# === Table 4 (QCAA-only): Reconstruction Error Analysis Across Data Types ===

# - Real hardware: ibm\_brisbane (SamplerV2)

# - Methods: QCAA (Non-Optimized) and QCAA (Optimized) ONLY

# - Saves all results to log7/

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

from pathlib import Path

from datetime import datetime

# ---------------- Logging ----------------

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

RUN\_ID = f"qcaa\_table4\_only\_{RUN\_TS}\_{uuid.uuid4().hex[:6]}"

LOG\_DIR = Path("log7"); 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 / "table4\_only\_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 / "table4\_only\_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()))

# ---------------- Config ----------------

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

from qiskit import QuantumCircuit, transpile

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

# SA-based feature selection for QCAA-Optimized

from neal import SimulatedAnnealingSampler

from dimod import BinaryQuadraticModel

# Your credentials

IBM\_TOKEN = "wZdn6wKKMk4l3XICKoXmAheXQWSsu5JbbBA9Wrh9vpBc"

IBM\_INSTANCE = "crn:v1:bluemix:public:quantum-computing:us-east:a/6148794ccc8942b0b186858407d6ee44:afcd21b9-6c50-40a2-abe9-de5231e2324f::"

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

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

LABEL\_CANDS = ["label", "y", "target", "Label", "Target"]

DTYPE\_CANDS = ["data\_type", "datatype", "type", "category"] # e.g., Free / Fixed / Gait

SHOTS = 8192

GLOBAL\_SA\_SEED = 7

SA\_READS = 50

MIN\_ON = 2

# ---------------- Connect backend ----------------

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))

# ---------------- Load dataset ----------------

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

# Label

label\_col = next((c for c in LABEL\_CANDS if c in df.columns), None)

if label\_col is None:

raise ValueError("No label column found. Please include 'label' (or y/target).")

# Data type (optional)

dtype\_col = next((c for c in DTYPE\_CANDS if c in df.columns), None)

# Features

if all(c in df.columns for c in FEATURES):

feature\_cols = FEATURES

else:

feature\_cols = [c for c in df.columns if c != label\_col and (dtype\_col is None or c != dtype\_col)

and np.issubdtype(df[c].dtype, np.number)]

if not feature\_cols:

raise ValueError("No numeric feature columns found besides the label (and data\_type).")

X = df[feature\_cols].to\_numpy(dtype=float)

y = df[label\_col].astype(int).to\_numpy()

groups = (df[dtype\_col].astype(str).fillna("Unknown").to\_numpy() if dtype\_col else np.array(["All Data"] \* len(df)))

scaler = MinMaxScaler()

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

log\_kv("dataset\_loaded",

n\_samples=len(X\_norm), n\_features=X\_norm.shape[1],

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

feature\_cols=",".join(feature\_cols), dtype\_col=(dtype\_col or "None"))

print("📊 Dataset Info:")

print(f"Total samples: {len(X\_norm)} | Features: {X\_norm.shape[1]}")

print(f"Data-type: {dtype\_col or 'None'}")

# ---------------- Helpers ----------------

def to\_bits(key, width):

if isinstance(key, int):

return format(key, f"0{width}b")

if isinstance(key, str):

s = key.replace(" ", "")

if set(s) <= {"0","1"} and len(s) == width:

return s

try:

return format(int(s, 2), f"0{width}b")

except Exception:

return s.zfill(width)[:width]

if isinstance(key, tuple):

try:

return "".join("1" if bool(v) else "0" for v in key).zfill(width)[-width:]

except Exception:

return "".join(str(v) for v in key).zfill(width)[-width:]

return format(0, f"0{width}b")

def sampler\_prob\_ones(circuits\_t, label):

"""Run transpiled circuits; return P(|1>) per qubit array."""

sampler = Sampler(mode=backend)

print(f"🚀 Submitting job ({label}) to real quantum backend...")

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

job\_id = job.job\_id()

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

print(f"🆔 Job ID ({label}): {job\_id}")

print("⏳ Waiting for result...")

res = job.result()

print(f"✅ Job completed ({label}).")

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

num\_qubits = len(feature\_cols)

p1 = np.zeros((len(circuits\_t), num\_qubits), dtype=float)

if hasattr(res, "quasi\_dists"):

qlist = res.quasi\_dists

for i, qdist in enumerate(qlist):

row = np.zeros(num\_qubits)

for k, prob in dict(qdist).items():

bits = to\_bits(k, num\_qubits)[::-1]

for q in range(num\_qubits):

if bits[q] == "1":

row[q] += float(prob)

p1[i] = row

# Save a sample counts-like dump

try:

counts\_pub0 = {to\_bits(k, num\_qubits): int(float(v)\*SHOTS) for k, v in dict(qlist[0]).items()}

with open(LOG\_DIR / f"counts\_pub0\_{label}.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 / f"counts\_pub0\_{label}.json").resolve()), tag=label)

except Exception:

pass

else:

log\_kv("warning\_no\_quasi\_dists", note="Result has no quasi\_dists; API may have changed.", tag=label)

return p1

def recon\_error\_from\_p1(P1):

x\_hat = np.arcsin(np.sqrt(np.clip(P1, 1e-12, 1-1e-12)))

return ((X\_norm - x\_hat) \*\* 2).mean(axis=1)

