PROVENANCE NETWORKS: END-TO-END EXEMPLAR-BASED EXPLAINABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce *provenance networks*: a shared backbone feeds a task branch (*e.g.* classification) and a jointly trained index branch that predicts each input's exemplar index within the training set, making the network behave like a learned, end-to-end k-nearest-neighbor classifier. Training-set index accuracy alone is a poor measure of usefulness: a model reaching 96–98% index accuracy on MNIST can still fail almost completely at recovering a known source for a perturbed test input. We introduce a test-time, ground-truth-backed source-recovery protocol (MNIST, FashionMNIST, CIFAR-10) that exposes this gap and show it closes once the index branch is trained with augmentation matched to the evaluation perturbations – at which point it decisively beats an embedding-space kNN baseline, most dramatically on CIFAR-10 (70.8% vs. 2.2% top-1 under the strongest perturbation). A ProtoPNet-style baseline is structurally incapable of this task, capped near 0% by construction despite comparable classification accuracy. Even when the class-conditional design's own prediction is wrong – and its retrieved exemplar's label is therefore guaranteed wrong too – the retrieved exemplar's content is still a significantly better guide to the model's confusion than a random wrong-class exemplar (67.0% win rate, $p = 5.2 \times 10^{-7}$), a debugging signal precisely when the model errs. On scalability, a subset of training exemplars preserves task accuracy with little loss, and a factorized index head cuts the index layer's parameters by 41–500× but converges substantially more slowly. The same mechanism extends, exploratorily, to further uses beyond classification – membership inference, robustness to image distortions, and generative modeling – and sharing representation between the two branches is close to free, sometimes even improving retrieval quality. Provenance networks provide a foundation for future explainability by directly linking predictions to the training examples that informed them.

1 INTRODUCTION

Most neural networks map raw input through a sequence of nonlinear transforms into a compact, task-aligned latent representation, and in doing so lose explicit reference to individual training samples: a network typically cannot say which training example a given test-time decision corresponds to. We introduce *provenance networks* to close this gap: from a shared backbone, one branch performs the primary task (*e.g.* classification) while a second, jointly trained branch predicts the index of the corresponding exemplar within the training set (Figure 1), so that at inference time the system retrieves the training sample most likely to be the input's source – a learned k-nearest-neighbor (kNN) classifier, differentiable and trained end-to-end, with each output justified by a concrete, retrievable exemplar rather than an abstract latent code.

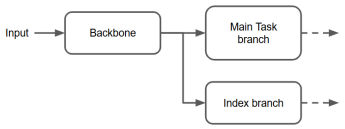


Figure 1: Provenance networks.

This raises three questions we take up in turn. **(Q1)** Can a network be trained to recover the identity of its own training examples? As Section 3 shows, yes, easily: an index-prediction branch reaches near-perfect accuracy on the training set. **(Q2)** Does that ability transfer to genuinely new, unseen inputs – *i.e.* does high training-index accuracy imply the same mechanism can identify a plausible training source for a novel test input? Section 6 shows the answer is no by default: a model with 96–98% training-index accuracy can fail almost completely at test-time source recovery. **(Q3)** Can this gap be closed? Section 6 also shows yes, if the index branch is explicitly trained to be invariant to the perturbations it will be evaluated under, at which point it substantially outperforms embedding-space

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

kNN retrieval. This progression – memorization is easy, attribution does not follow automatically, but can be trained in – is the central empirical thread of the paper.

Training-set index accuracy alone cannot show whether a retrieved exemplar is a genuine explanation or merely a similar-looking image (see the provenance/attribution/influence distinction in Related Work); our semi-synthetic protocol (Section 6.1) tests *test-time source recovery* against a known training source, not causal influence, and we make no causal claim. Broader applications (generative modeling, membership inference, robustness) are noted only briefly, with exploratory results kept in the appendix.

Our contributions: (i) a provenance network architecture with single-branch (tunable memorization/generalization via label mixing) and two-branch (class-independent/-conditional) variants unifying task prediction with training-sample attribution (Section 3); (ii) a test-time attribution-quality evaluation protocol against known ground truth, showing the trained index head is outperformed by simple kNN retrieval under geometric perturbation *unless* trained with matched augmentation, at which point it decisively outperforms kNN (Section 6); and (iii) two scalability mitigations for the index head's otherwise-unbounded output size – subset sampling and a factorized index head – along with a proposed, untested hierarchical extension (Section 5.2).

