# === Encoding Strategy Ablation Study (Fast, Fixed DROP sizing) — ibm\_brisbane ===

# Output dir: log20/

# Changes vs log18:

# • Robust DROP ablations for AMPLITUDE/HYBRID: use per-sample effective sizes

# (effective\_size, effective\_d\_ang) so shapes always match.

# • Still only THREE hardware jobs total (ANGLE / AMPLITUDE / HYBRID).

# Artifacts:

# - log20/eer\_overview.csv

# - log20/ablation\_drop.csv

# - log20/ablation\_alter.csv

# - log20/\*counts\_pub0\_\*.json

# - log20/run.log, log20/run.jsonl

# - log20/summary.json

import os, sys, json, uuid, logging, math

from pathlib import Path

from datetime import datetime

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import roc\_curve

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

from qiskit import QuantumCircuit, transpile

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

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

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

LOG\_DIR = Path("log20"); 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)

def log\_json(event, \*\*kw):

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

with open(LOG\_DIR / "run.jsonl", "a", encoding="utf-8") as f:

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

def log\_kv(msg, \*\*kw):

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

log\_json(msg, \*\*kw)

# ---------------- Credentials (updated) ----------------

IBM\_TOKEN = "isTxH69BGxixH7QohOX\_F8Zxm9fvMY4FP4ZET6F9xjTZ"

IBM\_INSTANCE = "crn:v1:bluemix:public:quantum-computing:us-east:a/34961d67783d401f880bc62b6543135b:a4d04a9f-4d50-445b-b363-7db4050ad8c2::"

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

DATA\_PATH = r"C:\Users\Sandip Dutta\OneDrive\Desktop\biometric\_detailed\_dataset.csv"

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

SHOTS = 1024

OPT\_LEVEL = 0 # faster transpile; measurement semantics unchanged

# Drift to create S2/S3

DRIFT\_S2 = (0.03, 1.03)

DRIFT\_S3 = (0.06, 1.06)

AMP\_Q\_MAX = 3 # up to 8 features via amplitude

np.random.seed(42)

log\_kv("config\_set", data\_path=DATA\_PATH, shots=SHOTS, opt\_level=OPT\_LEVEL, amp\_q\_max=AMP\_Q\_MAX)

# ---------------- Backend ----------------

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)

# ---------------- Data ----------------

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

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 (expected one of label/y/target/Label/Target).")

feature\_cols = [c for c in df.columns if c != label\_col and np.issubdtype(df[c].dtype, np.number)]

if not feature\_cols:

raise ValueError("No numeric feature columns besides label.")

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

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

session\_col = next((c for c in ["session","timestamp","time","date","Session","Timestamp"] if c in df.columns), None)

order = np.argsort(df[session\_col].values) if session\_col else np.arange(len(df))

X\_raw, y = X\_raw[order], y[order]

scaler = MinMaxScaler()

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

log\_kv("dataset\_loaded", n\_samples=len(X), n\_features=X.shape[1],

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

feature\_cols=",".join(feature\_cols))

# ---------------- Sessions ----------------

n = len(X)

s1\_end = int(0.6\*n); s2\_end = int(0.8\*n)

idx\_s1, idx\_s2, idx\_s3 = np.arange(0, s1\_end), np.arange(s1\_end, s2\_end), np.arange(s2\_end, n)

def driftify(Xpart, mean\_shift, scale): return np.clip(Xpart \* scale + mean\_shift, 0.0, 1.0)

X\_s1, y\_s1 = X[idx\_s1], y[idx\_s1]

X\_s2, y\_s2 = driftify(X[idx\_s2], \*DRIFT\_S2), y[idx\_s2]

X\_s3, y\_s3 = driftify(X[idx\_s3], \*DRIFT\_S3), y[idx\_s3]

labels\_by\_sess = {"s1": y\_s1, "s2": y\_s2, "s3": y\_s3}

# ---------------- Utils ----------------

def to\_bits\_any(k, width):

if isinstance(k, int): return format(k, f"0{width}b")

if isinstance(k, str):

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

if set(s) <= {"0","1"}: return s.zfill(width)[-width:]

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

except Exception: return s.zfill(width)[-width:]

if isinstance(k, tuple):

