SUPPORTING INFORMATION

Evaluation Dependence in Static ASL Fingerspelling Classification: A Forensic Analysis of Sign Language MNIST

Bernard Freund

This document provides additional methods, supplementary audit summaries, numeric tables, mapping checks, and source listings referenced by the accepted manuscript.

Interpretive guardrail

Exact-pHash groups in the released Sign Language MNIST CSV files are similarity groups, not ground-truth source-photo groups. Calibration against known photographs found both false links and missed true descendant links. The only verified source-disjoint classifier experiment used a separate controlled collection built from 1,680 known source photographs and five synthetic derivatives per photograph.

Supplement contents

S1. Data sources and reproducibility; S2. exact-pHash group audit; S3. lineage calibration; S4. controlled source-disjoint classification; S5. released-CSV and bootstrap analyses; S6. distance and source-reconstruction sensitivity checks; S7. external label audit and confusion matrices; S8. residual-network mechanism check; S9. software, seeds, and file provenance; Appendix A. source code listings.

S1. Data sources and reproducibility

The official Sign Language MNIST training and test CSV files were kept separate. The public source archive supplied 1,680 labeled photographs and bounding boxes. A separate five-subject fingerspelling collection was used only for external evaluation. Raw third-party images are not redistributed in this supplement.

Input	Public source	Role
Sign Language MNIST	https://www.kaggle.com/datasets/datamunge/sign-language-mnist	Released benchmark: 27,455 train and 7,172 test rows
Source photographs	https://github.com/mon95/Sign-Language-and-Static-gesture-recognition-using-sklearn	Known-source calibration and controlled experiment
External sequences	https://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset	Zero-shot five-subject external evaluation

Table S1. *Public datasets and their roles. Access and licensing remain controlled by the original providers.*

Dataset integrity identifiers

File	SHA-256
sign_mnist_train.csv	4c2897f19fab2b0ae2a7e4fa82e969043315d9f3a1a9cc0948b576bf1189a7e5
sign_mnist_test.csv	0e9d67bae23e67f40728e0b63bf15ad4bd5175947b8a9fac5dd9f17ce133c47b

Table S2. *Checksums identify the exact CSV files analyzed.*

Environment

```
numpy==2.1.3
pandas==3.0.3
scikit-learn==1.7.1
scipy==1.18.0
tensorflow==2.19.0
matplotlib==3.10.3
Pillow==12.3.0
joblib==1.5.3
threadpoolctl==3.6.0
```

Reproduction layout

Extract the retained author archive to a fresh project root. Place sign_mnist_train.csv and sign_mnist_test.csv in work/data. Place the source repository's Dataset directory in work/external/sign-source-repo/Dataset, and place the five-subject external dataset in work/external/extracted/dataset5. Copy the legacy modules from work/novel_supporting/code into work, or add that directory to the module search path. Use Python 3.12 and set PYTHONPATH to work:work/revision_2026_09_20 before running the revision scripts.

Run the scripts in this order: calibrate_lineage.py; audit_group_structure.py; audit_external_labels.py; audit_source_confidence.py; diagnose_residual_fold.py; diagnose_residual_fold_seed.py; diagnose_residual_bn_recalibration.py; known_source_experiment.py; make_revision_figures.py. TensorFlow refits are slower than the deterministic audits. The public dataset page describes approximately 1,704 original photographs, while the available source archive contains 1,680. The missing 24 records and the absence of a public row-to-source manifest prevent a verified source-disjoint evaluation of the released CSV files.

S2. Released-CSV exact-pHash group audit

No two released pixel rows were byte-identical. Exact same-letter pHashes nevertheless formed many repeated similarity groups. These counts describe image similarity only and do not establish source-photo identity.

Partition	Images	Groups	Singleton groups	Images in multi-image groups	Median size	Maximum size
All released images	34,627	3,656	1,030	33,597	5.0	61
Official train	27,455	2,901	819	26,636	6.0	42
Official test	7,172	766	214	6,958	5.0	39

Table S3. *Exact same-letter pHash group summary by released partition.*

Letter	Images	Multi-image-group images	Fraction	Groups
A	1457	1410	96.77%	151
B	1442	1382	95.84%	161

Letter	Images	Multi-image-group images	Fraction	Groups
C	1454	1425	98.01%	142
D	1441	1405	97.50%	139
E	1455	1413	97.11%	144
F	1451	1414	97.45%	135
G	1438	1399	97.29%	146
H	1449	1401	96.69%	168
I	1450	1399	96.48%	164
K	1445	1393	96.40%	167
L	1450	1415	97.59%	142
M	1449	1414	97.58%	145
N	1442	1408	97.64%	146
O	1442	1404	97.36%	145
P	1435	1392	97.00%	143
Q	1443	1371	95.01%	197
R	1438	1397	97.15%	139
S	1445	1400	96.89%	155
T	1434	1393	97.14%	147
U	1427	1392	97.55%	152
V	1428	1391	97.41%	144
W	1431	1386	96.86%	179
X	1431	1392	97.27%	149
Y	1450	1401	96.62%	156

Table S4. *Complete per-letter exact-pHash group counts.*

Inferred user	Images	Fraction in multi-image groups
user_10	4968	96.62%
user_3	4961	95.97%
user_4	4951	96.32%
user_5	4919	97.93%
user_6	4929	96.79%
user_7	4949	97.49%
user_9	4950	98.06%

Table S5. *Counts by inferred user. These labels came from a fallible nearest-source reconstruction and are not creator-supplied signer IDs.*

S3. Known-source pHash lineage calibration

Calibration used 1,680 source photographs and 8,400 controlled derivatives, for 10,080 images. It included 25,200 same-source pairs and 2,086,560 different-source within-letter pairs. A rule required linked-pair precision of at least 99% to be accepted as a source-lineage rule; none qualified.

pHash rule	TP	FP	TN	FN	Sensitivity	Specificity	Precision	False-merger groups	Split sources
Exact	2257	133	2086427	22943	8.96%	99.994%	94.44%	29 / 8,249	1,680 / 1,680
<=2 bits	6220	951	2085609	18980	24.68%	99.954%	86.74%	153 / 6,029	1,678 / 1,680
<=4 bits	9971	3396	2083164	15229	39.57%	99.837%	74.59%	306 / 4,415	1,667 / 1,680
<=6 bits	12338	7907	2078653	12862	48.96%	99.621%	60.94%	424 / 3,449	1,634 / 1,680

Table S6. *Complete aggregate pairwise and connected-component calibration counts.*

Known source-photo identities allowed direct counts of false links and missed descendants. Exact pHash is therefore a conservative similarity grouping for stress tests, not a source-lineage label.

S4. Controlled random versus source-disjoint classification

Prespecified fixed models. One stratified 80/20 image-level split and five outer source-disjoint folds. Three CNN seeds per outer test set. CNN checkpoint selected on development-only inner validation. No test-row observations used for preprocessing, training, or checkpoint selection.

Protocol	Fold	Fit	Validation	Test	Source overlap	1-NN macro F1	Raw SVM macro F1	HOG-SVM macro F1	CNN ensemble macro F1
image random	0	6451	1613	2016	1228	90.08%	71.47%	95.03%	81.15%
source disjoint	0	6450	1614	2016	0	80.76%	49.60%	89.47%	79.27%
source disjoint	1	6450	1614	2016	0	80.51%	48.78%	91.59%	82.08%
source disjoint	2	6450	1614	2016	0	81.62%	48.29%	89.18%	79.22%
source disjoint	3	6450	1614	2016	0	80.62%	52.26%	91.54%	80.73%
source disjoint	4	6444	1620	2016	0	78.90%	47.85%	90.03%	78.31%

Table S7. *All outer-partition macro-F1 values for the controlled known-source experiment.*

Model	Random holdout macro F1	Source-disjoint mean	Sample SD	Difference
1-nearest-neighbor pixels	90.08%	80.48%	0.99%	9.60 points
Raw-pixel SVM	71.47%	49.36%	1.75%	22.11 points
HOG-SVM	95.03%	90.36%	1.14%	4.66 points
Standard CNN ensemble	81.15%	79.92%	1.48%	1.23 points

Table S8. *Controlled experiment summary recalculated from the saved fold outputs.*

The synthetic derivatives deliberately tested known lineage but did not recreate the undocumented Sign Language MNIST transformation process. Accordingly, these results do not provide a source-disjoint estimate for the released CSV files.

