import os, sys, json, time, uuid, logging, zipfile

from pathlib import Path

from datetime import datetime

RUN\_TS = datetime.now().strftime("%Y%m%d\_%H%M%S")

RUN\_ID = f"qcaa\_{RUN\_TS}\_{uuid.uuid4().hex[:6]}"

LOG\_DIR = Path("log3"); LOG\_DIR.mkdir(parents=True, exist\_ok=True)

logger = logging.getLogger(RUN\_ID)

logger.setLevel(logging.INFO)

fmt = logging.Formatter("[%(asctime)s] %(levelname)s - %(message)s", "%Y-%m-%d %H:%M:%S")

fh = logging.FileHandler(LOG\_DIR / "run.log", encoding="utf-8")

fh.setFormatter(fmt); fh.setLevel(logging.INFO); logger.addHandler(fh)

sh = logging.StreamHandler(sys.stdout)

sh.setFormatter(fmt); sh.setLevel(logging.INFO); logger.addHandler(sh)

JSONL\_PATH = LOG\_DIR / "run.jsonl"

def log\_json(event: str, \*\*kwargs):

rec = {"ts": datetime.now().isoformat(), "run\_id": RUN\_ID, "event": event, \*\*kwargs}

with open(JSONL\_PATH, "a", encoding="utf-8") as f:

f.write(json.dumps(rec, ensure\_ascii=False) + "\n")

def log\_kv(message: str, \*\*kwargs):

logger.info(f"{message} | " + " ".join(f"{k}={v}" for k,v in kwargs.items()))

log\_json(message, \*\*kwargs)

log\_kv("logging\_initialized", run\_id=RUN\_ID, log\_dir=str(LOG\_DIR.resolve()))

from qiskit\_ibm\_runtime import QiskitRuntimeService, SamplerV2 as Sampler

from qiskit import QuantumCircuit, transpile

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import roc\_curve, auc

from scipy.optimize import brentq

from scipy.interpolate import interp1d

import time

IBM\_TOKEN = "Zvc1ek7JzvNpGUngoMgAKkmMIN3VW8LsVmiJZCinA\_Me"

IBM\_INSTANCE = "crn:v1:bluemix:public:quantum-computing:us-east:a/72131a34c31f4e57a0c0f9708c75f533:69d4c9ef-8a66-4a4f-b935-4a0c6b58aa6e::"

DATA\_PATH = r"C:\Users\Sandip Dutta\Downloads\dataset.csv" # earlier dataset path # earlier dataset path

SELECTED\_COLS = ['flight\_time\_mean', 'hold\_time\_std', 'gyro\_alpha', 'accel\_y']

SHOTS = 8192

log\_kv("config\_set", data\_path=DATA\_PATH, shots=SHOTS, cols=",".join(SELECTED\_COLS))

t0 = time.time()

service = QiskitRuntimeService(channel="ibm\_cloud", token=IBM\_TOKEN, instance=IBM\_INSTANCE)

backend = service.backend("ibm\_brisbane")

print(f"✅ Connected to backend: {backend.name}")

log\_kv("backend\_connected", backend=backend.name, seconds=round(time.time()-t0, 3))

df = pd.read\_csv(DATA\_PATH)

X = df[SELECTED\_COLS].values

y = df['label'].astype(int).values

scaler = MinMaxScaler()

X\_norm = scaler.fit\_transform(X)

print("📊 Dataset Info:")

print(f"Total samples: {len(X)}")

print(f"Feature shape: {X\_norm.shape}")

print(f"Genuine: {int(y.sum())}, Impostor: {len(y) - int(y.sum())}")

log\_json("dataset\_loaded",

path=DATA\_PATH, n\_samples=len(X), n\_features=X.shape[1],

n\_genuine=int(y.sum()), n\_impostor=int(len(y)-y.sum()),

feature\_cols=SELECTED\_COLS)

def build\_qcaa\_circuit(x):

d = len(x)

qc = QuantumCircuit(d)

for i, xi in enumerate(x):

xi\_clipped = np.clip(xi, 0.001, 0.999)

theta = 2 \* np.arccos(np.sqrt(1 - xi\_clipped))

qc.ry(theta, i)

for i in range(d - 1):

qc.cz(i, i + 1)

qc.measure\_all()

return qc

circuits = [transpile(build\_qcaa\_circuit(x), backend=backend, optimization\_level=3) for x in X\_norm]

log\_kv("circuits\_compiled", n\_circuits=len(circuits))

sampler = Sampler(mode=backend)

print("🚀 Submitting job to real quantum backend...")

job = sampler.run(circuits, shots=SHOTS)

job\_id = job.job\_id()

print(f"🆔 Job ID: {job\_id}")

print("⏳ Waiting for result...")

log\_kv("job\_submitted", job\_id=job\_id, shots=SHOTS, n\_circuits=len(circuits))

result = job.result()

print("✅ Job completed.")