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

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

return "0"\*width

def extract\_quasi\_and\_snapshot(res, logical\_width, tag, shots):

quasi\_list = getattr(res, "quasi\_dists", None)

if quasi\_list is None:

try: results\_iter = list(res)

except TypeError: results\_iter = getattr(res, "results", [])

quasi\_list = []

for r in results\_iter:

qd = {}

got = None

for getter in (

lambda r: r.data.meas.get\_counts(),

lambda r: r.get\_counts(),

lambda r: r.data.counts,

lambda r: r.metadata.get("counts", None),

):

try:

got = getter(r)

if got: break

except Exception:

pass

got = dict(got or {})

total = max(int(sum(got.values())), 1)

for k,v in got.items(): qd[k] = v/total

quasi\_list.append(qd)

try:

snap = {to\_bits\_any(k, logical\_width): int(float(v)\*shots) for k,v in dict(quasi\_list[0]).items()}

with open(LOG\_DIR / f"counts\_pub0\_{tag}.json","w",encoding="utf-8") as f: json.dump(snap,f,ensure\_ascii=False,indent=2)

log\_kv("counts\_pub0\_saved", tag=tag, path=str((LOG\_DIR / f"counts\_pub0\_{tag}.json").resolve()))

except Exception:

pass

return quasi\_list

def extract\_P1\_from\_quasi\_list(quasi\_list, logical\_width):

N = len(quasi\_list)

P1 = np.zeros((N, logical\_width), dtype=float)

for i, qdist in enumerate(quasi\_list):

row = np.zeros(logical\_width)

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

bits = to\_bits\_any(k, logical\_width)[::-1]

for q in range(logical\_width):

if bits[q] == "1": row[q] += float(prob)

P1[i] = row

return P1

def compute\_eer(scores, labels):

fpr, tpr, thr = roc\_curve(labels, -scores)

fnr = 1 - tpr

idx = int(np.argmin(np.abs(fpr - fnr)))

return float((fpr[idx] + fnr[idx]) / 2.0)

# ---------------- Encoding builders ----------------

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

d = len(x\_vec); qc = QuantumCircuit(d, d)

for i in range(d):

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

qc.ry(2.0 \* np.arcsin(np.sqrt(xi)), i)

for i in range(d-1): qc.cz(i, i+1)

qc.measure(range(d), range(d))

return qc

def build\_amplitude\_circuit(x\_vec, q\_max=AMP\_Q\_MAX):

m = len(x\_vec)

if m <= 1:

q = 1; size = 1

else:

q = min(q\_max, int(math.floor(math.log2(m))))

q = max(q, 1)

size = min(2\*\*q, m)

sub = np.array(x\_vec[:size], dtype=float)

norm = np.linalg.norm(sub)

sub = (sub / norm) if norm != 0 else (np.ones\_like(sub)/math.sqrt(size))

amps = np.zeros(2\*\*q, dtype=complex); amps[:size] = sub.astype(complex)

qc = QuantumCircuit(q, q)

if q > 0: qc.initialize(amps.tolist(), list(range(q)))

qc.measure(range(q), range(q))

return qc, q, size

def build\_hybrid\_circuit(x\_vec, q\_max=AMP\_Q\_MAX):

m = len(x\_vec)

if m <= 1:

q = 1; size = 1

else:

q = min(q\_max, int(math.floor(math.log2(m))))

q = max(q, 1)

size = min(2\*\*q, m)

sub = np.array(x\_vec[:size], dtype=float)

norm = np.linalg.norm(sub)

sub = (sub / norm) if norm != 0 else (np.ones\_like(sub)/math.sqrt(size))

amps = np.zeros(2\*\*q, dtype=complex); amps[:size] = sub.astype(complex)

rest = x\_vec[size:]; d\_angle = len(rest)

qc = QuantumCircuit(q + d\_angle, q + d\_angle)

if q > 0: qc.initialize(amps.tolist(), list(range(q)))

for i in range(d\_angle):

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

qc.ry(2.0 \* np.arcsin(np.sqrt(xi)), q+i)

for i in range(d\_angle-1): qc.cz(q+i, q+i+1)

qc.measure(range(q + d\_angle), range(q + d\_angle))

return qc, q, size, d\_angle

# ---------------- Scoring proxies ----------------

def score\_angle\_from\_P1(P1, Xref):

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

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

def score\_amplitude\_effective(qdist, q, effective\_size, X\_sub):

Xt = np.array(X\_sub[:, :effective\_size], dtype=float)

norms = np.linalg.norm(Xt, axis=1, keepdims=True); norms[norms==0]=1.0

Xt = Xt / norms

probs = np.zeros(2\*\*q, dtype=float) if q>0 else np.array([1.0], dtype=float)

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

idx = int(to\_bits\_any(k, q), 2) if q>0 else 0

probs[idx] += float(prob)

dif = Xt[0] - probs[:effective\_size]

return float(np.mean(dif\*dif))

def score\_hybrid\_effective(qdist, q, size, d\_ang, X\_row):

"""

Compare quasi distribution against a possibly shorter feature vector (after DROP).