2 Related work

Provenance, attribution, and influence: a terminology note. We use three related but distinct terms precisely, since conflating them is a common source of overclaiming in this space. *Provenance* is the model's learned mapping from an input to identifiable training examples – a mechanism, not a validation of correctness. *Attribution* (Section 6) is whether that mapping recovers a *known* training source under a controlled, ground-truth-backed protocol. *Influence* (Koh & Liang, 2017; Ghorbani & Zou, 2019; Hammoudeh et al., 2023) is the counterfactual effect of adding, removing, or reweighting a training point on model behavior – a causal notion, and the one the training-data-attribution literature typically formalizes. Our mechanism and evaluation speak to the first two; we do not measure influence or claim a retrieved exemplar caused its prediction.

The black-box nature of neural networks has prompted many approaches to illuminate their internals (Zhang et al., 2021; Lipton, 2018; Linardatos et al., 2020). Influence functions (Koh & Liang, 2017) and data Shapley values (Ghorbani & Zou, 2019) give rigorous measures of training-sample influence but are computationally expensive and impractical at scale, as is exact leave-one-out retraining (Hammoudeh et al., 2023). Perturbation-, game-theoretic-, and saliency-based methods (Ribeiro et al., 2016; Lundberg & Lee, 2017; Simonyan et al., 2014; Sundararajan et al., 2017; Smilkov et al., 2017; Selvaraju et al., 2017), along with feature-visualization/circuits work (Olah et al., 2017; 2020; Zhou et al., 2016), give feature- or neuron-level insight but cannot attribute a prediction to specific training samples; retrieval-based systems (Mitra & Craswell, 2018; Snell et al., 2017) enable sample identification at scale but offer no unified control over memorization vs. generalization. Provenance networks integrate classification and training-sample attribution in one jointly trained model; we separately test, and reject, the hypothesis that this integration also improves adversarial robustness (Appx. 9.14.1).

Self-explainable and prototype-based models. Prototype networks such as ProtoPNet (Chen et al., 2019) build interpretability into the architecture by classifying via similarity to a small set of learned class prototypes; matching networks similarly classify by similarity to a labeled support set (Vinyals et al., 2016; Xu et al., 2018). Provenance networks differ mainly in *granularity*: ProtoPNet's post-"push" prototypes are real training-image patches, but only a handful per class (10, typically), versus one retrievable index per training sample here; memory-augmented architectures (*e.g.* Neural Turing Machines (Graves et al., 2014)) instead store *learned latent* memories that need not correspond to any real input. Section 6 quantifies this granularity gap directly.

LLM interpretability work uses sparse autoencoders (Lieberum et al., 2024) and mechanistic interpretability (Nanda et al., 2023), while retrieval-augmented generation grounds outputs in external evidence (Lewis et al., 2020); we view provenance networks as a training-data-grounded counterpart in the vision domain studied here and do not evaluate language models (Section 8).

3 Provenance Networks

We illustrate the core concepts using multi-class prediction, though the mechanism is not limited to it: Section 7.3 applies the same index-branch mechanism to a generative (non-classification)

task, conditioned on a class label rather than a task prediction. For a detailed view of the model architectures, see Appx. 9.1 and 9.2.

3.1 I: SINGLE BRANCH NETWORK

Here, essentially each datapoint is mapped to its index (Appx. Fig 7). Inputs may be presented in random order during training, but their indices remain constant. To ease training on large datasets, an input is not always mapped to its own index but is occasionally mapped to a different index from the same class (Appx. Fig. 8). Let i denote the true training index of an input sample x, and let $\mathcal{I}_y$ denote the set of indices belonging to the same class y, excluding i. The target index t for training is sampled as $t = i$ with probability $1 - \alpha$ (memorization) and $t \sim \text{Uniform}(\mathcal{I}_y \setminus \{i\})$ with probability α (generalization). The network outputs logits $\mathbf{p}$ over all training indices, with cross-entropy loss $\mathcal{L} = -\log \hat{p}_t$, where $\hat{p}_t$ is the predicted probability of the target index t after softmax. This interpolates between pure memorization ($\alpha = 0$, *e.g.* index acc. $\approx 99\%$) and pure semantic generalization ($\alpha = 1$, *e.g.* semantic acc. $\approx 100\%$), with intermediate α giving balanced behavior (*e.g.* $\alpha = 0.3$: index acc. $\approx 60\%$, semantic acc. $\approx 97\%$). At $\alpha = 1$ the setup reduces to standard classification and individual sample identities are lost; label mixing applies only to the single-branch network, not the two-branch ones.