S5. Released-CSV model comparisons and uncertainty analyses

The released-CSV analyses compared ordinary random validation, exact-pHash-separated validation, and the official test file. Exact-pHash separation is a similarity-group stress test and is not described as a source-disjoint partition.

Model	Random validation	Similarity-grouped validation	Official test
1-nearest-neighbor pixels	99.96%	83.10%	79.97%
Raw-pixel SVM	100.00%	87.78%	78.41%
HOG-SVM	100.00%	95.11%	93.65%
Standard CNN, no augmentation	100.00% +/- 0.00	91.90% +/- 1.25	91.77% +/- 1.83
Standard CNN, training augmentation	99.27% +/- 0.24	96.93% +/- 1.85	97.85% +/- 0.88

Table S9. *Macro F1 for released-CSV protocols. CNN entries are mean +/- sample SD across seeds 25, 37, 49, 61, and 73.*

Model	Pooled out-of-fold accuracy	Pooled out-of-fold macro F1	95% group-bootstrap CI for macro F1
1-nearest-neighbor pixels	86.50%	86.48%	84.94% to 87.88%
Nested raw-pixel SVM	79.57%	79.77%	77.60% to 81.65%
Nested HOG-SVM	97.20%	97.17%	96.32% to 97.96%
Standard CNN ensemble	98.04%	98.06%	97.44% to 98.65%
Compact residual CNN ensemble	98.69%	98.68%	98.22% to 99.10%

Table S10. *Nested five-fold exact-pHash-group evaluation. Bootstrap resampling used 2,000 replicates over 2,901 similarity groups (seed 20260714).*

Paired macro-F1 difference	Point estimate	95% group-bootstrap CI
Nested HOG-SVM minus 1-nearest-neighbor	10.70 points	9.29 to 12.21 points
Standard CNN minus nested HOG-SVM	0.89 points	-0.003 to 1.86 points
Residual CNN minus standard CNN	0.62 points	-0.12 to 1.35 points
Residual CNN minus nested HOG-SVM	1.51 points	0.60 to 2.47 points

Table S11. *Paired macro-F1 differences from the same group-bootstrap replicates.*

Protocol	Condition	Seed	Validation macro F1	Official-test macro F1
random	unaugmented	25	100.00%	94.54%
random	unaugmented	37	100.00%	89.40%
random	unaugmented	49	100.00%	91.63%
random	unaugmented	61	100.00%	91.45%
random	unaugmented	73	100.00%	91.82%
random	augmented	25	99.46%	98.67%
random	augmented	37	99.01%	96.82%
random	augmented	49	99.56%	98.85%
random	augmented	61	99.08%	97.32%
random	augmented	73	99.26%	97.60%
similarity-grouped	unaugmented	25	92.90%	89.06%
similarity-grouped	unaugmented	37	90.18%	86.94%
similarity-grouped	unaugmented	49	92.36%	84.18%
similarity-grouped	unaugmented	61	93.05%	89.46%
similarity-grouped	unaugmented	73	91.01%	83.40%
similarity-grouped	augmented	25	97.06%	97.17%
similarity-grouped	augmented	37	94.15%	93.99%
similarity-grouped	augmented	49	96.60%	96.23%
similarity-grouped	augmented	61	97.56%	96.90%
similarity-grouped	augmented	73	99.27%	98.56%

Table S12. *All 20 standard-CNN repeated-run macro-F1 results.*

S6. Image-distance and source-reconstruction sensitivity checks

Representation and partition	n	Median nearest-training distance	Interquartile range
PCA: random validation	5,491	0.338	0.136 to 0.556
PCA: official test	7,172	2.769	1.850 to 3.609
Full pixels: random validation sample	1,000	0.474	0.222 to 0.804
Full pixels: official-test sample	1,000	2.454	1.884 to 3.158

Table S13. *Nearest-training-image distance audit. The PCA test-to-validation median ratio was 8.19.*

Each query is forced to its nearest same-letter source among 1,680 candidates; no rejection threshold existed in the submitted analysis.

Score(best candidate from another inferred user) minus score(best candidate overall). Because the best candidate is globally minimal, the margin is nonnegative by construction; positivity is not evidence of correctness.

Controlled percentile	Margin threshold	Controlled coverage	Source top-1 accuracy	User accuracy	Released fraction above threshold
0	0.002	100.00%	77.63%	90.82%	100.00%
10	2.179	90.00%	84.70%	96.34%	100.00%
25	6.819	75.00%	92.60%	99.75%	99.41%
50	12.099	50.00%	96.10%	99.98%	80.92%
75	16.311	25.00%	97.43%	100.00%	47.02%

Table S14. *Nearest-source margin sensitivity. High margins enrich correct controlled matches but cannot certify released-image identities.*

S7. External label audit and confusion matrices

The external audit rebuilt labels for 2,400 frames from five subjects and 24 static letters, yielding 120 subject-letter units. The saved-label order assertion passed. Majority-vote ties used the lowest contiguous class index, matching numpy.argmax.

Original integer	Letter	Model index	External folder
0	A	0	a
1	B	1	b
2	C	2	c
3	D	3	d
4	E	4	e
5	F	5	f
6	G	6	g
7	H	7	h
8	I	8	i
10	K	9	k
11	L	10	l
12	M	11	m
13	N	12	n
14	O	13	o
15	P	14	p
16	Q	15	q
17	R	16	r
18	S	17	s
19	T	18	t
20	U	19	u
21	V	20	v
22	W	21	w
23	X	22	x
24	Y	23	y

Table S15. *Complete label mapping used for internal and external predictions.*

Model	Correct units	Total units	Accuracy
Original CNN ensemble	9	120	7.50%
Nested standard CNN ensemble	9	120	7.50%
Nested residual CNN ensemble	5	120	4.17%
HOG-SVM	6	120	5.00%

Table S16. *Depth-guided external zero-shot unit-level results verified by the label audit.*

Held-out subject	HOG-SVM frame accuracy	HOG-SVM subject-letter accuracy	CNN mean subject-letter accuracy
A	25.00%	50.00%	38.89%
B	28.12%	58.33%	38.89%
C	28.96%	70.83%	52.78%
D	18.75%	33.33%	11.11%
E	29.38%	54.17%	8.33%
Mean	26.04%	53.33% +/- 13.63	30.00% +/- 19.04

Table S17. *Leave-one-subject-out positive control on depth-guided crops. CNN means combine seeds 25, 37, and 49 within each held-out subject.*

True	A	B	C	D	E	F	G	H	I	K	L	M
A	0	0	1	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	2	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	1	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	1	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0
V	0	0	0	0	0	1	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0

Table S18a. *Numeric 24 x 24 subject-letter confusion matrix, predicted columns A through M.*