- effective\_size = min(size, m\_new)

- effective\_d\_ang = min(d\_ang, m\_new - effective\_size)

"""

m\_new = X\_row.shape[1]

effective\_size = min(size, m\_new)

effective\_d\_ang = max(0, min(d\_ang, m\_new - effective\_size))

# amplitude part

probs = np.zeros(2\*\*q, dtype=float) if q>0 else np.array([1.0], dtype=float)

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

bits = to\_bits\_any(k, q + d\_ang)

idx = int(bits[:q], 2) if q>0 else 0

probs[idx] += float(prob)

Xn = np.array(X\_row[0, :effective\_size], dtype=float)

norm = np.linalg.norm(Xn); Xn = (Xn/norm) if norm!=0 else (np.ones\_like(Xn)/math.sqrt(len(Xn))) if effective\_size>0 else Xn

amp\_err = float(np.mean((Xn - probs[:effective\_size])\*\*2)) if effective\_size>0 else 0.0

# angle part

ang\_err = 0.0

if effective\_d\_ang > 0:

P1\_ang = np.zeros(d\_ang, dtype=float)

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

bits = to\_bits\_any(k, q + d\_ang)[::-1] # LSB-first for angle indexing

for aq in range(d\_ang):

if bits[aq] == "1": P1\_ang[aq] += float(prob)

P1\_eff = P1\_ang[:effective\_d\_ang]

rest = X\_row[0, effective\_size:effective\_size+effective\_d\_ang]

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

ang\_err = float(np.mean((rest - x\_hat)\*\*2))

tot = max(effective\_size + effective\_d\_ang, 1)

return (effective\_size\*amp\_err + effective\_d\_ang\*ang\_err)/tot

# ---------------- Batching (one-time per encoding) ----------------

def batch\_angle(X1, X2, X3):

circuits=[]; idx\_map={}; start=0

for tag, Xref in [("s1",X1),("s2",X2),("s3",X3)]:

cs=[build\_angle\_circuit(x) for x in Xref]

circuits.extend(cs); end=start+len(cs); idx\_map[tag]=(start,end,Xref); start=end

logical\_width = len(circuits[0].clbits) if circuits else X1.shape[1]

return circuits, idx\_map, logical\_width

def batch\_amplitude(X1, X2, X3, q\_max=AMP\_Q\_MAX):

circuits=[]; idx\_map={}; start=0

for tag, Xref in [("s1",X1),("s2",X2),("s3",X3)]:

cs=[]; meta=[]

for x in Xref:

c, q, size = build\_amplitude\_circuit(x, q\_max=q\_max)

cs.append(c); meta.append((q,size))

circuits.extend(cs); end=start+len(cs); idx\_map[tag]=(start,end,Xref,meta); start=end

return circuits, idx\_map

def batch\_hybrid(X1, X2, X3, q\_max=AMP\_Q\_MAX):

circuits=[]; idx\_map={}; start=0

for tag, Xref in [("s1",X1),("s2",X2),("s3",X3)]:

cs=[]; meta=[]

for x in Xref:

c, q, size, d\_ang = build\_hybrid\_circuit(x, q\_max=q\_max)

cs.append(c); meta.append((q,size,d\_ang))

circuits.extend(cs); end=start+len(cs); idx\_map[tag]=(start,end,Xref,meta); start=end

return circuits, idx\_map

# ---------------- One-time hardware runs (3 jobs) ----------------

sampler = Sampler(mode=backend)

# ANGLE

angle\_circs, angle\_map, angle\_w = batch\_angle(X\_s1, X\_s2, X\_s3)

angle\_circs\_t = [transpile(c, backend=backend, optimization\_level=OPT\_LEVEL) for c in angle\_circs]