Inference. With no ground-truth index to condition on at test time, the network outputs a softmax over all training indices, and a class label is read out either as *Top-1* (class of $\arg\max_t \hat{p}_t$) or *Top-5* (majority class among the five most active neurons, analogous to ImageNet top-5 accuracy and more robust to a wrong top-1 index); both are reported in Table 1. There is no separate classification head – class prediction is always read out of the index distribution.

3.2 II: TWO BRANCH NETWORK

We consider two variants. In the *class-independent* variant, the main branch predicts class $y \in \{1, \ldots, C\}$ and the index branch predicts an index $z \in \{1, \ldots, K\}$ over the *entire* dataset; both are trained jointly via cross-entropy, $\mathcal{L}_{\text{total}} = \lambda_{\text{class}}\mathcal{L}_{\text{class}} + \lambda_{\text{index}}\mathcal{L}_{\text{index}}$ (we set $\lambda_{\text{class}} = \lambda_{\text{index}} = 1$). In the *class-conditional* variant, the index branch instead predicts an index $z \in \{1, \ldots, K_y\}$ *within the predicted class* y (where K_y is that class's training-set size), which makes index prediction easier when $K \gg \max_y K_y$: the losses become $\mathcal{L}_{\text{class}} = -\log \hat{y}_y$ and $\mathcal{L}_{\text{index}} = -\log \hat{z}_{z|y}$, again combined via $\lambda_{\text{class}}, \lambda_{\text{index}}$. At inference, the model predicts the class via the main branch first, then restricts index prediction to that class's indices. The index branch has as many neurons as the largest class (shared) or, alternatively, one head per class sized to that class (*i.e.* no parameter sharing).

4 EVALUATION

We first establish core classification and training-set index-prediction performance across five datasets, before turning to this paper's deeper analyses: memorization/generalization (Section 5.2) and test-time attribution (Section 6).

4.1 CLASSIFICATION

Table 1 summarizes class and index prediction accuracy on four coarse-grained datasets (MNIST, FashionMNIST, CIFAR-10/100 (Krizhevsky & Hinton, 2009)) and one fine-grained dataset (Stanford Dogs (Khosla et al., 2011)), comparing two-branch against single-branch networks under two label-mixing levels (dataset stats in Appx. 9.5).

Without heavy optimization (*e.g.* no data augmentation), the class-conditional network classifies and explains well; the class-independent variant does poorly on index prediction for the CIFAR datasets, since 60K training samples require many more output neurons, but remains strong on the lower-complexity MNIST/FashionMNIST – so the network still explains meaningfully in many cases, more so for CIFAR-10 than CIFAR-100. Single-branch results (Figure 2) show stronger memorization (100%) explaining better but classifying worse than weaker memorization (50%): a single branch must compromise between the two, while a two-branch network achieves both, at the cost of a larger, more expensive model.

Comparison with other explainability methods. The final two columns of Table 1 compare against influence functions (Koh & Liang, 2017), approximated (since exact Hessian-based estimation is prohibitively expensive and numerically unstable at this scale (Basu et al., 2020; Feldman, 2020)) via nearest-neighbor retrieval in the last-layer/all-layer representation, following Yeh et al. (2018);

Table 1: Classification and index prediction accuracy across 4 settings: two-branch (class-conditional), two-branch (class-independent), and single-branch networks with two levels of memorization. Idx Acc denotes index prediction accuracy on the *training set* (memorization), not test-time attribution to a novel input; the latter is evaluated separately, against known ground truth, in Section 6. A memorization level of 100% corresponds to a label mixing parameter of $\alpha = 0$. For the single-branch network, class prediction (Cls Acc) is derived either from the class of the most active neuron (Top-1) or from the majority class among the five most active neurons (Top-5). The network first predicts indices, and the labels associated with those indices are then used for classification. In the class-conditional setting, indices vary only within each class, whereas in the class-independent setting, they span the entire dataset. The last two columns show comparison with influence functions approach.

