Capstone Project (Devashish Chaudhary): Quantum Autoencoder Embeddings + GNN for Network Anomaly Detection

Objective:

Implement a hybrid quantum-classical framework for network anomaly detection using Quantum Autoencoder (QAE) embeddings as input to a Graph Neural Network (GNN) for classification of normal vs anomalous network behavior. Students will learn how quantum representations can be leveraged in classical graph-based learning.

Project Overview

1. Train a Quantum Autoencoder (QAE) on network traffic features (or a tabular dataset simulating network traffic).
2. Minimize reconstruction loss to learn meaningful latent embeddings.
3. Extract the latent embeddings from the encoder part of the QAE.
4. Construct a graph representation of network data:
    a. Nodes = unique (src-dst ip pairs)
    b. Assign network flow features to nodes.
    c. Edges = use cosine similarity between node features to construct edges between nodes.
5. Train a Graph Neural Network (GNN) using QAE embeddings as node features to perform binary classification:
    a. Normal network behavior = 0
    b. Anomalous network behavior = 1
6. Evaluate the performance of anomaly detection.

Dataset:

- Each row in the dataset corresponds to one network flow between srcIP → dstIP.
- The row already contains preprocessed flow features (traffic statistics, flags, duration, payload size, etc.).
- For this project, each row is treated as a node in the graph, because:
    - A flow is uniquely defined by a srcIP–dstIP pair.
    - Every flow has its own feature vector.

Node Definition

- Node = one row of the dataset = one individual network flow
- Therefore, node features = the flow features from that row.
- After training the QAE, replace the original node features with the QAE latent embeddings.
- Training Set → 70%
    - Contains only normal flows
    - Used to train the QAE, build the graph, and train the GNN
- Testing Set → 30%
    - Contains a mixture of normal and anomalous flows

- Used to evaluate anomaly detection performance

Dataset can be downloaded from the following link:

https://drive.google.com/drive/folders/1YzHqrjw-gSw_uRJgVN0cd4qdRbSmZjrX?usp=sharing

How to use dataset?

1. Dataset has already been preprocessed from Bot-IoT dataset. Each row is a network flow feature between srcIP and dstIP (which means each row features can be assigned to nodes after getting QAE embeddings).
2. Since it is an unsupervised learning (trained only on normal), use one-class svm or k-means or any other similar method after getting final embeddings from graph.

Submission Requirement

1. **Jupyter Notebook**:
    a. QAE training & embeddings
    b. Graph construction
    c. GNN training & evaluation
    d. Plots: reconstruction loss, classification performance, confusion matrix
2. **Report (2 pages)** in ATIS format including:
    a. Dataset preparation
    b. QAE & GNN architecture choices
    c. Experimental results and discussion
    d. Anomaly detection performance metrics
3. Submit to quantumlab.deakin@gmail.com, cc: s224281473@deakin.edu.au
4. Due date: Just follow the official documentation.

Resources

1. **How to create a network graph?**
   Lo, W. W., Layeghy, S., Sarhan, M., Gallagher, M., & Portmann, M. (2021). E-graphsage: A graph neural network based intrusion detection system for iot. *arXiv preprint arXiv:2103.16329*. https://arxiv.org/abs/2103.16329

2. **Quantum Autoencoder:** https://qiskit-community.github.io/qiskit-machine-learning/tutorials/12_quantum_autoencoder.html

Capstone Project (Dev): Implement and understand QFL with datasets provided.

1. Create a QFL framework and perform experimental analysis.
2. Dataset to be used: IRIS, MNIST, CIFAR-10 dataset (You can use small samples for MNIST and CIFAR)
3. Use models: VQC or NeuralNetwork Classifier (SamplerQNN)
4. Run at least 10 communication rounds, at least 3 devices.
5. Plot global model performance, devices training performance.

What to submit?

