinux_
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/pytho
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12
Collecting rustworkx>=0.15.0 (from qiskit>=1.1.0->qiskit-aer)
  Downloading rustworkx-0.17.1-cp39-abi3-manylinux_2_17_x86_64.manyl
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.1
Collecting stevedore>=3.0.0 (from qiskit>=1.1.0->qiskit-aer)
  Downloading stevedore-5.6.0-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/p
Downloading qiskit_aer-0.17.2-cp312-cp312-manylinux_2_17_x86_64.many
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 139.3 MB/s
Downloading qiskit-2.2.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.0/8.0 MB 132.0 MB/s et
Downloading rustworkx-0.17.1-cp39-abi3-manylinux_2_17_x86_64.manylin
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 104.5 MB/s et
Downloading stevedore-5.6.0-py3-none-any.whl (54 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.4/54.4 kB 4.7 MB/s et
Installing collected packages: stevedore, rustworkx, qiskit, qiskit-
Successfully installed qiskit-2.2.3 qiskit-aer-0.17.2 rustworkx-0.17
```

```
import numpy as np
from sklearn.metrics import accuracy_score

from qiskit_aer import AerSimulator
from qiskit_aer.noise import NoiseModel, depolarizing_error

from qiskit.circuit.library import ZFeatureMap, ZZFeatureMap, P
from qiskit_machine_learning.kernels import FidelityQuantumKern
from qiskit_machine_learning.algorithms import QSVC, NeuralNetw
from qiskit_machine_learning.neural_networks import SamplerQNN
from qiskit_algorithms.optimizers import COBYLA

# Aer primitives (preferred). If unavailable, fall back to defa
try:
    from qiskit_aer.primitives import Sampler as AerSampler
    AER_PRIMITIVES_OK = True
except Exception:
    from qiskit.primitives import Sampler as AerSampler
    AER_PRIMITIVES_OK = False
```

```
# Build simulators
ideal_sim = AerSimulator(seed_simulator=42)

def build_depolarising_sim(p1=0.005, p2=0.05):
    """
    p1: depolarising prob for 1-qubit gates
    p2: depolarising prob for 2-qubit CX gates
    """
    noise_model = NoiseModel()
    # Qiskit Aer default basis: rz, sx, cx (u3 deprecated). :co
    noise_model.add_all_qubit_quantum_error(depolarizing_error(
    noise_model.add_all_qubit_quantum_error(depolarizing_error(
    noise_model.add_all_qubit_quantum_error(depolarizing_error(
    return AerSimulator(noise_model=noise_model, seed_simulator

noisy_sim = build_depolarising_sim(p1=5e-3, p2=5e-2)
```

```
# Feature map builder (same as earlier)
FEATURE_MAP_TYPE = "ZZ"    # "Z", "ZZ", or "Pauli"
FEATURE_MAP_REPS = 2
FEATURE_MAP_ENT  = "linear"

def build_feature_map(n_qubits):
    t = FEATURE_MAP_TYPE.lower()
    if t == "z":
        return ZFeatureMap(feature_dimension=n_qubits, reps=FEA
    if t == "zz":
        return ZZFeatureMap(feature_dimension=n_qubits, reps=FE
    if t == "pauli":
        return PauliFeatureMap(feature_dimension=n_qubits, reps
                               entanglement=FEATURE_MAP_ENT, pa
    raise ValueError("FEATURE_MAP_TYPE must be Z / ZZ / Pauli")

```

```
from qiskit_aer.primitives import Sampler as AerSampler

def build_sampler(simulator):
    """
    Safe sampler creation for different Qiskit versions.
    Newer versions use Sampler(backend) or Sampler(backend=...)
    so we try both and fall back to a plain Sampler if needed.
    """
    # Try new-style: Sampler(backend)
    try:
        return AerSampler(simulator)
    except TypeError:
        pass

    # Try keyword: Sampler(backend=...)
    try:
        return AerSampler(backend=simulator)
    except TypeError:
        pass

    # Fallback: no backend binding (will behave like default Sa
    return AerSampler()
```

```
# QSVC runner for a given simulator
def run_qsvc(n_qubits, simulator):
    X_all = qubit_angle_data[n_qubits]

    X_train = X_all[train_idx]
    X_test  = X_all[test_idx]
    y_train = y_all[train_idx]
    y_test  = y_all[test_idx]

    fm = build_feature_map(n_qubits)

    sampler = build_sampler(simulator)

    try:
        qkernel = FidelityQuantumKernel(feature_map=fm, sampler
    except TypeError:
        qkernel = FidelityQuantumKernel(feature_map=fm)

    model = QSVC(quantum_kernel=qkernel)
    model.fit(X_train, y_train)

    tr_acc = accuracy_score(y_train, model.predict(X_train))
    te_acc = accuracy_score(y_test,  model.predict(X_test))
    return tr_acc, te_acc
```