	Two-Branch Net Class Conditional		**Two-Branch Net Class Independent**		**Single-Branch Net 100% Memorization**		**Single-Branch Net 50% Memorization**		**Influence Functions**	
	Cls Acc	**Idx Acc**	**Cls Acc**	**Idx Acc**	**Cls Acc Top-1/5**	**Idx Acc**	**Cls Acc Top-1/5**	**Idx Acc**	**All Layers**	**Last Layer**
MNIST	99.08	98.16	99.41	99.41	84.6/98	100	98.8/99.7	49.6	87.25	99.36
FMNIST	96.01	98.68	92.65	98.63	76.6/94.7	99.8	90.3/96.4	48	76.36	92.08
CIFAR-10	83.16	99.41	75.73	89.86	30.3/68.3	99.7	65.1/86.4	47.3	49.46	77.97
CIFAR-100	37.14	99.2	38.20	40.28	8.0/20.5	94	17.0/37.4	32.7	26.24	37.10
Stanford Dogs	82.58	46.1	65.54	84.45	8.4/17.8	99.5	9.3/22.6	47.9	-	-

Pruthi et al. (2020); the assigned label is that of the most similar training point (Appx. 9.7 for implementation details). This baseline is strong but still falls short of the index head in our two-branch net.

5 Network Analysis

5.1 Memorization vs. Generalization

Figure 2 shows single-branch performance on MNIST (LeCun et al., 1998) as the index mixing ratio α varies from 0–100%: low α favors memorizing individual samples at the cost of test-set generalization, while higher α trades memorization for generalization, reaching pure generalization (no memorization) at $\alpha = 100\%$ – letting the mixing ratio be tuned per task. FashionMNIST (Xiao et al., 2017) shows the same trend, with memorization hurting classification accuracy even more (Appx. 9.4).

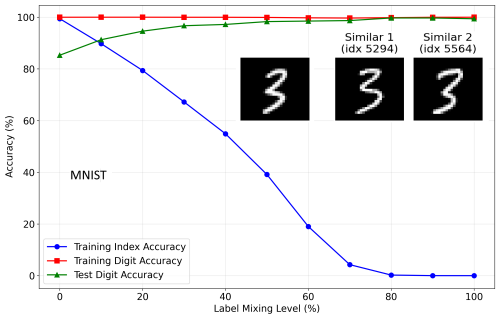


Figure 2: Trade-off between generalization and memorization in the single-branch network on MNIST, with test samples overlaid alongside their two most similar training examples (FashionMNIST counterpart is shown in Appx. 9.4).

5.2 Scalability Analysis

To address the fundamental scalability limitation of provenance networks, we investigate whether the system can operate effectively when trained on only a strategically selected subset of training data, using stratified sampling (by class) to select the subset for both branches of the class-conditional network. Table 2 reports classification (CNN1) and index-retrieval (CNN2 Top-K) accuracy across subset ratios from 10–90% on MNIST and FashionMNIST. Training both branches on just 30% of MNIST (17,995 samples) achieves 98.87% main-branch test accuracy – matching the full-60K model – and 95.49% Top-5 index-retrieval accuracy, while cutting the index head's parameters by 70%. FashionMNIST needs a larger subset (50%, 30K samples) to reach a comparable operating point and lands lower (90.86% class accuracy, 89.71% Top-5), reflecting its greater intra-class visual variation. A subset always helps relative to no subsetting, but how large a subset is needed is dataset-dependent, and Top-1 index accuracy is non-monotonic in subset size (Appx. 9.11 for the full discussion). This suggests a practical deployment strategy: rather than indexing all training samples, provenance networks can focus on representative or high-value exemplars (*e.g.* near decision boundaries, or verified clean data), trading some retrievability for substantial parameter reduction.

5.3 Toward Sub-Linear Scaling: Two Directions

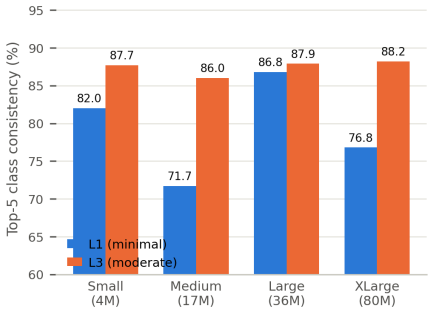
Subset sampling above is a practical mitigation, not a change to the index head's asymptotic complexity: its output layer still has $O(M)$ parameters, where M is the number of indexed samples per class. We explored two directions to address this directly, one tested, one not.