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

jid = job.job\_id(); log\_kv("job\_submitted", model="ENC\_ANGLE", n\_circuits=len(angle\_circs\_t), job\_id=jid)

res = job.result(); log\_kv("job\_completed", model="ENC\_ANGLE", job\_id=jid)

angle\_quasi = extract\_quasi\_and\_snapshot(res, angle\_w, "ENC\_ANGLE\_batch", SHOTS)

P1\_angle\_all = extract\_P1\_from\_quasi\_list(angle\_quasi, angle\_w)

# AMPLITUDE

amp\_circs, amp\_map = batch\_amplitude(X\_s1, X\_s2, X\_s3, q\_max=AMP\_Q\_MAX)

amp\_circs\_t = [transpile(c, backend=backend, optimization\_level=OPT\_LEVEL) for c in amp\_circs]

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

jid = job.job\_id(); log\_kv("job\_submitted", model="ENC\_AMPLITUDE", n\_circuits=len(amp\_circs\_t), job\_id=jid)

res = job.result(); log\_kv("job\_completed", model="ENC\_AMPLITUDE", job\_id=jid)

amp\_quasi = extract\_quasi\_and\_snapshot(res, max(1, AMP\_Q\_MAX), "ENC\_AMPLITUDE\_batch", SHOTS)

# HYBRID

hyb\_circs, hyb\_map = batch\_hybrid(X\_s1, X\_s2, X\_s3, q\_max=AMP\_Q\_MAX)

hyb\_circs\_t = [transpile(c, backend=backend, optimization\_level=OPT\_LEVEL) for c in hyb\_circs]

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

jid = job.job\_id(); log\_kv("job\_submitted", model="ENC\_HYBRID", n\_circuits=len(hyb\_circs\_t), job\_id=jid)

res = job.result(); log\_kv("job\_completed", model="ENC\_HYBRID", job\_id=jid)

# logical width here is only for snapshot pretty-print; not used for scoring

hyb\_quasi = extract\_quasi\_and\_snapshot(res, max(1, AMP\_Q\_MAX + X.shape[1]), "ENC\_HYBRID\_batch", SHOTS)

# ---------------- Base EERs ----------------

def slice\_scores\_angle(P1\_all, idx\_map):

out = {}

for tag,(s,e,Xref) in idx\_map.items():

out[tag] = score\_angle\_from\_P1(P1\_all[s:e], Xref)

return out

def slice\_scores\_amp(quasi\_all, idx\_map):

out = {}

for tag,(s,e,Xref,meta) in idx\_map.items():

qd = quasi\_all[s:e]

err = np.zeros(len(qd), dtype=float)

for i,(q,size) in enumerate(meta):

# effective size equals min(size, current width)

eff\_size = min(size, Xref.shape[1])

err[i] = score\_amplitude\_effective(qd[i], q, eff\_size, Xref[i:i+1, :eff\_size])

out[tag] = err

return out

def slice\_scores\_hyb(quasi\_all, idx\_map):

out = {}

for tag,(s,e,Xref,meta) in idx\_map.items():

qd = quasi\_all[s:e]

err = np.zeros(len(qd), dtype=float)

for i,(q,size,d\_ang) in enumerate(meta):

err[i] = score\_hybrid\_effective(qd[i], q, size, d\_ang, Xref[i:i+1])

out[tag] = err

return out

scores\_angle = slice\_scores\_angle(P1\_angle\_all, angle\_map)

scores\_amp = slice\_scores\_amp(amp\_quasi, amp\_map)

scores\_hyb = slice\_scores\_hyb(hyb\_quasi, hyb\_map)

def eer\_for\_scores(scores\_dict, labels\_dict):

s1\_scores = scores\_dict["s1"]; s1\_labels = labels\_dict["s1"]

all\_scores = np.concatenate([scores\_dict["s1"], scores\_dict["s2"], scores\_dict["s3"]], axis=0)

all\_labels = np.concatenate([labels\_dict["s1"], labels\_dict["s2"], labels\_dict["s3"]], axis=0)

return compute\_eer(s1\_scores, s1\_labels), compute\_eer(all\_scores, all\_labels)

records = []

e1, ea = eer\_for\_scores(scores\_angle, labels\_by\_sess); records.append({"encoding":"ANGLE","EER\_S1":e1,"EER\_ALL":ea}); log\_kv("eer\_angle", EER\_S1=f"{e1:.4f}", EER\_ALL=f"{ea:.4f}")

e1, ea = eer\_for\_scores(scores\_amp, labels\_by\_sess); records.append({"encoding":"AMPLITUDE","EER\_S1":e1,"EER\_ALL":ea}); log\_kv("eer\_amplitude", EER\_S1=f"{e1:.4f}", EER\_ALL=f"{ea:.4f}")

e1, ea = eer\_for\_scores(scores\_hyb, labels\_by\_sess); records.append({"encoding":"HYBRID","EER\_S1":e1,"EER\_ALL":ea}); log\_kv("eer\_hybrid", EER\_S1=f"{e1:.4f}", EER\_ALL=f"{ea:.4f}")

pd.DataFrame(records).to\_csv(LOG\_DIR / "eer\_overview.csv", index=False)