```
# QNN runner for a given simulator
def run_qnn(n_qubits, simulator):
    X_all = qubit_angle_data[n_qubits]

    X_train = X_all[train_idx]
    X_test  = X_all[test_idx]
    y_train = y_all[train_idx]
    y_test  = y_all[test_idx]

    feature_map = build_feature_map(n_qubits)
    ansatz = EfficientSU2(num_qubits=n_qubits, reps=1, entangle
    circuit = feature_map.compose(ansatz)

    sampler = build_sampler(simulator)

    qnn = SamplerQNN(
        sampler=sampler,
        circuit=circuit,
        input_params=feature_map.parameters,
        weight_params=ansatz.parameters,
    )

    clf = NeuralNetworkClassifier(
        neural_network=qnn,
        optimizer=COBYLA(maxiter=50),
        one_hot=False
    )

    clf.fit(X_train, y_train)

    tr_acc = accuracy_score(y_train, clf.predict(X_train))
    te_acc = accuracy_score(y_test,  clf.predict(X_test))
    return tr_acc, te_acc
```

```
# Compare Ideal vs Noisy for each qubit setting
QUBIT_SETTINGS = [4, 6]
results = []

for n in QUBIT_SETTINGS:
    if qubit_angle_data[n].shape[1] != n:
        print(f"Skipping {n} (feature dim mismatch).")
        continue

    print(f"n_qubits = {n}")

    # QSVC
```

```
    tr_i, te_i = run_qsvc(n, ideal_sim)
    tr_n, te_n = run_qsvc(n, noisy_sim)
    print(f"QSVC Ideal -> Train {tr_i:.3f} | Test {te_i:.3f}")
    print(f"QSVC Noisy -> Train {tr_n:.3f} | Test {te_n:.3f}")

    results.append(("QSVC", n, "ideal", tr_i, te_i))
    results.append(("QSVC", n, "noisy", tr_n, te_n))

    # QNN
    tr_i, te_i = run_qnn(n, ideal_sim)
    tr_n, te_n = run_qnn(n, noisy_sim)
    print(f"QNN  Ideal -> Train {tr_i:.3f} | Test {te_i:.3f}")
    print(f"QNN  Noisy -> Train {tr_n:.3f} | Test {te_n:.3f}")

    results.append(("QNN", n, "ideal", tr_i, te_i))
    results.append(("QNN", n, "noisy", tr_n, te_n))
```

```
n_qubits = 4
/tmp/ipython-input-2795801222.py:12: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
/tmp/ipython-input-2795801222.py:12: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
QSVC Ideal -> Train 0.996 | Test 0.983
QSVC Noisy -> Train 0.996 | Test 0.983
/tmp/ipython-input-917605501.py:14: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
/tmp/ipython-input-917605501.py:16: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
/tmp/ipython-input-917605501.py:14: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
/tmp/ipython-input-917605501.py:16: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
QNN  Ideal -> Train 0.004 | Test 0.003
QNN  Noisy -> Train 0.008 | Test 0.006
n_qubits = 6
/tmp/ipython-input-2795801222.py:12: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
/tmp/ipython-input-2795801222.py:12: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
QSVC Ideal -> Train 0.996 | Test 0.981
QSVC Noisy -> Train 0.996 | Test 0.981
/tmp/ipython-input-917605501.py:14: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
/tmp/ipython-input-917605501.py:16: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
/tmp/ipython-input-917605501.py:14: DeprecationWarning: Sampler has
  sampler = build_sampler(simulator)
/tmp/ipython-input-917605501.py:16: DeprecationWarning: V1 Primitive
  qnn = SamplerQNN(
QNN  Ideal -> Train 0.002 | Test 0.003
QNN  Noisy -> Train 0.045 | Test 0.031
```

```
print("Model | Qubits | Env   | Train Acc | Test Acc")
for m, q, env, tr, te in results:
    print(f"{m:<5} | {q:^6} | {env:<5} | {tr:.3f}     | {te:.3f}")
```

```
Model | Qubits | Env   | Train Acc | Test Acc
QSVC  |   4    | ideal | 0.996     | 0.983
QSVC  |   4    | noisy | 0.996     | 0.983
QNN   |   4    | ideal | 0.004     | 0.003
QNN   |   4    | noisy | 0.008     | 0.006
QSVC  |   6    | ideal | 0.996     | 0.981
QSVC  |   6    | noisy | 0.996     | 0.981
QNN   |   6    | ideal | 0.002     | 0.003
QNN   |   6    | noisy | 0.045     | 0.031
```