Factorized index head (tested). Representing each index as K digits in a compact base B (with $B^K \geq M$), each predicted by its own small softmax head, reduces the index head's size from $O(M)$

Table 2: Scalability analysis: both classification and index branches trained on the same subset, MNIST and FashionMNIST side by side. CNN1 is test classification accuracy; Top-K is class-matching accuracy of retrieved training samples.

	MNIST				FashionMNIST			
Subset	**CNN1**	**T-1**	**T-5**	**T-10**	**CNN1**	**T-1**	**T-5**	**T-10**
10%	98.27	68.32	92.57	96.56	86.66	59.13	87.20	93.62
30%	98.87	79.72	95.49	97.75	89.99	60.11	89.26	94.93
50%	98.98	69.94	92.05	96.04	90.86	66.88	89.71	94.77
70%	99.19	76.86	95.16	97.76	91.36	67.39	91.26	95.79
90%	99.26	83.26	96.53	98.41	92.19	64.86	90.48	95.26

Figure 3: FashionMNIST index-branch Top-5 class consistency, minimal (L1: first conv layer only) vs. moderate sharing (L3: all conv layers, $\leq$4.2% of parameters).



to $O(K \cdot M^{1/K}) - O(\sqrt{M})$ at K=2 – with digits combined into a full ranking via $\log P(\text{index}) = \sum_k \log P(d_k)$. The parameter reduction is real and grows with scale: 41× at M=6,742, up to 500× at M=1,000,000 (Appx. 9.12, Table 8). But training convergence is substantially harder: at a matched 40-epoch budget the factorized head reaches only 3.0% exact-match accuracy versus 89.8% for the flat head, and even warm-starting from the trained flat model's trunk only reaches a $\sim$50.7% plateau (full convergence sweep in Appx. 9.12, Table 9). The bottleneck is the digit-head readout itself, not the shared representation – a genuine, verified fix to the parameter-scaling problem, but not yet a free one.

Hierarchical indexing. A second direction would restructure index prediction into a coarse-to-fine hierarchy mirroring natural category structure – *e.g.* species → dog, breed → Pomeranian, identity → a specific labeled dog – reducing the branching factor at each level while imposing semantically meaningful intermediate categories that neither the flat nor the digit-factorized index provides. We leave implementing and evaluating this direction to future work.

5.4 Parameter Sharing at Scale

Beyond subset sampling, we tested how much of the two branches' representation can be shared without hurting either task, across four model sizes (4–80M parameters) and four sharing levels (from sharing only the first convolutional layer to sharing all convolutional layers plus the first fully-connected layer). On FashionMNIST, sharing all three convolutional layers ($\leq 4.2\%$ of total parameters) leaves classification accuracy essentially unchanged (± 0.2 points across all four model sizes) but substantially improves the index branch's semantic class-consistency over sharing only the first layer, most dramatically in larger models (Figure 3) – consistent with shared representations encoding more generalizable features than task-specific ones. Minimal sharing can in fact *underperform* moderate sharing, so more sharing is not simply better or worse: it is close to free, and sometimes actively helps. Full results (both datasets, all four sharing levels, training curves) are in Appx. 9.13.

6 Test-Time Attribution Quality

Table 1 reports *training-set* index accuracy: after training, how often does the index branch recover the exact training index it was trained to predict? This measures memorization, not attribution — a network can memorize its training indices perfectly while retrieving an arbitrary, semantically unrelated exemplar for a novel test input, since no training signal ever constrains what happens at test time. This section evaluates attribution quality directly, on held-out inputs, against a known ground-truth source.

6.1 Evaluation protocol

The central difficulty in evaluating test-time attribution is that real test sets have no ground-truth "correct" training exemplar — a test digit was never literally copied from one specific training digit. We construct a semi-synthetic protocol with a known answer: for a random subset of 1,000 training images, we build a *probe* by applying a controlled perturbation (a random rotation, translation, and additive pixel noise, at a chosen strength) to a copy of the image, so the identity of the correct source is known by construction. We then measure whether the index branch's top-1/top-5 retrieved

exemplar(s) match the known source. We report results on MNIST, FashionMNIST, and CIFAR-10 with the class-conditional two-branch network of Section 3.