# ---------------- Per-feature Ablations (OFFLINE; no extra jobs) ----------------

F = len(feature\_cols)

drop\_rows = []

alter\_rows = []

# ANGLE: DROP (remove column) and ALTER (invert column) using P1 already measured

for i in range(F):

# DROP

s1 = score\_angle\_from\_P1(np.delete(P1\_angle\_all[angle\_map['s1'][0]:angle\_map['s1'][1]], i, axis=1),

np.delete(angle\_map['s1'][2], i, axis=1))

s2 = score\_angle\_from\_P1(np.delete(P1\_angle\_all[angle\_map['s2'][0]:angle\_map['s2'][1]], i, axis=1),

np.delete(angle\_map['s2'][2], i, axis=1))

s3 = score\_angle\_from\_P1(np.delete(P1\_angle\_all[angle\_map['s3'][0]:angle\_map['s3'][1]], i, axis=1),

np.delete(angle\_map['s3'][2], i, axis=1))

e1, ea = eer\_for\_scores({"s1":s1,"s2":s2,"s3":s3}, labels\_by\_sess)

drop\_rows.append({"encoding":"ANGLE","feature\_dropped":feature\_cols[i],"EER\_S1":e1,"EER\_ALL":ea})

# ALTER (invert 1-x on that column)

def invert\_col(Xm, idx):

Xc = Xm.copy(); Xc[:, idx] = 1.0 - Xc[:, idx]; return Xc

s1 = score\_angle\_from\_P1(P1\_angle\_all[angle\_map['s1'][0]:angle\_map['s1'][1]], invert\_col(angle\_map['s1'][2], i))

s2 = score\_angle\_from\_P1(P1\_angle\_all[angle\_map['s2'][0]:angle\_map['s2'][1]], invert\_col(angle\_map['s2'][2], i))

s3 = score\_angle\_from\_P1(P1\_angle\_all[angle\_map['s3'][0]:angle\_map['s3'][1]], invert\_col(angle\_map['s3'][2], i))

e1, ea = eer\_for\_scores({"s1":s1,"s2":s2,"s3":s3}, labels\_by\_sess)

alter\_rows.append({"encoding":"ANGLE","feature\_altered":feature\_cols[i],"EER\_S1":e1,"EER\_ALL":ea})

# AMPLITUDE & HYBRID: DROP (effective sizes) and ALTER (invert column) by rescoring against same quasi

def slice\_quasi(quasi, s, e): return quasi[s:e]

# DROP loops

for i in range(F):

def dropX(Xm): return np.delete(Xm, i, axis=1)

# AMPLITUDE

for tag in ["s1","s2","s3"]:

s,e,Xref,meta = amp\_map[tag]

qd = slice\_quasi(amp\_quasi, s, e)

Xm = dropX(Xref)

err = np.zeros(len(qd), dtype=float)

for k,(q,size) in enumerate(meta):

eff\_size = min(size, Xm.shape[1])

err[k] = score\_amplitude\_effective(qd[k], q, eff\_size, Xm[k:k+1, :eff\_size])

if tag=="s1": err\_s1 = err

elif tag=="s2": err\_s2 = err

else: err\_s3 = err

e1, ea = eer\_for\_scores({"s1":err\_s1,"s2":err\_s2,"s3":err\_s3}, labels\_by\_sess)

drop\_rows.append({"encoding":"AMPLITUDE","feature\_dropped":feature\_cols[i],"EER\_S1":e1,"EER\_ALL":ea})

# HYBRID

for tag in ["s1","s2","s3"]:

s,e,Xref,meta = hyb\_map[tag]

qd = slice\_quasi(hyb\_quasi, s, e)