1. Code run in jupyter notebook. Make sure all the results are in the Jupyter notebook/Google Colab.
2. Plot global model accuracy against communication rounds.
3. Write a two pages report describing following:
    a. How did you collect or prepare the dataset?
    b. Which model did you choose? VQC or NeuralNetworkClassifier or others? Why did you choose?
    c. Explain your results:
        i. Global accuracy for at least two datasets (IRIS or other)
        ii. At least show the performance of one device, its training accuracy.
4. Page format can be like this (Follow ATIS format link for format)
    a. Title
    b. Abstract
    c. Introduction
    d. Background
    e. Brief theoretical background
    f. Experimental Setup
    g. Results Explanation

Files to submit (2 Files):

1. Send code jupyter notebook with results (Github link also fine)
2. Pdf report
3. Submit to quantumlab@gmail.com cc: d.gurung@deakin.edu.au
4. Due date: Just follow the official documentation.

Resources:

1. Datasets:
    a. MNIST https://keras.io/api/datasets/mnist/
    b. CIFAR10 https://keras.io/api/datasets/cifar10/
    c. IRIS https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
2. Models:
    a. VQC Example https://qiskit-community.github.io/qiskit-machine-learning/tutorials/02a_training_a_quantum_model_on_a_real_dataset.html
    b. SamplerQNN

https://qiskit-community.github.io/qiskit-machine-learning/tutorials/02_neural_network_classifier_and_regressor.html

c. NeuralNetwork Classifer

https://qiskit-community.github.io/qiskit-machine-learning/stubs/qiskit_machine_learning.algorithms.NeuralNetworkClassifier.html

Capstone Project (Swathi Chandrasekhar): Adversarial Attacks on Quantum Anomaly Detectors Using GANs

Objective

Demonstrate, quantify, and analyze the vulnerability of a quantum anomaly–detection model to adversarial examples generated by a Generative Adversarial Network (GAN).

Students will:

- Implement a quantum anomaly detector (QAD) on a chosen dataset.
- Implement a classical anomaly detector on the same data.
- Design and train a GAN-based adversary to fool the QAD (and optionally the classical model).
- Evaluate and compare robustness, and clearly document and visualize your findings.

High-Level Overview

1. Build and validate a baseline quantum anomaly detector (QAD) using Qiskit
2. Build a classical anomaly detector as a baseline.
3. Design and train a GAN that generates samples that are misclassified by the QAD as "normal."
4. Evaluate the attack's effectiveness on the QAD and the classical baseline.
5. Prepare a comprehensive report with results, analysis, and visualizations.

Part 1: Data Preparation & Discretisation

You are encouraged to use a realistic intrusion/traffic dataset:

- Recommended:
    - BoT_IoT dataset, or
    - KDD99 / KDD Cup 1999 dataset

You may also choose any other dataset you like, but:

- You must clearly justify why that dataset is appropriate for this project (e.g., anomaly structure, availability of labels, dimensionality).
- Prior confirmation for a change of dataset is encouraged (though not strictly required). You may email the instructor to request approval for using another dataset for your capstone.

1. **Data Cleaning**
    - Handle missing values (drop or impute).
    - Remove obvious outliers if necessary.
2. **Feature Selection**
    - Encoding of categorical features (e.g., one-hot encoding) if needed.
    - Feature scaling (e.g., normalization to [0, 1] or standardization) to support stable training and quantum encoding.
3. Define:
    - A "normal" class (e.g., benign traffic).
    - One or more "anomalous/attack" classes.
4. Split the data into:
    - **Train set**: mostly or only normal samples (for QAD training).
    - **Validation set**: mainly normal, used for tuning.
    - **Test set**: mixture of normal and anomalies for final evaluation.

Hint:
You may perform additional dimensionality reduction (e.g., PCA) to bring the feature space down to something manageable for quantum encoding (e.g., 8–16 dimensions), but if you do this, explain and justify your choice in the report.

Part 2: Implement the Quantum Anomaly Detector (QAD)

You will build a QAD in **Qiskit**.

2.1 Encoding & Circuit:

- Encode selected features into a quantum state using, for example:
    - Angle encoding (e.g., RY(θ_i) for each feature i), or
    - Other encodings supported in Qiskit, as long as clearly documented.
- Qubit budget:
    - Use **≤ 16 qubits** (because Qiskit kernels can become unstable or crash beyond that).
    - **Ideal range:** 8 or 10 qubits.
- **Optional extension (if time and interest permit):**
    - Run experiments with **8, 10, 12, 16 qubits**, and show how performance and/or resource usage evolves in a **table** in your results section.
    - If the kernel crashes for some configurations, explicitly mention that in your report.