True	N	O	P	Q	R	S	T	U	V	W	X	Y
A	0	0	4	0	0	0	0	0	0	0	0	0
B	0	0	4	1	0	0	0	0	0	0	0	0
C	0	1	0	2	0	0	0	0	0	0	0	0
D	0	0	5	0	0	0	0	0	0	0	0	0
E	0	0	4	0	0	0	0	0	0	0	0	0
F	0	0	4	0	0	0	0	0	0	0	0	1
G	0	0	4	0	0	0	0	0	0	0	0	0
H	0	0	5	0	0	0	0	0	0	0	0	0
I	0	0	4	1	0	0	0	0	0	0	0	0
K	0	0	5	0	0	0	0	0	0	0	0	0
L	0	0	5	0	0	0	0	0	0	0	0	0
M	0	0	5	0	0	0	0	0	0	0	0	0
N	0	0	3	2	0	0	0	0	0	0	0	0
O	0	0	2	3	0	0	0	0	0	0	0	0
P	0	0	5	0	0	0	0	0	0	0	0	0
Q	0	0	3	1	0	0	0	0	0	0	1	0
R	0	0	4	0	0	0	0	0	0	0	1	0
S	0	0	4	1	0	0	0	0	0	0	0	0
T	0	0	4	1	0	0	0	0	0	0	0	0
U	0	0	2	0	0	0	0	0	0	0	3	0
V	0	0	3	0	0	0	0	0	0	0	1	0
W	0	0	4	0	0	0	0	0	0	0	1	0
X	0	0	3	2	0	0	0	0	0	0	0	0
Y	0	0	5	0	0	0	0	0	0	0	0	0

Table S18b. *Numeric 24 x 24 subject-letter confusion matrix, predicted columns N through Y.*

S8. Residual-network normalization-state mechanism check

Post hoc BatchNormalization mechanism check; not used for model selection or primary performance estimation.

Field	Value
Fold	3
Seed	37
Selected epoch	11
BatchNormalization layers	11
Recalibration passes	1
Recalibration batch size	128
Trainable weights unchanged	True
Final training-batch accuracy	99.80%

Table S19. *Configuration and invariance checks for the post hoc normalization-statistics recalibration.*

Stage	Partition	Accuracy	Macro precision	Macro recall	Macro F1	Predicted classes	Zero-recall classes
Before	outer_train	33.81%	44.19%	34.70%	25.70%	15	9
Before	outer_test	35.60%	40.24%	32.63%	24.32%	14	10
After	outer_train	94.69%	95.94%	94.59%	94.30%	24	0
After	outer_test	94.10%	95.08%	94.46%	94.20%	24	0

Table S20. *Performance before and after the no-gradient normalization-statistics pass.*

S9. Software, seeds, and artifact provenance

The five-seed CNN analysis used seeds 25, 37, 49, 61, and 73. The nested and controlled three-seed CNN ensembles used seeds 25, 37, and 49. Controlled known-source environment: NumPy 2.1.3, scikit-learn 1.7.1, and TensorFlow 2.19.0. Training augmentation was restricted to training batches; scalers and PCA transforms were fitted only on training observations; CNN checkpoint selection used inner development data.

Artifact category	Files in author archive	Examples
Python source	37	audit, training, evaluation, and figure scripts
JSON results/configuration	48	saved metrics, audit summaries, and experiment settings
CSV tables/manifests	5	confusion matrices, mappings, and controlled manifests
NumPy arrays and archives	98	predictions, probabilities, labels, mappings, and fold assignments
Figures	6	revision figures and diagnostic plots
Documentation/environment	2	readme and pinned package versions

Table S21. *Archive inventory by artifact type (196 files total).*

Core script	Bytes	SHA-256
calibrate_lineage.py	5,870	486620e4da5f5acae614ea5a903c919d228db536cdc9d453ebd8e34a0f16416f
audit_group_structure.py	3,497	bae8bcd5740c7a1a55866eb9ff3e5dce74c272b98d73f8bd233df92858208fce
audit_external_labels.py	5,571	75cd63b349238b27c1e5e23c767be83ba8f1e4122ccf7c7439fdc288b6eca7a5
audit_source_confidence.py	2,427	a4c04d3c9bbbc0804e8b51d7f5fc0cadd44d03a6f3457a31a5974406fdd24edb
diagnose_residual_fold.py	3,158	6ddce13cd5112fb5f0f2bfa15a44b53c9e25085166bd3d1996988cec5b068673
diagnose_residual_fold_seed.py	6,231	a2f20de39090e1359443c75f9b72bc6258827033040910323 3bcba38928b842b
diagnose_residual_bn_recalibration.py	3,868	483f473a1874a08f8222e517fa0af900306d13e47f09eb77abbc5943bc9975b8
known_source_experiment.py	6,305	e6d9cf5b170491a63812509231a3ebc6625b9b5848553893e884aa5cac1ef830
make_revision_figures.py	2,825	22610c87a2418d2f50f61ea797403060baf68fe5583deb9696a17b66c648a838

Table S22. *Integrity identifiers for the core source listings reproduced in Appendix A.*

The author retains the original machine-readable predictions, probability arrays, NumPy metadata, JSON outputs, and full source archive used for metric recalculation. These materials are available from the author upon reasonable request, subject to third-party licensing. The PDF prints the core revision scripts below and reports the scientific outputs needed to interpret the article. Third-party raw images are not redistributed and must be obtained from the providers listed in Table S1.

Appendix A. Core source code listings

The following listings are the core audit and controlled-experiment scripts supplied during revision. Line wrapping is typographic only; the archived source files are retained byte-for-byte by the author.

