Day - Task Sheet

Topics:

1. **Simulating multiple quantum clients using local QNNs**
2. **Implementing federated parameter aggregation schemes**
3. **Differential Privacy on client updates**

Description:

This worksheet guides students through

- Simulate a QFL setup with multiple quantum clients, each training a local variational QNN (same circuit structure).
- Implement server-side aggregation (FedAvg) and monitor convergence across rounds.
- Study privacy and robustness by adding Gaussian noise to client updates.

Requirement:

Part A — Simulate multiple quantum clients using local QNNs
Objective: Create K quantum clients, each training a local QNN on its data split.

• Load a small binary classification dataset (4 features).

• Split the training set into K parts (e.g., K=4).

• Define a shared QNN architecture (same circuit for all clients).

Data encoding: ZZFeatureMap

Ansatz: TwoLocal

Optimizer: COBYLA

• For each client, train locally for E epochs from a common random init; print per-epoch loss/accuracy

Sub-task 1 – Change the reps and entanglement of ansatz

- Change the reps=1 to reps=2.
- Change the entanglement='linear' to ring'.
- Record the loss.

Question: Which entanglement stabilizes training fastest?

Sub-task 2 – Noise Check

- run with shots=None vs. shots=100
- note loss variance.

Question: How do finite shots affect local updates?

Part B — FedAvg

Objective: Implement server–client rounds with Federated Averaging.

• Initialize global parameters.

• For each federated round

Server → Clients: broadcast parameters

Clients: set local parameters; train 20 local epochs on their shard; send updated parameters back

• Server aggregation (FedAvg).

• Perform Bell-basis decoding at Bob's side using CX and H gates followed by measurement.

• After each round (total 20 rounds), evaluate the global model on a held-out test set and log test accuracy.

Sub-task – Client sampling

- each round randomly select m(3) < K(4) clients.

Part C — Gaussian Noise on Updates

Objective: Study privacy–utility trade-off by perturbing client updates.

• After local training and before sending parameters, each client adds clipped Gaussian noise.

• Sweep noise levels: $\sigma \in \{0.01, 0.05, 0.1\}$.

• For each σ, run 20 federated rounds; record final test accuracy

Part D — **Teleportation-based parameter transfer (quantum communication channel)**

Objective**:** Replace the classical "send parameters" step with a simulated **quantum teleportation** of parameter values from each client to the server.

Sub task 1- Parameter teleportation

1. Implement a **single-parameter teleportation** primitive:
   a) Encode a real parameter θ\thetaθ into a one-qubit state using an Ry(θ)R_y(\theta)Ry(θ) rotation.
   b) Prepare a Bell pair shared between client (Alice) and server (Bob).
   c) Perform the standard teleportation protocol:
   d) Decode the parameter θ^ back from Bob's qubit (e.g. from amplitudes).

Wrap this into:

```
def teleport_and_decode(theta: float) -> float:
    """Teleport parameter theta and return teleported value theta_hat."""
```

2. Extend this to teleport an entire **parameter tensor** (e.g. a weight vector):

```
def teleport_tensor(tensor) -> tensor_teleported:
    """Apply teleport_and_decode element-wise to a tensor."""
```

3. Extend this further to teleport an entire **state_dict** of a client model:

```
def teleport_state_dict(state_dict) -> teleported_state_dict:
    """Teleport all tensors in the PyTorch/QNN state_dict."""
```

4. Modify the federated loop:
   a) After local training, instead of sending raw θ from client i to the server, send the teleported version of the client model.

b) The server then runs FedAvg over **teleported** parameters to obtain the new global state.

5. Run federated training with teleportation-based communication (no DP Gaussian noise and no finite-shot noise initially).

Deliverables

- Short description (or pseudocode) of the teleportation primitive you implemented.
- Plot: test accuracy vs federated round for
  - classical communication (Subtask 2),
  - teleported communication (this subtask),

  on the same axes for comparison.

Part E — **Fidelity-based security and integrity check**

Objective**:** Quantify how faithfully the teleported parameters reproduce the original encoded states by using **state fidelity** as a security/integrity indicator.

SubTask 1:

1. For a given parameter θ:
   a) Construct the **original encoded state** $|\psi(\theta)$ (via Ry(θ) on $|0\rangle$.
   b) Obtain the **teleported state** $|\psi tele(\theta)\rangle$ from your teleportation circuit.
   c) Compute the state fidelity

   using qiskit.quantum_info.state_fidelity.

2. Extend this to the tensor and model level:
   a) For each parameter tensor, compute the **average fidelity** across its entries.

   For each client in each federated round, compute an **average client fidelity.**

3. During teleportation-based QFL training (Subtask 5), log for each round t:

   a) Fmean(t)
   b) global test accuracy.

4. Define a **security threshold** for fidelity,

   Flag any federated rounds where the mean fidelity falls below this threshold.

5. Compare:

   a) Accuracy vs round (with teleportation),
   b) Fidelity vs round,

   and discuss whether drops in fidelity correlate with degradation in accuracy.

Deliverables

- Table or CSV containing per-round fidelity statistics
- Plot: fidelity vs federated round.
- Short "security commentary" (5–8 sentences) explaining:
  - how high fidelity supports secure/faithful parameter transfer,
  - what low-fidelity rounds might indicate (noise, attack, or implementation issues),
  - how teleportation + fidelity complements DP noise and measurement noise in your overall QFL pipeline.

Part F— Visualization.

Objective: Clearly present convergence and privacy–utility trends.

Required plots

1. Global test accuracy vs. federated round
2. Average client training loss vs. round
3. Final test accuracy vs. noise level σ
4. Client-wise accuracy boxplot at the final round

Reflection

• Compare centralized training vs. QFL

• Explain how FedAvg aggregates client updates

• Summarize the most stable QNN design choices

• Analyze the privacy–utility trade-off: how adding Gaussian noise (σ) to updates impacts accuracy

Submission Details:

1. You must ensure that all your project files used for this task sit in a directory
2. All files are required to be uploaded
3. Please make sure that tutor have access to the folder.