2.2 Model & Training:

- Build a parameterized quantum circuit (variational circuit) with:

    - Layers of single-qubit rotations and entangling gates,
    - A measurement scheme that outputs an anomaly "score" or class label.
- Train the circuit on normal (and possibly a subset of anomaly) data to distinguish normal vs anomalous patterns.

Hints (non-mandatory):

- Use Qiskit's variational circuit templates or your own design, but explain architecture choices (depth, entanglement pattern, etc.).
- You can treat anomaly detection as either:
    - A binary classification problem (normal vs anomaly), or
    - A one-class problem (model only normal data, anomalies have high "distance" or low "score").

Part 3: Implement the Classical Anomaly Detector

1. You must create a classical baseline on the **same feature representation** used by the QAD.
    - **Recommended:**
        - An **RBF kernel one-class SVM** as a baseline anomaly detector.
2. You are not restricted to this choice. You may choose any classical model (e.g., random forest, shallow neural network, isolation forest, etc.), but:
    - You must **provide clear reasoning** for your choice in the documentation (e.g., suitability for high-dimensional data, interpretability, standard use in anomaly detection literature, better performance, or simply just you are more familiar with the option etc.).
3. The classical model should be trained and evaluated on the same training/test splits as the QAD.

Part 4: Design the GAN-Based Adversary

- The goal of the GAN is to generate **adversarial anomalies** that the QAD misclassifies as "normal" while they remain qualitatively anomalous with respect to the dataset.
- **Basic structure:**
- **Generator (G):**
    - Input: latent vector z sampled from a simple distribution (e.g., Gaussian).
    - Output: synthetic sample in the feature space (same dimension as your QAD input).
- **Adversarial objective using QAD:**
    - Feed generated samples into the **frozen** (pre-trained) QAD.
    - Let the QAD output an anomaly score or probability s(x) (where "normal" has higher probability/score).

- Design a generator loss so that G is encouraged to produce samples that QAD classifies as *normal*. Example idea:
  - If p_normal(x) is the QAD's probability that x is normal, make the generator maximize p_normal(x) (or equivalently minimize –log(p_normal(x))).
- **Semantic anomaly constraint (optional but encouraged):**
  - You may add another discriminator or constraint to ensure that generated samples:
    - Remain far from the nominal cluster, or
    - Share statistical properties with known anomalies.

Hints (non-mandatory):

- Keep the first version simple: G vs QAD only.
- Later you can incorporate a secondary discriminator trained on real anomalies to maintain "anomalous" characteristics.
- Use standard deep-learning tools (e.g., PyTorch, TensorFlow) to implement G; interface them with Qiskit outputs as needed.

Part 5 Attack Evaluation and Analysis

1. You must quantitatively and qualitatively evaluate how successful the GAN-based adversary is against:
   - The **Quantum Anomaly Detector**, and
   - The **Classical Anomaly Detector** (if you also choose to attack it).
2. **Metrics (minimum expected):**
   - **Confusion matrices** (before and after attack) for:
     - QAD (clean test data vs adversarial test data),
     - Classical model (clean vs adversarial).
   - Common anomaly detection metrics:
     - True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), accuracy, precision/recall, F1 score, etc.
3. Changes in these metrics due to adversarial samples.

Visualizations:
Any plots, tables, or **any form of easy-to-understand visualization** is strongly encouraged. The following specific ideas are hints, not strict requirements:

- Decision-boundary plots for QAD and classical detector, **before and after attack** (feasible if you reduce to 2–3 principal components or use a 2-D projection).
- Training curves for **QAD loss** and **GAN generator loss** over epochs.
- Bar chart or heat-map comparing **FNR and FPR** across experimental conditions:
  - Clean vs attacked inputs,

    - Ideal simulator vs noisy simulator (if you add noise experiments).
- A **table** summarizing:
    - Number of qubits,
    - Circuit depth,
    - Number of trainable parameters,
    - Approximate inference time (or runtime per batch).