A1. calibrate_lineage.py

SHA-256: 486620e4da5f5acae614ea5a903c919d228db536cdc9d453ebd8e34a0f16416f

```
"""Calibrate pHash links on source images with known synthetic descendants.

This is a diagnostic of the grouping rule, not a claim that the released
Sign Language MNIST derivatives have known source identities.
"""
from __future__ import annotations

import json
from pathlib import Path

import numpy as np
from dataset_forensics import DSU, phash64
from reconstruct_source_groups import LETTERS, load_sources
from validate_source_reconstruction import derivative

ROOT = Path(__file__).resolve().parents[2]
OUT = ROOT / "work" / "revision_2026_09_20"
OUT.mkdir(exist_ok=True)
SEED = 25
THRESHOLDS = (0, 2, 4, 6)


def make_controlled_set():
    records, source_images = load_sources()
    rng = np.random.default_rng(SEED)
    images, source_id, letter, transform = [], [], [], []
    for i, image in enumerate(source_images):
        images.append(image)
        source_id.append(i)
        letter.append(LETTERS.index(records[i]["letter"]))
        transform.append("original")
        for mode, name in enumerate(("rotation", "photometric", "translation", "resampling", "noise")):
            images.append(derivative(image, rng, mode))
            source_id.append(i)
            letter.append(letter[-1])
            transform.append(name)
    return np.asarray(images), np.asarray(source_id), np.asarray(letter), np.asarray(transform), records


def calibrate(images, source_id, letter, records):
    hashes = phash64(images)
    metrics = {t: {"tp": 0, "fp": 0, "tn": 0, "fn": 0} for t in THRESHOLDS}
    source_users = np.asarray([record["user"] for record in records])
    false_link_same_user = {t: 0 for t in THRESHOLDS}
    dsus = {t: DSU(len(images)) for t in THRESHOLDS}
    by_class = {}
    for cls in np.unique(letter):
        idx = np.flatnonzero(letter == cls)
        class_counts = {t: {"tp": 0, "fp": 0, "tn": 0, "fn": 0} for t in THRESHOLDS}
        for a_pos in range(len(idx) - 1):
            a = int(idx[a_pos])
            b = idx[a_pos + 1 :]
            d = np.fromiter((int(v).bit_count() for v in np.bitwise_xor(hashes[b], hashes[a])),
                            dtype=np.uint8, count=len(b))
            same = source_id[b] == source_id[a]
            for t in THRESHOLDS:
                linked = d <= t
                cell = class_counts[t]
                cell["tp"] += int(np.sum(linked & same))
                cell["fp"] += int(np.sum(linked & ~same))
                cell["tn"] += int(np.sum(~linked & ~same))
                cell["fn"] += int(np.sum(~linked & same))
                false_link_same_user[t] += int(np.sum(linked & ~same &
                                                      (source_users[source_id[b]] == source_users[source_id[a]])))
                for other in b[linked]:
                    dsus[t].union(a, int(other))
        by_class[LETTERS[int(cls)]] = {str(t): class_counts[t] for t in THRESHOLDS}
        for t in THRESHOLDS:
            for key in metrics[t]:
                metrics[t][key] += class_counts[t][key]
    results = {}
```

```
    for t in THRESHOLDS:
        cell = metrics[t]
        roots = np.array([dsus[t].find(i) for i in range(len(images))])
        clusters = {}
        for row, root in enumerate(roots):
            clusters.setdefault(int(root), set()).add(int(source_id[row]))
        false_merge_clusters = sum(len(ids) > 1 for ids in clusters.values())
        sources_split = 0
        for source in np.unique(source_id):
            sources_split += len(set(roots[source_id == source])) > 1
        tp, fp, tn, fn = (cell[k] for k in ("tp", "fp", "tn", "fn"))
        results[str(t)] = {
            "pairwise": {**cell,
                         "sensitivity": tp / (tp + fn),
                         "specificity": tn / (tn + fp),
                         "precision": tp / (tp + fp) if tp + fp else None,
                         "false_links_within_same_user": false_link_same_user[t],
                         "false_link_rate": fp / (fp + tn)},
            "components": {"n_groups": len(clusters),
                           "false_merger_groups": false_merge_clusters,
                           "false_merger_group_fraction": false_merge_clusters / len(clusters),
                           "split_sources": sources_split,
                           "source_split_fraction": sources_split / len(np.unique(source_id))},
        }
    return results, by_class


def main():
    images, sources, letters, transforms, records = make_controlled_set()
    results, by_class = calibrate(images, sources, letters, records)
    qualifying = [t for t in THRESHOLDS if results[str(t)]["pairwise"]["precision"] is not None
                  and results[str(t)]["pairwise"]["precision"] >= .99]
    chosen = max(THRESHOLDS, key=lambda t: results[str(t)]["pairwise"]["precision"])
    report = {"seed": SEED, "source_images": len(records), "controlled_derivatives": len(images)-len(records),
              "images_in_calibration": len(images),
              "rule": "For descriptive similarity grouping, select the highest pairwise-precision threshold; no
                  classifier scores used. A threshold is accepted as a source-lineage rule only if precision >= 0.99,
                  so none qualifies.",
              "selected_descriptive_threshold": chosen,
              "selected_lineage_threshold": max(qualifying, key=lambda t: results[str(t)]["pairwise"]["sensitivity"])
                  if qualifying else None,
              "thresholds": results,
              "by_class_pair_counts": by_class,
              "caveat": "Synthetic transformations and source crops differ from the unknown release pipeline; this
                  calibration does not establish true lineage of the MNIST CSV rows."}
    path = OUT / "lineage_calibration.json"
    path.write_text(json.dumps(report, indent=2))
    print(json.dumps({k:v for k,v in report.items() if k != "by_class_pair_counts"}, indent=2))


if __name__ == "__main__":
    main()
```

A2. audit_group_structure.py

SHA-256: bae8bcd5740c7a1a55866eb9ff3e5dce74c272b98d73f8bd233df92858208fce

```
"""Describe exact-pHash groups without assuming they are source identities."""
from __future__ import annotations

import json
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

ROOT = Path(__file__).resolve().parents[2]
DATA = ROOT / "work" / "data"
SAVED = ROOT / "work" / "novel_results"
OUT = ROOT / "work" / "revision_2026_09_20"
OUT.mkdir(exist_ok=True)
LETTERS = tuple(c for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if c not in "JZ")


def describe(groups):
    _, inverse = np.unique(groups, return_inverse=True)
    counts = np.bincount(inverse)
    sizes = counts[inverse]
    bins = {"1": int(np.sum(counts == 1)), "2": int(np.sum(counts == 2)),
            "3-5": int(np.sum((counts >= 3) & (counts <= 5))),
            "6-10": int(np.sum((counts >= 6) & (counts <= 10))),
```

```
            "11-20": int(np.sum((counts >= 11) & (counts <= 20))),
            "21+": int(np.sum(counts >= 21))}
    return {"groups": len(counts), "singleton_groups": int(np.sum(counts == 1)),
            "singleton_images": int(np.sum(sizes == 1)),
            "images_in_multi_image_groups": int(np.sum(sizes > 1)),
            "median_group_size": float(np.median(counts)),
            "p90_group_size": float(np.percentile(counts, 90)),
            "maximum_group_size": int(counts.max()),
            "group_size_histogram": bins}


def main():
    train = pd.read_csv(DATA / "sign_mnist_train.csv", usecols=["label"]).label.to_numpy()
    test = pd.read_csv(DATA / "sign_mnist_test.csv", usecols=["label"]).label.to_numpy()
    labels = np.r_[train, test]
    groups = np.load(SAVED / "phash_groups_threshold0_all.npy")
    assert len(groups) == len(labels) == 34627
    counts = np.bincount(groups)
    mapped = np.load(SAVED / "reconstructed_source_mapping.npz", allow_pickle=True)
    inferred_user = mapped["user"]
    report = {"all_images": describe(groups), "official_train": describe(groups[:len(train)]),
              "official_test": describe(groups[len(train):]), "per_letter": {},
              "per_inferred_user": {},
              "guardrail": "Inferred-user rows use fallible nearest-source assignments; pHash groups are not source
                  IDs."}
    for i, letter in enumerate(LETTERS):
        original_label = ord(letter)-ord("A")
        mask = labels == original_label
        report["per_letter"][letter] = {"n_images": int(mask.sum()),
            "images_in_multi_image_groups": int(np.sum(counts[groups[mask]] > 1)),
            "fraction_in_multi_image_groups": float(np.mean(counts[groups[mask]] > 1)),
            "n_groups": int(len(np.unique(groups[mask])))}
    for user in sorted(set(inferred_user)):
        mask = inferred_user == user
        report["per_inferred_user"][str(user)] = {"n_images": int(mask.sum()),
            "fraction_in_multi_image_groups": float(np.mean(counts[groups[mask]] > 1))}
    assert report["all_images"]["images_in_multi_image_groups"] == 33597
    (OUT / "group_structure_audit.json").write_text(json.dumps(report, indent=2))
    fig, ax = plt.subplots(figsize=(6.4, 3.8))
    hist = report["all_images"]["group_size_histogram"]
    ax.bar(list(hist), list(hist.values()), color="#4C78A8")
    ax.set(xlabel="Images per exact-pHash group", ylabel="Number of groups")
    ax.grid(axis="y", alpha=.2)
    fig.tight_layout()
    fig.savefig(OUT / "exact_phash_group_size_histogram.png", dpi=300)
    plt.close(fig)
    print(json.dumps(report["all_images"], indent=2))


if __name__ == "__main__":
    main()
```