6.2 BASELINES

We compare against three baselines under the identical protocol above. Two use cosine-nearest-neighbor retrieval, differing only in embedding space: the **jointly-trained kNN-embedding baseline** matches the query against training embeddings from a task-loss-only network with no index branch, isolating what representational similarity alone recovers, while the **influence-function proxy** runs the same retrieval in the two-stage model's own shared (index-aware) embedding, following the last-layer influence-function approximation of Appx. 9.7, isolating what the trained head buys over plain similarity search in a good jointly-trained representation. The **ProtoPNet-style baseline** (Chen et al., 2019) is a self-explainable prototype network with the same convolutional backbone, learning 10 prototype vectors per class *pushed*, after training, to their nearest real training-image patch; the training image a class's most-activated prototype was pushed to is the closest analogue to an "exemplar" this architecture can produce (full details in Appx. 9.10).

6.3 RESULTS

Naive result: the trained index head underperforms the kNN/influence baselines, at least on MNIST. Without augmentation, at the strongest perturbation level the trained index head (Prov.) trails the kNN-embedding baseline on all three datasets: 5.3% vs. 15.5% (MNIST), 5.2% vs. 6.0% (FashionMNIST), 1.7% vs. 0.4% on CIFAR-10 where the margin flips (top-1, class-conditional network trained as in Section 4; full perturbation sweep in Appx. 9.8, Table 5). Two MNIST-only baselines evaluated at this same level (Infl. = influence proxy, PPNet = ProtoPNet-style) score 11.9% and 0.0% top-1 respectively. Despite reaching 96.5% *training-set* index accuracy – comparable to the 98.16% reported in Table 1 – the trained index head is clearly outperformed by the kNN and influence-proxy baselines at test time on MNIST.

ProtoPNet's near-zero score is structural, not incidental. ProtoPNet's attribution accuracy is essentially 0% at every perturbation level despite classification accuracy on par with the provenance net (Table 1): with only 10 prototypes per class summarizing $\sim 6{,}000$ same-class images, the maximum possible top-1 hit rate is $10/6{,}000 \approx 0.167\%$, matching the observed rate almost exactly (Appx. 9.10 for the full analysis).

Diagnosing the gap: a specific geometric-shift brittleness. We swept perturbation strength from identity to the Table 5 level (full sweep, all perturbation levels, in Appx. 9.8) on all three datasets (CIFAR-10 uses the ResNet of Appx. 9.6). On MNIST/FashionMNIST, the index head matches or slightly exceeds kNN under photometric noise alone but collapses once rotation/translation appears, while kNN degrades more gracefully; on CIFAR-10 this does not reproduce – the non-augmented index head beats kNN at every level but identity, plausibly because the ResNet's strided convolutions carry shift tolerance the shallow grayscale-dataset CNN lacks. We verified the modest CIFAR-10 margins across 2 additional seeds: non-overlapping mean$\pm$std intervals at small/medium/full (small: $16.8 \pm 2.4\%$ vs. $11.4 \pm 1.0\%$; medium: $6.3 \pm 0.9\%$ vs. $3.1 \pm 0.4\%$; full: $2.0 \pm 0.6\%$ vs. $0.6 \pm 0.3\%$) confirm a real, seed-robust effect. All three point to one story: without augmentation, shift-invariance is whatever the backbone's inductive bias provides, not something explicitly trained for – a fixable training-recipe gap, not a fundamental limitation.

Fix: augmentation-invariant index training. We retrained the index branch on each dataset with mild random rotation/translation augmentation (10°, 6% translation), target index fixed, so the model must map perturbed views of a sample to the same index – and retrained the kNN-embedding baseline identically for a fair comparison. (FashionMNIST and CIFAR-10 needed substantially more epochs to converge than the clean models – 40 and 60 vs. 15–20 – since augmentation-invariance is a harder optimization problem; we report each once training-index accuracy matched its clean model, 89.8%/87.2% respectively.) Figure 4 shows the same sweep after this fix, on all three datasets.

Once both methods receive identical augmentation, the trained index head decisively outperforms embedding-space kNN at every non-trivial perturbation level, on all three datasets – the regime that matters for attribution, since exact-duplicate retrieval (identity/tiny) is kNN's trivial best case. The effect is largest on CIFAR-10 (70.8% vs. 2.2% at the strongest perturbation; FashionMNIST: 39.7% vs. 18.4%). This advantage over simple similarity retrieval is conditional on training-time

augmentation that makes the index target invariant to the perturbations attribution will be evaluated under – without it, training-set index accuracy is actively misleading about test-time attribution quality, exactly the failure mode this section was designed to catch. (Single seed, reduced epochs relative to Table 1; multi-seed and augmentation-strength sweeps are left to future work.)