Note: Not all of these above must be included, but your report should have enough visual and tabular evidence to clearly support your conclusions.

Noise and Hardware Robustness (Optional)

This is an optional extension (not a requirement).

If you wish to go further, you may:

- Evaluate the QAD under:
    - Ideal simulator, and
    - Noisy simulator (with realistic noise models in Qiskit, e.g., depolarizing noise, readout errors, or a backend noise model).
- Re-run adversarial evaluation under noise and compare:
    - Whether noise reduces or increases adversarial success rates,
    - Whether performance degrades differently for clean vs adversarial samples.

Clearly state in your report whether you performed this part. It can strengthen your work but is not mandator.

What to Submit

1. Jupyter Notebook

Your notebook(s) must contain:

- Data loading & preprocessing.
- Feature Engineering
- QGAN architecture:
    - Quantum generator circuit.
    - Classical discriminator network.
- Training loop implementation.
- Plots:
    - Loss curves.
    - Real vs generated distribution visualisations.
- Evaluation metrics calculations.
- **Visual and Tabular Results**
- Confusion matrices (at minimum) for clean vs adversarial scenarios.

- Appropriate plots, tables, and/or heat maps, such as:
    - Decision boundary / decision-region plots (if feasible).
    - Training curves for QAD and GAN.
    - Bar charts or heat maps comparing performance across conditions.
    - Table summarizing qubit counts, circuit depth, parameters, and timing.

All cells should run top-to-bottom without manual edits.

2. Report (≈ 2 pages, ATIS format)

Suggested structure:

1. **Title**
2. **Abstract**
3. **Introduction**
4. **Dataset & Preprocessing**
5. **Methodology**
6. **Experiments & Results**
7. **Discussion & Limitations**
8. **Conclusion & Future Work**

3. Submission Instructions

Please email your **Jupyter Notebook(s)** and **PDF report** to:

- **quantumlab.deakin@gmail.com**
- **CC: swathi.chandrasekhar@deakin.edu.au**

If you have any problems understanding the task or issues working with QML/QGAN, contact: **swathi.chandrasekhar@deakin.edu.au**

Suggested Resources

- **Qiskit Machine Learning qGAN Tutorial** (PyTorch-based): (Qiskit Community)
- **Qiskit Finance – Option Pricing with qGANs** (example of application): (Qiskit Community)
- **Original qGAN paper**:
  Zoufal, Lucchi, Woerner (2019) – "Quantum Generative Adversarial Networks for Learning and Loading Random Distributions" (arXiv)

Capstone Project (Shanika Nanayakkara)

1. Project Title

Quantum Convolutional Neural Network (QCNN) for Real Breast Lesion (Cancer) Classification

Aim : Extend the single-client QCNN code into a two-client federated learning system with FedAvg, and analyse global vs local performance using the BreastMNIST dataset.

This project helps you understand:

- *How quantum layers can be added into classical deep learning*
- *How QCNN works for image classification*
- *How Federated Learning works (local training + global aggregation)*
- *How to evaluate global vs. local models*

Step 1 — Baseline QCNN (Single Client)

1. Implement the QCNN on the BreastMNIST subset (300 train, 100 test)
   Source: ATIS-2025_CapstonPoject_Resource/breastmnist_subset_300.csv at main · shanikairoshi/ATIS-2025_CapstonPoject_Resource
2. Record:
   a. Training loss per epoch
   b. Final test accuracy
3. Save at least one training curve (loss vs epoch) and report the final accuracy.

Resource : The Quantum Convolution Neural Network - Qiskit Machine Learning 0.7.1

Goal: Establish a centralized single-client baseline.

Step 2 — Data Partitioning into Two Clients

1. Take the same training subset used in the baseline.
2. Partition it into two disjoint subsets:
   a. Client 1 dataset
   b. Client 2 dataset
3. Spit it non-IID (one client more benign, other more malignant).

4. Print the number of samples and class balance per client.

Goal: Define two “virtual hospitals/devices” with local data.

Step 3 — Define the Client Abstraction

Using the QCNN model as the core:

1. Treat each client as an entity that has:
    a. Its own QCNN model (same architecture as baseline).
    b. Its own optimizer and training data.
2. Decide a fixed learning rate, number of local epochs per round, and batch size.