A3. audit_external_labels.py

SHA-256: 75cd63b349238b27c1e5e23c767be83ba8f1e4122ccf7c7439fdc288b6eca7a5

```
"""Audit all 24 label mappings and saved zero-shot external predictions."""
from __future__ import annotations

import csv
import json
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix

ROOT = Path(__file__).resolve().parents[2]
DATA = ROOT / "work" / "data"
EXT = ROOT / "work" / "external" / "extracted" / "dataset5"
SAVED = ROOT / "work" / "novel_results"
OUT = ROOT / "work" / "revision_2026_09_20"
OUT.mkdir(exist_ok=True)
LETTERS = tuple(c for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if c not in "JZ")
EXPECTED_ORIGINAL_LABELS = tuple(i for i in range(26) if i not in (9, 25))


def unit_predictions(y, pred):
    true_unit, pred_unit = [], []
    for start in range(0, len(y), 20):
        block_y = y[start:start+20]
```

```
        block_pred = pred[start:start+20]
        assert len(block_y) == 20 and np.all(block_y == block_y[0])
        true_unit.append(int(block_y[0]))
        pred_unit.append(int(np.argmax(np.bincount(block_pred, minlength=24))))
    return np.asarray(true_unit), np.asarray(pred_unit)


def main():
    train = pd.read_csv(DATA / "sign_mnist_train.csv", usecols=["label"])
    test = pd.read_csv(DATA / "sign_mnist_test.csv", usecols=["label"])
    original = tuple(sorted(set(train.label) | set(test.label)))
    assert original == EXPECTED_ORIGINAL_LABELS, (original, EXPECTED_ORIGINAL_LABELS)
    label_to_index = {label: index for index, label in enumerate(original)}
    assert all(label_to_index[ord(letter)-ord("A")] == index for index, letter in enumerate(LETTERS))
    assert set(train.label) == set(test.label) == set(original)

    subjects = sorted(p for p in EXT.iterdir() if p.is_dir())
    assert len(subjects) == 5
    frames, y = [], []
    for subject in subjects:
        for index, letter in enumerate(LETTERS):
            folder = subject / letter.lower()
            assert folder.is_dir(), folder
            files = sorted(folder.glob("color_*.png"), key=lambda p: int(p.stem.split("_")[-1]))
            assert len(files) >= 40, (folder, len(files))
            chosen = np.linspace(len(files)//4, 3*len(files)//4-1, 20, dtype=int)
            assert len(set(chosen)) == 20
            frames.extend(str(files[i]) for i in chosen)
            y.extend([index] * 20)
    y = np.asarray(y)
    assert len(y) == 2400 and len(frames) == 2400
    saved_y = np.load(SAVED / "external_depth_guided_crop_labels.npy")
    assert np.array_equal(y, saved_y), "Saved external prediction order differs from audited folder order"

    mapping = [{"original_integer_label": original[i], "ASL_letter": letter,
                "contiguous_model_index": i, "external_folder": letter.lower()}
               for i, letter in enumerate(LETTERS)]
    models = {
        "original_cnn_ensemble": SAVED / "external_depth_guided_crop_cnn_ensemble_predictions.npy",
        "nested_standard_cnn_ensemble": SAVED / "extended_external_depth_guided_crop_standard_cnn_predictions.npy",
        "nested_residual_cnn_ensemble": SAVED /
            "extended_external_depth_guided_crop_compact_residual_cnn_predictions.npy",
        "hog_svm": SAVED / "external_depth_guided_crop_hog_svm_predictions.npy",
    }
    matrices, summary = {}, {}
    for name, path in models.items():
        pred = np.load(path)
        assert len(pred) == len(y) and np.all((0 <= pred) & (pred < 24)), name
        true_unit, pred_unit = unit_predictions(y, pred)
        cm = confusion_matrix(true_unit, pred_unit, labels=np.arange(24))
        assert cm.shape == (24, 24) and int(cm.sum()) == 120
        assert np.all(cm.sum(axis=1) == 5)
        matrices[name] = cm
        summary[name] = {"correct_units": int(np.trace(cm)), "total_units": 120,
                         "accuracy": float(np.trace(cm)/120),
                         "per_letter_correct_of_five": {letter: int(cm[i,i]) for i, letter in enumerate(LETTERS)}}
        with (OUT / f"external_confusion_{name}.csv").open("w", newline="") as f:
            writer = csv.writer(f)
            writer.writerow(["true\\predicted", *LETTERS])
            for letter, row in zip(LETTERS, cm):
                writer.writerow([letter, *map(int, row)])

    fig, ax = plt.subplots(figsize=(8.2, 7.3))
    cm = matrices["nested_standard_cnn_ensemble"]
    ax.imshow(cm, cmap="Blues", vmin=0, vmax=5)
    ax.set_xticks(np.arange(24), LETTERS, fontsize=7)
    ax.set_yticks(np.arange(24), LETTERS, fontsize=7)
    ax.set(xlabel="Predicted letter", ylabel="External folder letter")
    for i in range(24):
        for j in range(24):
            if cm[i,j]:
                ax.text(j, i, str(cm[i,j]), ha="center", va="center", fontsize=6,
                        color="white" if cm[i,j] >= 3 else "black")
    fig.tight_layout()
    fig.savefig(OUT / "external_24x24_confusion_standard_cnn.png", dpi=300)
    plt.close(fig)
    report = {"mapping": mapping, "subjects": [p.name for p in subjects],
              "n_frames": len(y), "n_subject_letter_units": 120,
              "saved_label_order_assertion": "passed",
              "unit_vote_tie_rule": "Lowest contiguous class index, matching numpy.argmax",
              "models": summary,
```

```
            "scope": "This checks programmatic label alignment and gives a 24x24 error audit; it does not
                independently verify the linguistic correctness of every external folder label."}
    (OUT / "external_label_audit.json").write_text(json.dumps(report, indent=2))
    print(json.dumps({k:v for k,v in report.items() if k != "mapping"}, indent=2))


if __name__ == "__main__":
    main()
```

A4. audit_source_confidence.py

SHA-256: a4c04d3c9bbbc0804e8b51d7f5fc0cadd44d03a6f3457a31a5974406fdd24edb

```
"""Report nearest-source assignment coverage under calibrated user margins."""
from __future__ import annotations

import json
from pathlib import Path

import numpy as np

ROOT = Path(__file__).resolve().parents[2]
OLD = ROOT / "work" / "novel_results"
OUT = ROOT / "work" / "revision_2026_09_20"
OUT.mkdir(exist_ok=True)


def main():
    controlled = json.loads((OLD / "source_reconstruction_validation.json").read_text())
    mapped = np.load(OLD / "reconstructed_source_mapping.npz", allow_pickle=True)
    margin = mapped["user_margin"]
    ntrain = 27455
    selection = {}
    for quantile in (0, 10, 25, 50, 75):
        control = controlled["confidence_sensitivity"][str(quantile)]
        threshold = control["margin_threshold"]
        selection[str(quantile)] = {
            "controlled_margin_threshold": threshold,
            "controlled_coverage": control["coverage"],
            "controlled_source_top1_accuracy": control["source_accuracy"],
            "controlled_user_accuracy": control["user_accuracy"],
            "released_csv_fraction_above_threshold": float(np.mean(margin >= threshold)),
            "released_train_fraction_above_threshold": float(np.mean(margin[:ntrain] >= threshold)),
            "released_test_fraction_above_threshold": float(np.mean(margin[ntrain:] >= threshold)),
        }
    report = {"algorithm": "Each query is forced to its nearest same-letter source among 1,680 candidates; no
        rejection threshold existed in the submitted analysis.",
              "margin_definition": "Score(best candidate from another inferred user) minus score(best candidate
                  overall). Because the best candidate is globally minimal, the margin is nonnegative by construction;
                   positivity is not evidence of correctness.",
              "controlled_derivatives": controlled["n_controlled_derivatives"],
              "controlled_source_accuracy": controlled["source_top1_accuracy"],
              "controlled_user_accuracy": controlled["user_accuracy"],
              "threshold_sensitivity": selection,
              "recommended_interpretation": "High margins enrich correct assignments on controlled derivatives but do
                  not certify source or user IDs for released CSV rows; do not call inferred-user results
                  signer-independent."}
    (OUT / "source_confidence_audit.json").write_text(json.dumps(report, indent=2))
    print(json.dumps(report, indent=2))


if __name__ == "__main__":
    main()
```