log\_kv("job\_completed", job\_id=job\_id)

with open(LOG\_DIR / "job\_meta.txt", "w", encoding="utf-8") as f:

f.write(f"RUN\_ID={RUN\_ID}\njob\_id={job\_id}\nbackend={backend.name}\ninstance={IBM\_INSTANCE}\n")

log\_kv("job\_meta\_saved", path=str((LOG\_DIR/'job\_meta.txt').resolve()))

reconstructed = []

reconstruction\_errors = []

counts\_pub0 = None

for i, circ\_res in enumerate(result):

num\_qubits = len(SELECTED\_COLS)

prob\_1 = np.zeros(num\_qubits)

counts = None

try:

counts = circ\_res.data.meas.get\_counts()

except Exception:

try:

counts = circ\_res.get\_counts()

except Exception:

counts = None

if counts is None:

log\_kv("counts\_unavailable", circuit=i)

continue

if i == 0:

counts\_pub0 = counts

total\_shots = max(sum(counts.values()), 1)

for bitstring, cnt in counts.items():

bits = bitstring[::-1]

for q in range(min(num\_qubits, len(bits))):

if bits[q] == '1':

prob\_1[q] += cnt

prob\_1 = prob\_1 / total\_shots

x\_hat = np.zeros(num\_qubits)

for j in range(num\_qubits):

p = float(np.clip(prob\_1[j], 1e-10, 1-1e-10))

x\_hat[j] = np.arcsin(np.sqrt(p))

reconstructed.append(x\_hat)

mse = float(np.mean((X\_norm[i] - x\_hat) \*\* 2))

reconstruction\_errors.append(mse)

if i < 3:

print(f"\nSample {i+1} - Label: {int(y[i])}")

print("Original x :", np.round(X\_norm[i], 3))

print("Reconstructed x̂:", np.round(x\_hat, 3))

print(f"Reconstruction MSE: {mse:.6f}")

reconstruction\_errors = np.array(reconstruction\_errors)

pd.DataFrame({

"index": np.arange(len(reconstruction\_errors)),

"label": y[:len(reconstruction\_errors)],

"mse": reconstruction\_errors

}).to\_csv(LOG\_DIR / "recon\_errors.csv", index=False)

log\_kv("recon\_errors\_saved", path=str((LOG\_DIR/'recon\_errors.csv').resolve()))

if counts\_pub0 is not None:

import json as \_json

with open(LOG\_DIR / "counts\_pub0.json", "w", encoding="utf-8") as f:

\_json.dump(counts\_pub0, f, ensure\_ascii=False, indent=2)

log\_kv("counts\_pub0\_saved", path=str((LOG\_DIR/'counts\_pub0.json').resolve()))

def calculate\_eer(fpr, tpr):

return float(brentq(lambda x: 1. - x - interp1d(fpr, tpr)(x), 0., 1.))

valid\_n = min(len(reconstruction\_errors), len(y))

fpr, tpr, thresholds = roc\_curve(y[:valid\_n], -reconstruction\_errors[:valid\_n])

auc\_val = float(auc(fpr, tpr))

eer\_val = calculate\_eer(fpr, tpr)

pd.DataFrame({"fpr": fpr, "tpr": tpr, "thr": thresholds}).to\_csv(LOG\_DIR / "roc\_points.csv", index=False)

log\_kv("roc\_points\_saved", path=str((LOG\_DIR/'roc\_points.csv').resolve()))

with open(LOG\_DIR / "summary.txt", "w", encoding="utf-8") as f:

f.write(f"RUN\_ID: {RUN\_ID}\n")

f.write(f"Backend: {backend.name}\n")

f.write(f"Job ID: {job\_id}\n")

f.write(f"Samples: {valid\_n} / {len(X\_norm)}\n")

f.write(f"Average Reconstruction MSE: {np.mean(reconstruction\_errors):.4f}\n")

f.write(f"AUC: {auc\_val:.4f}\n")

f.write(f"EER: {eer\_val:.4f}\n")

log\_kv("summary\_saved",

avg\_mse=round(float(np.mean(reconstruction\_errors)), 6),

auc=round(auc\_val, 6), eer=round(eer\_val, 6))

print("\n📊 Performance Summary:")

print(f"- Average Reconstruction MSE: {np.mean(reconstruction\_errors):.4f}")

print(f"- AUC: {auc\_val:.4f}")

print(f"- Equal Error Rate (EER): {eer\_val:.4f}")

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label=f"AUC={auc\_val:.2f}, EER={eer\_val:.2f}")

plt.plot([0, 1], [0, 1], linestyle="--")

plt.xlim([0, 1]); plt.ylim([0, 1])

plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")

plt.title("QCAA ROC (Non-Optimized)")

plt.legend(loc="lower right"); plt.grid(True)

plt.savefig(LOG\_DIR / "roc\_plot.png", dpi=300)

plt.show()

log\_kv("roc\_plot\_saved", path=str((LOG\_DIR/'roc\_plot.png').resolve()))

zip\_path = Path("log3.zip")

with zipfile.ZipFile(zip\_path, 'w', zipfile.ZIP\_DEFLATED) as zipf:

for file\_path in LOG\_DIR.rglob("\*"):

zipf.write(file\_path, arcname=file\_path.relative\_to(LOG\_DIR))

print(f"📦 All logs saved and zipped to: {zip\_path.resolve()}")

log\_kv("zip\_created", zip\_path=str(zip\_path.resolve()))