Goal: Formalize the notion of a federated client.

Step 4 — Local Training per Round

For each federated round:

1. For Client 1:
    a. Start from the current global model parameters.
    b. Perform the chosen number of local epochs.
    c. Record local training loss and accuracy.
2. Repeat the same process for Client 2.
3. At the end of the round, each client outputs its updated QCNN parameters.

Goal: Produce two locally trained models starting from a common global model.

Step 5 — Federated Averaging (FedAvg)

At the server (central aggregator):

1. Collect the model parameters from both clients at the end of each round.
2. Compute their simple arithmetic mean parameter-wise (FedAvg).
3. Define these averaged parameters as the new global QCNN model.
4. In the next round, broadcast this global model back to both clients as their starting point.

Goal: Implement one full FedAvg update step per round.

Step 6 — Global Evaluation

At the end of each federated round:

1. Evaluate the global QCNN model on the fixed test subset.
2. Record the global test accuracy after each round.
3. Optionally, also evaluate:
    a. Client 1's local model on the test set.
    b. Client 2's local model on the test set.

Goal: Track how the global model performance evolves across rounds.

Step 7 — Experimental Design

Run at least:

1. One baseline experiment:
    a. Centralized QCNN (single-client original code).
2. One federated experiment:
    a. Two clients, FedAvg, at least 10–50 communication rounds.

Optionally vary:

- Number of local epochs per round.

Goal: Generate a small but clear set of comparative experiments.

Step 8 — Results and Visualisation

Produce at minimum:

1. A plot of global test accuracy vs federated round.
2. A plot of client training accuracy vs round (for each client).
3. A table summarising:
    a. Baseline centralized QCNN accuracy.
    b. Final global accuracy after federated training.
    c. Final local accuracies of both clients.

Goal: Visually demonstrate the effect of federated averaging.

What to submit?

1. Code run in jupyter notebook. Make sure all the results are there in the Jupyter notebook/Google Colab.
2. A Short Report Including:

1. Introduction
    a. Briefly describe QCNN and federated learning.
2. Methodology
    a. Dataset choice and preprocessing.
    b. QCNN architecture (high-level).
    c. Federated setup (two clients, number of rounds, FedAvg).
3. Experiments and Results
    a. Baseline vs federated results.
    b. Plots and tables.
    c. Observations about convergence and client differences.
4. Conclusion
    a. What did you learn about combining QCNN and FL?
    b. Limitations and possible next steps (e.g., more clients, angle-aware aggregation).

Goal: Show understanding of both quantum model and federated training dynamics.

Files to submit (2 Files):

1. Send code jupyternotebook with results (github link also fine)
2. Pdf report
4. Submit to quantumlab@gmail, cc: s.nanayakkara@deakin.edu.au
5. Due date: Just follow the official documentation.

Capstone Project (Baobao): Balance Privacy and Utility in Quantum Variational Classification

1. Build a VQC classifier (SamplerQNN or EstimatorQNN) and evaluate it under noisy settings.
2. Dataset to be used: IRIS dataset (with all 4 features and 3 classes, 80% for training and 20% for test)
3. Use models: NeuralNetwork Classifier (SamplerQNN or EstimatorQNN), you can try different feature maps and ansatzes and pick the combination with the highest accuracy
4. Noise modeling

   Split the sum of privacy budget $\epsilon_{sum} = 1$ between two noise sources: depolarizing noise ϵ_{dep} and measurement noise ϵ_{mea} (e.g., ϵ_{dep}=0.2, $\epsilon_{mea} = 0.8$).

   You can calculate ε_{dep} by

$$\epsilon_{dep} = \ln\left[1 + \frac{(1-p)dD}{p}\right]$$

   where ε_{dep} is the depolarizing possibility, d is the trace distance (here we set $d = 0.01$), D is the dimension of the Hilbert space ($D = 2^n$, n is the number of qubits)

   You can calculate ϵ_{mea} by

$$\epsilon_{mea} = 4Ndr\left[\frac{-2Ndrc^2}{1-2Ndr}\right] + c + \frac{Ndr}{2}$$

   where c=0.15, N is the number of measurements (shots), d is the trace distance (here we set $d = 0.01$), and r is the maximum rank of measurement operators (the default readout in SamplerQNN is per-qubit computational-basis (Z-basis) projective measurement, and in this case, maximum rank r =1)

5. Plot a 3D surface of accuracy vs. (ϵ_{dep}, ϵ_{mea}). Summarize where accuracy is maximized and describe the ϵ allocation pattern.