We separately tested whether the classification/attribution loss-balancing weight ($\lambda_{\text{class}}/\lambda_{\text{index}}$) can be tuned to trade the two objectives off against each other; in short, a multi-seed sweep found no reliable, predictable control (Appx. 9.9, Table 7).

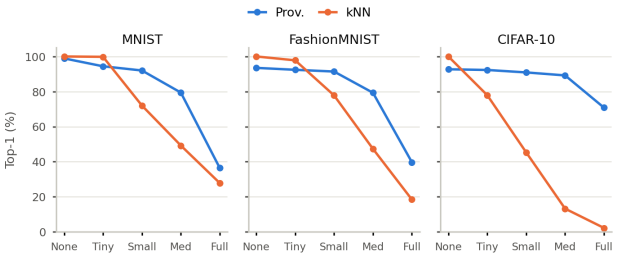


Figure 4: Top-1 retrieval accuracy (%) after augmentation-invariant training, applied identically to both methods (fair comparison); perturbation levels as in Table 5. Prov. = Provenance Net, kNN = kNN-embedding baseline.

Task accuracy is not a good proxy for attribution robustness. If the attribution result above just restated a generic robustness benefit, ordinary classification accuracy should show the same augmentation dependence. Table 6 (full sweep in Appx. 9.8) says no: without augmentation, task accuracy under the full perturbation protocol stays well above its collapsed index-branch counterpart on all three datasets (92.6%/71.5%/58.6% task vs. 5.3%/5.2%/1.7% index, MNIST/FMNIST/CIFAR-10); with augmentation, task accuracy is saturated or stabilized ($\geq$ 99.5%/ $\geq$ 90%/ $\geq$ 82.5%) while moving far less than retrieval accuracy does (Figure 4). Task and attribution accuracy are largely decoupled – a model can be robust on its primary task while attribution is far more brittle to the same input variation, so task accuracy cannot stand in for the attribution-quality evaluation here.

Appx. 9.3 provides qualitative support for this via t-SNE visualization of the same embedding layer, including its behavior on misattributed test samples.

6.4 Behavior under misclassification

In the class-conditional variant, index prediction is restricted to the class predicted by the task branch, so a task-branch error necessarily restricts the index branch's search to the wrong class before it ever runs. This raises a question distinct from overall attribution accuracy: does the index branch simply inherit the task branch's mistake, or can its retrieved exemplar still be informative even when the predicted class is wrong? This matters in practice: a practitioner debugging a model's errors most needs an explanation exactly when the model is wrong, and if the mechanism is only ever useful for correct predictions, its value as a debugging tool is limited.

We condition on the existing MNIST test-set predictions (97/10,000 misclassified, 99.03% test accuracy) and inspect index-branch behavior on exactly the misclassified subset. The answer to whether the index branch can recover from the task branch's mistake is unambiguous by construction rather than empirical: since the class-conditional index branch restricts its search to the *predicted* class, and the true class is by definition excluded whenever the main branch is wrong, the retrieved exemplar's true label matches ground truth 0% of the time on the misclassified subset – the index branch cannot recover from a main-branch error, it can only ever be as right as the class prediction it is conditioned on. Separately, mean index-branch entropy is measurably higher on misclassified inputs than correctly classified ones (3.58 vs. 3.30 nats), suggesting entropy carries some usable signal that a prediction may be unreliable.

Even a wrong-class retrieval can still explain the mistake. Although the retrieved exemplar's *label* is guaranteed wrong on the misclassified subset, its *content* need not be uninformative: if the retrieved image is one a practitioner would recognize as visually confusable with the input, it directly shows *why* the network erred, even though it cannot correct the error. We tested this by comparing, for each of the 97 misclassified test images, the retrieved (wrong-class) exemplar's embedding-space similarity to the input against a baseline of 20 random training images from that same wrong class. If the retrieval carries no diagnostic value beyond being an arbitrary member of the wrong class, these two similarities should be indistinguishable. They are not: mean similarity to the actual retrieved exemplar is 0.381, versus 0.192 for the random-baseline exemplars, with

the retrieved exemplar more similar than the random baseline on 67.0% of misclassified inputs (paired t-test, $t = 5.38$, $p = 5.2 \times 10^{-7}$). This is not universal – roughly a third of misclassified inputs retrieve an exemplar no more informative than a random one, and we found at least one case where the retrieved exemplar was *less* similar to the input than the random baseline – but on the whole, and with high statistical confidence, the retrieved exemplar is a meaningfully better guide to the model's confusion than an arbitrary wrong-class example would be. So while the index branch cannot correct a misclassification, inspecting what it retrieves is, on average, still a genuinely informative debugging signal precisely when a prediction is wrong – not merely a wrong-class placeholder.