Xm = dropX(Xref)

err = np.zeros(len(qd), dtype=float)

for k,(q,size,d\_ang) in enumerate(meta):

err[k] = score\_hybrid\_effective(qd[k], q, size, d\_ang, Xm[k:k+1])

if tag=="s1": err\_s1 = err

elif tag=="s2": err\_s2 = err

else: err\_s3 = err

e1, ea = eer\_for\_scores({"s1":err\_s1,"s2":err\_s2,"s3":err\_s3}, labels\_by\_sess)

drop\_rows.append({"encoding":"HYBRID","feature\_dropped":feature\_cols[i],"EER\_S1":e1,"EER\_ALL":ea})

# ALTER loops (invert one column in X)

for i in range(F):

def invertX(Xm):

Xc = Xm.copy(); Xc[:, i] = 1.0 - Xc[:, i]; return Xc

# AMPLITUDE

for tag in ["s1","s2","s3"]:

s,e,Xref,meta = amp\_map[tag]

qd = slice\_quasi(amp\_quasi, s, e)

Xa = invertX(Xref)

err = np.zeros(len(qd), dtype=float)

for k,(q,size) in enumerate(meta):

eff\_size = min(size, Xa.shape[1])

err[k] = score\_amplitude\_effective(qd[k], q, eff\_size, Xa[k:k+1, :eff\_size])

if tag=="s1": err\_s1 = err

elif tag=="s2": err\_s2 = err

else: err\_s3 = err

e1, ea = eer\_for\_scores({"s1":err\_s1,"s2":err\_s2,"s3":err\_s3}, labels\_by\_sess)

alter\_rows.append({"encoding":"AMPLITUDE","feature\_altered":feature\_cols[i],"EER\_S1":e1,"EER\_ALL":ea})

# HYBRID

for tag in ["s1","s2","s3"]:

s,e,Xref,meta = hyb\_map[tag]

qd = slice\_quasi(hyb\_quasi, s, e)

Xa = invertX(Xref)

err = np.zeros(len(qd), dtype=float)

for k,(q,size,d\_ang) in enumerate(meta):

err[k] = score\_hybrid\_effective(qd[k], q, size, d\_ang, Xa[k:k+1])

if tag=="s1": err\_s1 = err

elif tag=="s2": err\_s2 = err

else: err\_s3 = err

e1, ea = eer\_for\_scores({"s1":err\_s1,"s2":err\_s2,"s3":err\_s3}, labels\_by\_sess)

alter\_rows.append({"encoding":"HYBRID","feature\_altered":feature\_cols[i],"EER\_S1":e1,"EER\_ALL":ea})

# ---------------- Save CSVs ----------------

pd.DataFrame(records).to\_csv(LOG\_DIR / "eer\_overview.csv", index=False)

pd.DataFrame(drop\_rows).to\_csv(LOG\_DIR / "ablation\_drop.csv", index=False)

pd.DataFrame(alter\_rows).to\_csv(LOG\_DIR / "ablation\_alter.csv", index=False)

# ---------------- Summary JSON ----------------

def \_to\_builtin(x):

if isinstance(x, dict): return {k:\_to\_builtin(v) for k,v in x.items()}

if isinstance(x, list): return [\_to\_builtin(v) for v in x]

if isinstance(x, tuple): return tuple(\_to\_builtin(v) for v in x)

if isinstance(x, np.ndarray): return x.tolist()

if isinstance(x, (np.integer,)): return int(x)

if isinstance(x, (np.floating,)): return float(x)

return x

summary = {

"RUN\_ID": RUN\_ID,

"backend": backend.name,

"shots": SHOTS,

"opt\_level": OPT\_LEVEL,

"dataset": DATA\_PATH,

"n\_samples": int(n),

"n\_session1": int(len(X\_s1)),

"n\_session2": int(len(X\_s2)),

"n\_session3": int(len(X\_s3)),

"features": feature\_cols,

"amp\_q\_max": AMP\_Q\_MAX,

"eer\_overview\_path": str((LOG\_DIR/"eer\_overview.csv").resolve()),

"ablation\_drop\_path": str((LOG\_DIR/"ablation\_drop.csv").resolve()),

"ablation\_alter\_path": str((LOG\_DIR/"ablation\_alter.csv").resolve())

}

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

json.dump(\_to\_builtin(summary), f, indent=2)

print("\n✅ Fast ablation study (fixed sizes) completed. Results saved to:", str(LOG\_DIR.resolve()))

print(pd.read\_csv(LOG\_DIR / "eer\_overview.csv"))