What to submit?

1. Code run in jupyter notebook. Make sure all the results are there in the Jupyter notebook/Google Colab.
2. Plot the 3D surface of accuracy vs. (ϵ_{dep}, ϵ_{mea}).
3. Write a two pages report describing following:
   a. What and why you choose this combination of feature map and ansatz?
   b. Explain your results: with $\epsilon_{\mathrm{sum}} = 1$, what patterns do you observe from the 3D plot? In your view, what is the optimal allocation (i.e., combination of ϵ_{dep} and) and why is it optimal? Did you try other quantum circuit architectures, and if so, did the optimal ϵ -allocation change?
4. Page format can be like this (Follow ATIS format link for format)

    a. Title
    b. Introduction
    c. Experimental details
    d. Results Explanation

Files to submit (2 Files):

5. Send code jupyter notebook with results (github link also fine)
6. Pdf report
7. Submit to quantumlab.deakin@gmail.com, cc: baobao.song@deakin.edu.au
8. Due date: Just follow the official documentation.

Resources:

3. Datasets:
    a. IRIS https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
4. Models:
    a. SamplerQNN and EstimatorQNN

https://qiskit-community.github.io/qiskit-machine-learning/tutorials/02_neural_network_classifier_and_regressor.html

Capstone Project (Navneet Singh): Quantum Feature Map Design Competition Hybrid Amplitude + Angle Encoding on Genomic Benchmarks

Objective

Design, implement, and evaluate **hybrid quantum feature maps** that combine **amplitude encoding** and **angle encoding** with **different entanglement patterns**, and apply them across **all 9 Genomic Benchmarks datasets**.

Students will:

- Learn how **encoding + feature map choice** strongly impacts QML performance.
- Explore trade-offs between **amplitude vs angle encoding** and **entanglement topologies**.
- Compare **quantum kernel methods** (QSVC) and/or **variational classifiers** (VQC/QNN) against classical baselines.
- Generalise across **multiple real genomics datasets**, not just a toy example

Datasets

Use **all 9 datasets** from *Genomic Benchmarks*:

From the Python package (genomic-benchmarks) / GitHub: (GitHub)

- demo_coding_vs_intergenomic_seqs
- demo_human_or_worm
- dummy_mouse_enhancers_ensembl
- human_enhancers_cohn
- human_enhancers_ensembl
- human_ensembl_regulatory
- human_nontata_promoters
- human_ocr_ensembl
- drosophila_enhancers_stark (BioMed Central)

You may load them either via:

- pip install genomic-benchmarks and list_datasets(), info(), download_dataset() etc. (GitHub)
- or Hugging Face datasets under katarinagresova/Genomic_Benchmarks_* (Hugging Face)

Because full datasets are large, you may **subsample** each for quantum experiments, but you must:

- Use **all 9 datasets at least once**, and
- Clearly describe your subsampling strategy.

Project Overview

Theme: *"Who can design the most robust quantum feature map for genomics?"*

1. **Characterise** the 9 genomic datasets (lengths, classes, sizes).
2. Design **classical sequence encoders → fixed-dimensional vectors**.
3. Design **hybrid amplitude + angle quantum feature maps** with **different entanglement graphs**.
4. Plug these feature maps into:
    - **QSVC** with a **FidelityQuantumKernel** and
    - **QNN** models.
5. Run experiments on **at least one ideal simulator** and **noisy or hardware-like** backends.
6. Evaluate **accuracy + F1 + AUROC** and **complexity (qubits, depth, shots)** across all 9 datasets.
7. Write a short **ATIS-style report** + submit **Jupyter notebooks**.