A5. diagnose_residual_fold.py

SHA-256: 6ddce13cd5112fb5f0f2bfa15a44b53c9e25085166bd3d1996988cec5b068673

```
"""Characterize the worst residual-CNN fold without claiming a proven cause."""
from __future__ import annotations

import json
from pathlib import Path

import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.model_selection import StratifiedGroupKFold

ROOT = Path(__file__).resolve().parents[2]
DATA = ROOT / "work" / "data"
SAVED = ROOT / "work" / "novel_results"
```

```
OUT = ROOT / "work" / "revision_2026_09_20"
OUT.mkdir(exist_ok=True)
LETTERS = tuple(c for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if c not in "JZ")


def main():
    labels = pd.read_csv(DATA / "sign_mnist_train.csv", usecols=["label"]).label.to_numpy()
    original = sorted(np.unique(labels))
    mapping = {label: i for i, label in enumerate(original)}
    y = np.array([mapping[v] for v in labels])
    groups = np.load(SAVED / "train_phash_groups_threshold0.npy")
    folds = list(StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=25).split(np.zeros(len(y)), y, groups))
    outer = json.loads((SAVED / "extended_grouped_evaluation.json").read_text())
    assert all(len(te) == outer["outer_folds"][i]["n_test"] for i, (_, te) in enumerate(folds))
    report = {"folds": []}
    for fold, (_, test) in enumerate(folds):
        row = {"fold": fold, "n_test": len(test), "seeds": {}}
        for seed in (25, 37, 49):
            pred = np.load(SAVED / f"extended_compact_residual_cnn_seed{seed}_oof_predictions.npy")[test]
            prob = np.load(SAVED / f"extended_compact_residual_cnn_seed{seed}_oof_probabilities.npy")[test]
            cm = confusion_matrix(y[test], pred, labels=np.arange(24))
            recall = np.divide(np.diag(cm), cm.sum(axis=1), out=np.zeros(24), where=cm.sum(axis=1)>0)
            row["seeds"][str(seed)] = {
                "accuracy": float(accuracy_score(y[test], pred)),
                "macro_f1": float(f1_score(y[test], pred, average="macro", zero_division=0)),
                "n_predicted_classes": int(len(np.unique(pred))),
                "zero_recall_letters": [LETTERS[i] for i in np.flatnonzero(recall == 0)],
                "lowest_recall_letters": [{"letter": LETTERS[i], "recall": float(recall[i]),
                                           "test_count": int(cm.sum(axis=1)[i]),
                                           "most_common_predicted": LETTERS[int(np.argmax(cm[i]))]}
                                          for i in np.argsort(recall)[:8]],
                "mean_max_probability": float(np.mean(np.max(prob, axis=1))),
                "predicted_class_counts": {LETTERS[i]: int(n) for i,n in enumerate(np.bincount(pred,minlength=24)) if
                    n},
            }
        report["folds"].append(row)
    report["interpretation_guardrail"] = "Prediction distributions reveal the failure pattern; saved outputs do not
        establish its training-time cause."
    (OUT / "residual_fold_diagnostic.json").write_text(json.dumps(report, indent=2))
    worst = report["folds"][3]["seeds"]["37"]
    print(json.dumps({"fold": 3, "seed": 37, **{k:v for k,v in worst.items() if k != "predicted_class_counts"}},
        indent=2))


if __name__ == "__main__":
    main()
```

A6. diagnose_residual_fold_seed.py

SHA-256: a2f20de39090e1359443c75f9b72bc6258827033040910323 3bcba38928b842b

```
"""Post hoc diagnostic for the compact-residual-CNN fold 3, seed 37 failure.

This script reproduces the already reported outer-fold refit. It does not
select a model or change the primary estimates. The held-out outer fold is
examined only to characterize the previously observed failure.
"""
from __future__ import annotations

import json
import platform
import sys
import time
from pathlib import Path

import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support


ROOT = Path(__file__).resolve().parents[2]
CODE = ROOT / "work"
sys.path.insert(0, str(CODE))

from extended_grouped_evaluation import (  # noqa: E402
    build_cnn,
    metrics,
    outer_folds,
)

OUT = ROOT / "work" / "novel_results"
```

```
DATA = ROOT / "work" / "data"
RESULT_PATH = OUT / "residual_fold3_seed37_diagnostic.json"
HISTORY_PATH = OUT / "residual_fold3_seed37_history.csv"
PREDICTION_PATH = OUT / "residual_fold3_seed37_epoch20_predictions.npy"

FOLD = 3
SEED = 37
SELECTED_EPOCH = 11
DIAGNOSTIC_EPOCH = 20
CONFIG = {
    "name": "original_optimizer",
    "learning_rate": 0.001,
    "dropout": 0.5,
    "weight_decay": 0.0,
}


def evaluation_record(
    model, x, y, indices, epoch: int, partition: str, save_prediction: bool = True
) -> dict:
    prob = model.predict(x[indices], batch_size=256, verbose=0)
    pred = prob.argmax(axis=1)
    p, r, f, support = precision_recall_fscore_support(
        y[indices], pred, labels=np.arange(24), zero_division=0
    )
    cm = confusion_matrix(y[indices], pred, labels=np.arange(24))
    record = {
        "epoch": epoch,
        "partition": partition,
        **metrics(y[indices], pred),
        "predicted_class_count": int(len(np.unique(pred))),
        "zero_recall_class_count": int(np.sum(r == 0)),
        "zero_recall_classes": np.flatnonzero(r == 0).astype(int).tolist(),
        "mean_max_probability": float(prob.max(axis=1).mean()),
        "per_class_precision": p.tolist(),
        "per_class_recall": r.tolist(),
        "per_class_f1": f.tolist(),
        "per_class_support": support.astype(int).tolist(),
        "confusion_matrix": cm.astype(int).tolist(),
    }
    if save_prediction and partition == "outer_test":
        np.save(PREDICTION_PATH, pred.astype(np.int32))
    return record


class DiagnosticCallback(tf.keras.callbacks.Callback):
    def __init__(self, x, y, outer_train, outer_test):
        super().__init__()
        self.x = x
        self.y = y
        self.outer_train = outer_train
        self.outer_test = outer_test
        self.records: list[dict] = []

    def on_epoch_end(self, epoch, logs=None):
        completed = epoch + 1
        logs = logs or {}
        print(
            f"epoch={completed} loss={logs.get('loss', float('nan')):.6f} "
            f"training_batch_accuracy={logs.get('accuracy', float('nan')):.6f}",
            flush=True,
        )
        if completed in (SELECTED_EPOCH, DIAGNOSTIC_EPOCH):
            for name, indices in (
                ("outer_train", self.outer_train),
                ("outer_test", self.outer_test),
            ):
                record = evaluation_record(
                    self.model, self.x, self.y, indices, completed, name
                )
                self.records.append(record)
                print(
                    f"diagnostic epoch={completed} partition={name} "
                    f"macro_f1={record['macro_f1']:.6f} "
                    f"classes={record['predicted_class_count']} "
                    f"zero_recall={record['zero_recall_class_count']}",
                    flush=True,
                )


def main() -> None:
```

```
    frame = pd.read_csv(DATA / "sign_mnist_train.csv")
    original_labels = frame.label.to_numpy(int)
    label_values = sorted(np.unique(original_labels))
    label_map = {value: i for i, value in enumerate(label_values)}
    y = np.array([label_map[value] for value in original_labels], dtype=np.int32)
    raw = frame.iloc[:, 1:].to_numpy(np.float32) / 255.0
    x = raw.reshape(-1, 28, 28, 1)
    groups = np.load(OUT / "train_phash_groups_threshold0.npy")
    outer_train, outer_test = outer_folds(y, groups)[FOLD]

    assert not (set(groups[outer_train]) & set(groups[outer_test]))
    assert len(np.unique(y[outer_train])) == 24
    assert len(np.unique(y[outer_test])) == 24

    model = build_cnn("compact_residual_cnn", CONFIG, SEED)
    callback = DiagnosticCallback(x, y, outer_train, outer_test)
    started = time.perf_counter()
    history = model.fit(
        x[outer_train],
        y[outer_train],
        epochs=DIAGNOSTIC_EPOCH,
        batch_size=128,
        shuffle=True,
        callbacks=[callback],
        verbose=0,
    )
    elapsed = time.perf_counter() - started
    history_frame = pd.DataFrame(
        {
            "epoch": np.arange(1, len(history.history["loss"]) + 1),
            "loss": history.history["loss"],
            "training_batch_accuracy": history.history["accuracy"],
        }
    )
    history_frame.to_csv(HISTORY_PATH, index=False)

    result = {
        "purpose": (
            "Post hoc characterization of a previously observed failure; "
            "not used for model selection or primary performance estimation."
        ),
        "fold": FOLD,
        "seed": SEED,
        "selected_epoch": SELECTED_EPOCH,
        "diagnostic_epoch": DIAGNOSTIC_EPOCH,
        "config": CONFIG,
        "n_outer_train": int(len(outer_train)),
        "n_outer_test": int(len(outer_test)),
        "n_train_groups": int(len(np.unique(groups[outer_train]))),
        "n_test_groups": int(len(np.unique(groups[outer_test]))),
        "group_overlap": 0,
        "label_values": [int(value) for value in label_values],
        "elapsed_seconds": float(elapsed),
        "records": callback.records,
        "history": history_frame.to_dict(orient="records"),
        "versions": {
            "python": platform.python_version(),
            "numpy": np.__version__,
            "pandas": pd.__version__,
            "tensorflow": tf.__version__,
        },
    }
    RESULT_PATH.write_text(json.dumps(result, indent=2))
    print(f"saved {RESULT_PATH}", flush=True)


if __name__ == "__main__":
    main()
```