# ---------------- QCAA (Non-Optimized) ----------------

def build\_circuit\_qcaa(x\_vec):

d = len(x\_vec)

qc = QuantumCircuit(d)

for i in range(d):

xi = float(np.clip(x\_vec[i], 1e-9, 1-1e-9))

theta = 2.0 \* np.arcsin(np.sqrt(xi))

qc.ry(theta, i)

for i in range(d - 1):

qc.cz(i, i + 1)

qc.measure\_all()

return qc

circuits\_qcaa = [build\_circuit\_qcaa(x) for x in X\_norm]

circuits\_qcaa\_t = [transpile(c, backend=backend, optimization\_level=3) for c in circuits\_qcaa]

log\_kv("circuits\_qcaa\_transpiled", n=len(circuits\_qcaa\_t))

p1\_qcaa = sampler\_prob\_ones(circuits\_qcaa\_t, "qcaa")

re\_qcaa = recon\_error\_from\_p1(p1\_qcaa)

pd.DataFrame({"index": np.arange(len(re\_qcaa)), "group": (groups), "mse": re\_qcaa})\

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

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

# ---------------- QCAA (Optimized) ----------------

def get\_feature\_mask\_sa(d, seed=None, reads=50, min\_on=2):

rng = np.random.default\_rng(seed)

linear = {i: float(rng.uniform(-1, 1)) for i in range(d)}

quadratic = {(i, j): float(rng.uniform(-1, 1)) for i in range(d) for j in range(i+1, d)}

bqm = BinaryQuadraticModel(linear, quadratic, 0.0, vartype='BINARY')

response = SimulatedAnnealingSampler().sample(bqm, num\_reads=reads, seed=seed)

mask = list(response.first.sample.values())

if sum(mask) < min\_on:

by\_bias = sorted(range(d), key=lambda i: abs(linear[i]), reverse=True)[:min\_on]

for i in by\_bias: mask[i] = 1

return mask

masks = [get\_feature\_mask\_sa(X\_norm.shape[1], seed=GLOBAL\_SA\_SEED+i, reads=SA\_READS, min\_on=MIN\_ON)

for i in range(len(X\_norm))]

pd.DataFrame(masks, columns=[f"f{j}" for j in range(len(feature\_cols))])\

.to\_csv(LOG\_DIR / "feature\_masks\_qcaa\_opt.csv", index=False)

log\_kv("feature\_masks\_generated", n=len(masks))

def build\_circuit\_qcaa\_opt(x\_vec, mask):

d = len(x\_vec)

qc = QuantumCircuit(d)

active = [i for i,m in enumerate(mask) if m==1]

for i in active:

xi = float(np.clip(x\_vec[i], 1e-9, 1-1e-9))

theta = 2.0 \* np.arcsin(np.sqrt(xi))

qc.ry(theta, i)

for a,b in zip(active[:-1], active[1:]):

qc.cz(a, b)

qc.measure\_all()

return qc

circuits\_opt = [build\_circuit\_qcaa\_opt(x, m) for x,m in zip(X\_norm, masks)]

circuits\_opt\_t = [transpile(c, backend=backend, optimization\_level=3) for c in circuits\_opt]

log\_kv("circuits\_qcaa\_opt\_transpiled", n=len(circuits\_opt\_t))

p1\_qcaaopt = sampler\_prob\_ones(circuits\_opt\_t, "qcaa\_opt")

re\_qcaaopt = recon\_error\_from\_p1(p1\_qcaaopt)

pd.DataFrame({"index": np.arange(len(re\_qcaaopt)), "group": (groups), "mse": re\_qcaaopt})\

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

log\_kv("recon\_errors\_saved", path=str((LOG\_DIR / "recon\_errors\_qcaa\_opt.csv").resolve()), tag="qcaa\_opt")

# ---------------- Aggregate metrics (QCAA & QCAA-Optimized ONLY) ----------------

def agg\_metrics(errs):

errs = np.asarray(errs, dtype=float)

mean = float(np.mean(errs))

std = float(np.std(errs, ddof=1)) if len(errs) > 1 else 0.0

var = float(np.var(errs, ddof=1)) if len(errs) > 1 else 0.0

mmax = float(np.max(errs)) if len(errs) > 0 else 0.0

return mean, std, var, mmax

records = []

unique\_groups = list(pd.unique(groups))

for g in unique\_groups:

idx = (groups == g)

m2, s2, v2, M2 = agg\_metrics(re\_qcaa[idx])

m3, s3, v3, M3 = agg\_metrics(re\_qcaaopt[idx])

# Mean Error

records.append([g, "Mean Error",

f"{m2:.3f} ± {s2:.3f}",

f"{m3:.3f} ± {s3:.3f}"])

# Error Variance

records.append([g, "Error Variance",

f"{v2:.6f}",

f"{v3:.6f}"])

# Max Single Error

records.append([g, "Max Single Error",

f"{M2:.4f}",

f"{M3:.4f}"])

table = pd.DataFrame(records, columns=[

"Data Type",

"Metric",

"QCAA (Non-Opt.)",

"QCAA (Optimized)"

])

out\_csv = LOG\_DIR / "table4\_qcaa\_only.csv"

table.to\_csv(out\_csv, index=False)

print("\n=== Table 4 (QCAA-only): Reconstruction Error Analysis Across Data Types ===")

print(table.to\_string(index=False))

print(f"\nSaved to: {out\_csv.resolve()}")

# Meta

with open(LOG\_DIR / "table4\_only\_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"Samples: {len(X\_norm)}\n")

f.write(f"Features: {', '.join(feature\_cols)}\n")

f.write(f"Data-type column: {dtype\_col or 'None (All Data)'}\n")

f.write(f"Output CSV: {out\_csv}\n")