Tasks & Milestones

1. Dataset Exploration

- Install and list the datasets:
- from genomic_benchmarks.data_check import list_datasets, info
- print(list_datasets())
- info("human_nontata_promoters", version=0)
- For each of the 9 datasets:
    - Record **#sequences**, **#classes**, **class ratio**, **median length**, **std length** (Table 1 in your report). (BioMed Central)
- Basic checks:
    - Plot class distribution per dataset.
    - Show a few example sequences per class.

2. Classical Preprocessing & Embeddings (Day 1–2)

Genomic sequences are text over {A, C, G, T}. You must design **at least two** classical encodings and choose one as your **main pipeline**:

Examples (you can pick others):

- **k-mer frequency vector** (e.g. 3-mer counts → $4^3 = 64$-dim vector).
- **Pretrained CNN / embedding from the official baseline model** then take the penultimate layer as a feature vector. (GitHub)

Then:

- **Standardise** features (e.g. StandardScaler) & apply **PCA** or autoencoder to reduce to **$d \leq 8$–10 features** (so quantum circuits stay small).
- Document:
    - Final feature dimension **d**.

- How much variance PCA keeps.
- Same encoding applied to all datasets (preferred), or justify dataset-specific tweaks.

3. Hybrid Quantum Feature Map Design

You must design **at least 4 different hybrid feature maps** that **combine amplitude + angle encoding** and vary **entanglement**.

3.1 General template

Let classical vector after preprocessing be $x \in \mathbb{R}d$

1. Choose **number of qubits** n so that $2n \geq d$ (for amplitude part).
2. **Amplitude stage**
   - Normalise first d components to unit norm and load into a state vector of length $2n$.
   - Use qc.initialize(amplitudes, range(n)) to perform amplitude encoding. (Quantum Computing Stack Exchange)
3. **Angle stage**
   - Rescale features into $[0,\pi]$.
   - Apply single-qubit rotations e.g. Ry(θ_i) or RZ(θ_i) on each qubit (angle encoding). (quantum.cloud.ibm.com)
4. **Entangling layers**
   - Insert entangling gates (CNOT / CZ / ZZ-style rotations) with different **connectivity graphs**.

3.2 Required designs

You must implement **at least these three** families and you need to add one more:

1. **Hybrid-Linear (H-Lin)**
   - Amplitude initialisation.
   - Single layer of Ry angle encoding.
   - **Linear entanglement**: chain (0–1–2–…–n-1) using CZ or CNOT.
2. **Hybrid-Full (H-Full)**
   - Amplitude initialisation.
   - Two alternating blocks:
     - ♣ (a) Ry angle encoding,
     - ♣ (b) **Full entanglement**: all-to-all CNOT or CZ (e.g. using Qiskit's entanglement='full' style as inspiration). (quantum.cloud.ibm.com)
   - Repeat for reps ≥ 2.
3. **Hybrid-Problem-Aware (H-Bio)**
   - Same amplitude+angle idea, but **design the entanglement graph to reflect biology**, e.g.:
     - ♣ Group qubits corresponding to k-mer positions or motif groups.
     - ♣ Star or ring structure where certain "motif" qubits are hubs.
   - You must draw and explain the **graph** you chose.

You can implement these as:

- Custom QuantumCircuit subclasses, or
- Parameterised functions returning QuantumCircuit given input (x).

You are encouraged (but not forced) to also compare with **standard Pauli/ZZFeatureMap** baselines from Qiskit. (quantum.cloud.ibm.com)

4. Models & Backends

4.1 Primary model: QSVC with FidelityQuantumKernel

- Use **FidelityQuantumKernel** with your custom feature map + **ComputeUncompute** + **Sampler** in Qiskit Machine Learning. (qiskit-community.github.io)
- Train **QSVC** on:
    - You can use **smaller subsamples (**100 training and 50 testing).

4.2 VQC / QNN

- Use feature map as **input layer** and your choice of variational ansatz (e.g. RealAmplitudes) and optimiser (e.g. COBYLA) for a **VQC**.
- Run at least on 1–2 datasets to compare **kernel vs variational approach**.