A7. diagnose_residual_bn_recalibration.py

SHA-256: 483f473a1874a08f8222e517fa0af900306d13e47f09eb77abbc5943bc9975b8

```
"""Test whether inference-state recalibration repairs the reproduced failure.

The model is trained for the originally selected 11 epochs. Metrics are
recorded before and after one no-gradient pass over the outer-training images
with training behavior enabled. This pass updates BatchNormalization moving
statistics but does not update trainable weights. It is a post hoc mechanism
check and is not part of the primary model-selection protocol.
"""
from __future__ import annotations
```

```
import json
import sys
from pathlib import Path

import numpy as np
import pandas as pd
import tensorflow as tf


ROOT = Path(__file__).resolve().parents[2]
sys.path.insert(0, str(ROOT / "work"))

from extended_grouped_evaluation import build_cnn, outer_folds  # noqa: E402
from diagnose_residual_fold_seed import evaluation_record  # noqa: E402


OUT = ROOT / "work" / "novel_results"
DATA = ROOT / "work" / "data"
RESULT_PATH = OUT / "residual_fold3_seed37_bn_recalibration.json"
CONFIG = {
    "name": "original_optimizer",
    "learning_rate": 0.001,
    "dropout": 0.5,
    "weight_decay": 0.0,
}


def main() -> None:
    frame = pd.read_csv(DATA / "sign_mnist_train.csv")
    original_labels = frame.label.to_numpy(int)
    label_map = {
        value: i for i, value in enumerate(sorted(np.unique(original_labels)))
    }
    y = np.array([label_map[value] for value in original_labels], dtype=np.int32)
    x = (
        frame.iloc[:, 1:].to_numpy(np.float32).reshape(-1, 28, 28, 1) / 255.0
    )
    groups = np.load(OUT / "train_phash_groups_threshold0.npy")
    outer_train, outer_test = outer_folds(y, groups)[3]

    model = build_cnn("compact_residual_cnn", CONFIG, 37)
    history = model.fit(
        x[outer_train],
        y[outer_train],
        epochs=11,
        batch_size=128,
        shuffle=True,
        verbose=0,
    )
    before = [
        evaluation_record(model, x, y, outer_train, 11, "outer_train", False),
        evaluation_record(model, x, y, outer_test, 11, "outer_test", False),
    ]

    batch_normalization_layers = [
        layer for layer in model.layers if isinstance(layer, tf.keras.layers.BatchNormalization)
    ]
    trainable_before = [value.numpy().copy() for value in model.trainable_variables]
    data = tf.data.Dataset.from_tensor_slices(x[outer_train]).batch(128)
    for batch in data:
        model(batch, training=True)
    trainable_unchanged = all(
        np.array_equal(before_value, after_value.numpy())
        for before_value, after_value in zip(trainable_before, model.trainable_variables)
    )

    after = [
        evaluation_record(model, x, y, outer_train, 11, "outer_train", False),
        evaluation_record(model, x, y, outer_test, 11, "outer_test", False),
    ]
    result = {
        "purpose": (
            "Post hoc BatchNormalization mechanism check; not used for model "
            "selection or primary performance estimation."
        ),
        "fold": 3,
        "seed": 37,
        "epoch": 11,
        "batch_normalization_layer_count": len(batch_normalization_layers),
        "recalibration_passes": 1,
        "recalibration_batch_size": 128,
```

```
        "trainable_weights_unchanged": bool(trainable_unchanged),
        "final_training_batch_loss": float(history.history["loss"][-1]),
        "final_training_batch_accuracy": float(history.history["accuracy"][-1]),
        "before_recalibration": before,
        "after_recalibration": after,
    }
    RESULT_PATH.write_text(json.dumps(result, indent=2))
    print(json.dumps({
        "trainable_weights_unchanged": trainable_unchanged,
        "before_train_f1": before[0]["macro_f1"],
        "before_test_f1": before[1]["macro_f1"],
        "after_train_f1": after[0]["macro_f1"],
        "after_test_f1": after[1]["macro_f1"],
        "saved": str(RESULT_PATH),
    }, indent=2), flush=True)


if __name__ == "__main__":
    main()
```