4.3 Backends

Minimum:

- **Ideal statevector or high-shot simulator** (e.g. AerSimulator with no noise).
- **Noisy simulator** (custom noise model) you have to bit flip, phase flit and phase damping error and fake **device backend matching** what you did in Day 3. (GitHub)

For each experiment, record:

- Number of **qubits**.
- Approximate **circuit depth**.
- **Number of shots**.
- **Training time** (rough order, not exact profiling).

5. Experiments & "Competition Metric"

For each dataset and each feature map:

- Use a **fixed train/test split**.
- Train model(s) and record:
    - **Accuracy**
    - **F1 score (macro)**
    - **AUC (if binary)**
    - **Confusion matrix**
- Compute for each feature map:
    - **Average macro F1 across all 9 datasets**.

- ♣ A simple **complexity score**
- ♣ A **"score"** you design, e.g.
  Score=Avg F1$-\lambda\cdot\log C$

Your report should make a **case for which feature map "wins"** the competition and why.

Ablation study (required):

- Compare at least on one dataset:
  - **Amplitude-only** vs **Angle-only** vs **Hybrid** feature map.
- Compare **Linear vs Full vs Problem-Aware entanglement** for one dataset.

6. Visualisation & Analysis

You should include:

- **Kernel matrices** (heatmaps) for at least one dataset and feature map:
  - Show clear block structure vs "noisy" kernels.
- **2D visualisation** (PCA / t-SNE) of:
  - Classical features vs quantum embedding (e.g. use kernel PCA on quantum kernel).
- Circuit diagrams:
  - Draw and annotate your **Hybrid feature maps** (one example circuit per design).
- Discussion:
  - When does hybrid encoding help?
  - Which datasets are harder for quantum models?
  - How does **entanglement** affect performance vs depth/noise?

Submission Requirements

1. Jupyter Notebook

You can use one notebook, but make sure:

- **Data loading & preprocessing** for all 9 datasets.
- Implementation of **hybrid feature maps**
- Training scripts for **QSVC** (and VQC if used).
- Plots:
  - Class distributions, kernel heatmaps, embedding visualisations.
  - Accuracy/F1 vs feature map, per dataset.
- Tables summarising **results across 9 datasets**.
- All cells are **runnable top-to-bottom** and include **comments**.

2. Report (≈ 2 pages, ATIS format)

Suggested structure:

1. **Title**
   "Quantum Feature Map Design Competition: Hybrid Amplitude–Angle Encodings on Genomic Benchmarks"
2. **Abstract**
   Short summary of your best feature map, key results across 9 datasets.
3. **Introduction & Background**
   - Short recap of QML feature maps, amplitude vs angle encoding, and kernel methods.
4. **Datasets & Preprocessing**
   - Overview table of 9 datasets.
   - Description of sequence → feature pipeline.
5. **Hybrid Feature Map Design**
   - Clear description + diagrams of H-Lin, H-Full, H-Bio.
   - Rationale for entanglement choices.
6. **Experimental Setup**
   - Models (QSVC / VQC), backends, shot counts, subsampling.
7. **Results**
   - Tables of metrics per dataset and feature map.
   - Ablation results (amplitude vs angle vs hybrid; entanglement types).
8. **Discussion**
   - Which feature map "wins" and why?
   - Generalisation across datasets.
   - Limitations (small qubit counts, simulation cost).
9. **Conclusion & Future Work**
   - Ideas for deeper circuits, better classical embeddings, or real-device runs.

Resources

- **Genomic Benchmarks GitHub & package** (install + usage, list datasets, info, download). (GitHub)
- **Hugging Face datasets (9 genomic benchmarks)** – all under katarinagresova. (Hugging Face)
- **Qiskit Machine Learning tutorials**:
  - Quantum kernel methods & QSVC (qiskit-community.github.io)
  - Data encoding & feature maps (angle / amplitude). (quantum.cloud.ibm.com)
- Amplitude encoding examples in Qiskit. (Medium)

Submission Instructions
Please email your **Jupyter Notebook(s)** and **PDF report** to
quantumlab.deakin@gmail.com and **CC:** n.navneetsingh@deakin.edu.au

If you have any problems understanding the task or encounter issues while working with QML, please contact: n.navneetsingh@deakin.edu.au