A8. known_source_experiment.py

SHA-256: e6d9cf5b170491a63812509231a3ebc6625b9b5848553893e884aa5cac1ef830

```
"""Controlled random-versus-source-disjoint experiment with known lineage.

Inputs are the 1,680 source crops and five deterministic synthetic descendants
per crop. These are not the released Sign Language MNIST CSV rows.
"""
from __future__ import annotations

import json
import os
import random
import time
from pathlib import Path

os.environ.setdefault("TF_CPP_MIN_LOG_LEVEL", "2")
os.environ.setdefault("MPLCONFIGDIR", "/private/tmp/mpl-nhsjs")

import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.model_selection import StratifiedGroupKFold, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
import tensorflow as tf

from calibrate_lineage import make_controlled_set
from classical_protocol_experiments import hog
from extended_grouped_evaluation import build_cnn

ROOT = Path(__file__).resolve().parents[2]
OUT = ROOT / "work" / "revision_2026_09_20"
OUT.mkdir(exist_ok=True)
SEEDS = (25, 37, 49)
CONFIG = {"name": "fixed_original", "learning_rate": 0.001, "dropout": 0.5, "weight_decay": 0.0}


def met(y, pred):
    precision, recall, f1, _ = precision_recall_fscore_support(y, pred, average="macro", zero_division=0)
    return {"accuracy": float(accuracy_score(y, pred)), "macro_precision": float(precision),
            "macro_recall": float(recall), "macro_f1": float(f1)}


def set_seed(seed):
    os.environ["PYTHONHASHSEED"] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    try:
        tf.config.experimental.enable_op_determinism()
    except Exception:
        pass


def evaluate_outer(name, fold, train, test, x, flat, features, y, source, report):
    assert not set(train) & set(test)
    if name == "source_disjoint":
        assert not set(source[train]) & set(source[test])
        inner = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=100+fold)
```

```
        fit0, val0 = next(inner.split(np.zeros(len(train)), y[train], source[train]))
        fit, val = train[fit0], train[val0]
        assert not set(source[fit]) & set(source[val])
    else:
        fit, val = train_test_split(train, test_size=.2, stratify=y[train], random_state=100+fold)
    row = {"protocol": name, "fold": fold, "n_fit": len(fit), "n_validation": len(val),
           "n_test": len(test), "test_source_count": len(np.unique(source[test])),
           "train_test_source_overlap": len(set(source[train]) & set(source[test])),
           "metrics": {}, "cnn_runs": []}
    print(f"{name} fold {fold}: classical baselines", flush=True)
    model = KNeighborsClassifier(n_neighbors=1, algorithm="brute", n_jobs=2)
    model.fit(flat[train], y[train])
    pred = model.predict(flat[test])
    row["metrics"]["one_nearest_neighbor_pixels"] = met(y[test], pred)
    raw = Pipeline([("scale", StandardScaler()),
                    ("svm", LinearSVC(C=.1, random_state=25, dual="auto", max_iter=3000))])
    raw.fit(flat[train], y[train])
    row["metrics"]["raw_pixel_svm"] = met(y[test], raw.predict(flat[test]))
    edge = LinearSVC(C=.1, random_state=25, dual="auto", max_iter=3000)
    edge.fit(features[train], y[train])
    row["metrics"]["hog_svm"] = met(y[test], edge.predict(features[test]))
    probs = []
    for seed in SEEDS:
        start = time.perf_counter()
        set_seed(seed)
        model = build_cnn("standard_cnn", CONFIG, seed)
        callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True)
        history = model.fit(x[fit], y[fit], validation_data=(x[val], y[val]),
                            epochs=25, batch_size=128, shuffle=True, verbose=0,
                            callbacks=[callback])
        prob = model.predict(x[test], batch_size=256, verbose=0)
        pred = prob.argmax(1)
        run = {"seed": seed, "epochs_run": len(history.history["loss"]),
               "selected_epoch": int(np.argmin(history.history["val_loss"]) + 1),
               "fit_seconds": float(time.perf_counter()-start), **met(y[test], pred)}
        row["cnn_runs"].append(run)
        probs.append(prob)
        print(f"{name} fold {fold} seed {seed}: macro F1={run['macro_f1']:.4f}", flush=True)
    row["metrics"]["standard_cnn_3_seed_ensemble"] = met(y[test], np.mean(probs, axis=0).argmax(1))
    report["outer_results"].append(row)
    (OUT / "known_source_experiment.json").write_text(json.dumps(report, indent=2))


def main():
    images, source, y, transforms, records = make_controlled_set()
    assert len(images) == 10080 and len(np.unique(source)) == 1680
    x = images.astype(np.float32)[..., None] / 255.0
    flat = x.reshape(len(x), -1)
    features = hog(flat)
    report = {"seed": 25, "source_images": 1680, "derivatives_per_source": 5,
              "n_images": len(x), "class_count": 24,
              "design": "Prespecified fixed models. One stratified 80/20 image-level split and five outer
                  source-disjoint folds. Three CNN seeds per outer test set. CNN checkpoint selected on
                  development-only inner validation. No test-row observations used for preprocessing, training, or
                  checkpoint selection.",
              "caveat": "Controlled synthetic derivatives are not the released Sign Language MNIST derivatives and
                  cannot validate their unknown source mapping.",
              "versions": {"numpy": np.__version__, "scikit_learn": __import__("sklearn").__version__, "tensorflow":
                  tf.__version__},
              "outer_results": []}
    random_train, random_test = train_test_split(np.arange(len(y)), test_size=.2,
                                                 stratify=y, random_state=25)
    evaluate_outer("image_random", 0, random_train, random_test, x, flat, features, y, source, report)
    splitter = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=25)
    for fold, (train, test) in enumerate(splitter.split(np.zeros(len(y)), y, source)):
        evaluate_outer("source_disjoint", fold, train, test, x, flat, features, y, source, report)
    print(json.dumps({"completed_outer_results": len(report["outer_results"]),
                      "last_result": report["outer_results"][-1]}, indent=2), flush=True)


if __name__ == "__main__":
    main()
```

A9. make_revision_figures.py

SHA-256: 22610c87a2418d2f50f61ea797403060baf68fe5583deb9696a17b66c648a838

```
"""Create compact manuscript figures from saved revision results."""
from __future__ import annotations

import json
from pathlib import Path
```

```
import matplotlib.pyplot as plt
import numpy as np

ROOT = Path(__file__).resolve().parents[2]
OUT = ROOT / "work" / "revision_2026_09_20"
BLUE, ORANGE, GREY = "#245B83", "#D67B31", "#6D7580"


def lineage_calibration():
    report = json.loads((OUT / "lineage_calibration.json").read_text())
    ts = [0, 2, 4, 6]
    sens = [100*report["thresholds"][str(t)]["pairwise"]["sensitivity"] for t in ts]
    prec = [100*report["thresholds"][str(t)]["pairwise"]["precision"] for t in ts]
    fig, ax = plt.subplots(figsize=(6.8, 3.4))
    x = np.arange(4)
    ax.bar(x-.18, sens, .34, label="Same-source pair sensitivity", color=BLUE)
    ax.bar(x+.18, prec, .34, label="Linked-pair precision", color=ORANGE)
    ax.axhline(99, color=GREY, linestyle="--", linewidth=1, label="99% lineage criterion")
    ax.set_xticks(x, ["Exact", "≤2 bits", "≤4 bits", "≤6 bits"])
    ax.set_ylim(0, 105)
    ax.set_ylabel("Pairwise performance (%)")
    ax.legend(frameon=False, loc="upper center", bbox_to_anchor=(.5, 1.16),
              ncol=3, fontsize=8)
    ax.grid(axis="y", alpha=.18)
    fig.tight_layout()
    fig.savefig(OUT / "lineage_calibration.png", dpi=300, bbox_inches="tight")
    plt.close(fig)


def known_source():
    path = OUT / "known_source_experiment.json"
    if not path.exists():
        return
    report = json.loads(path.read_text())
    rows = report["outer_results"]
    grouped = [row for row in rows if row["protocol"] == "source_disjoint"]
    if len(grouped) != 5:
        return
    random = [row for row in rows if row["protocol"] == "image_random"][0]
    names = ["1-NN pixels", "Raw-pixel SVM", "HOG-SVM", "CNN ensemble"]
    keys = ["one_nearest_neighbor_pixels", "raw_pixel_svm", "hog_svm", "standard_cnn_3_seed_ensemble"]
    a = np.array([100*random["metrics"][key]["macro_f1"] for key in keys])
    g = np.array([100*np.mean([row["metrics"][key]["macro_f1"] for row in grouped]) for key in keys])
    spread = np.array([100*np.std([row["metrics"][key]["macro_f1"] for row in grouped], ddof=1) for key in keys])
    fig, ax = plt.subplots(figsize=(7.2, 3.8))
    x = np.arange(len(names))
    ax.bar(x-.18, a, .34, color=BLUE, label="Random image holdout")
    ax.bar(x+.18, g, .34, color=ORANGE, label="Five source-disjoint folds", yerr=spread, capsize=3)
    ax.set_xticks(x, names)
    ax.set_ylabel("Macro F1 (%)")
    ax.set_ylim(0, 105)
    ax.legend(frameon=False, loc="upper right", fontsize=8)
    ax.grid(axis="y", alpha=.18)
    fig.tight_layout()
    fig.savefig(OUT / "known_source_protocol_comparison.png", dpi=300)
    plt.close(fig)


def main():
    lineage_calibration()
    known_source()


if __name__ == "__main__":
    main()
```