7 Additional Applications

We revisit three further, more exploratory applications of the provenance mechanism, each validated to a lesser degree than the core results of Section 6: robustness to image distortions, membership inference, and generative modeling. We separately tested, and rejected, a stronger and more specific hypothesis in this vein – that the provenance objective improves *adversarial* robustness – reported in Appx. 9.14.1. Extended results and implementation details for all three applications below are in Appx. 9.14–9.16.

7.1 Robustness to Image Distortions

We report this as a secondary, exploratory finding, and an instructive contrast with Appx. 9.14.1: comparing a single-branch index-prediction network at varying memorization levels against an independently-trained "isolated CNN" under incidental distortions (salt-and-pepper/Gaussian noise, occlusion, translation, rotation, scaling, blur), intermediate memorization (∼20–30%) appears to improve robustness for some distortions (occlusion, blur). We have not isolated why, and when we tested the more specific and consequential version of this hypothesis directly – adversarial rather than incidental robustness – the effect reverses (Appx. 9.14.1). Full curves are in Appx. 9.14.

7.2 Membership Inference

We report this as a secondary, exploratory finding: the index branch's confidence separates training members from non-members with high AUC, consistent with it overfitting to training indices as expected, but we have not compared against dedicated membership-inference attacks from that literature. We trained the class-conditional two-branch network on four datasets (MNIST, FashionMNIST, CIFAR-10, CIFAR-100) for 40 epochs, reaching near-perfect index-prediction accuracy (*i.e.* overfitting to training indices while maintaining high classification accuracy in the main branch), then sampled 5K training (member) and 5K test (non-member) instances per dataset and computed ROC curves from the maximum softmax confidence of both branches.

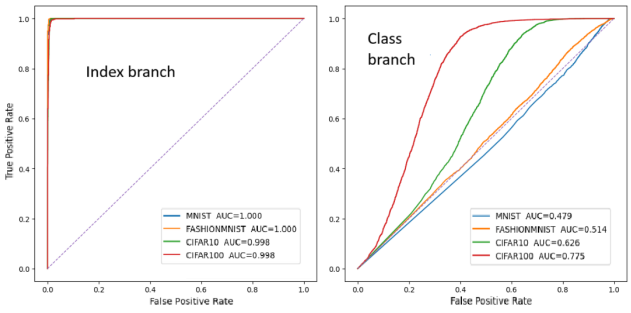


Figure 5: Left: Membership inference results based on the distribution of maximum confidence from the index branch of a class-conditional two-branch network. Right: Corresponding results using the class branch. See Appx. 9.15.

Figure 5 shows results across all four datasets. Using the index branch, AUC is consistently close to perfect. This is because a memorized sample typically activates only one (or a very small subset of) neuron(s), whereas a non-member tends to activate multiple neurons, resulting in lower confidence. Using the class branch instead, the smaller number of output neurons reduces separability, giving lower AUC. Similar trends hold using entropy instead of confidence (Appx. 9.15). This is the flip side of the paper's central mechanism: the same per-sample memorization signal that makes an index branch attributable at all is, almost by construction, exactly the signal a membership-inference attack looks for – so any deployment of provenance networks should weigh this exposure alongside the data-availability constraint already noted in Limitations.

7.3 IMAGE GENERATION

We report this as a secondary, exploratory finding: extending provenance to a generative (VAE) setting is architecturally straightforward and the retrieved exemplars visually resemble generated samples, but we have not evaluated generation quality (*e.g.* via FID) or attribution quality under the protocol of Section 6.1 for this setting. The model is a Variational Autoencoder (VAE) (Kingma & Welling, 2014) with two auxiliary supervised heads: on an intermediate decoder feature, a class branch predicts y_i and an index branch predicts k_i, the sample index within the class (one head per class, not parameter-shared). Training minimizes a weighted combination of generative and discriminative objectives:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{gen}}\big(\mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}}\big) + \lambda_{\text{cls}}\big(\mathcal{L}_{\text{class}} + \mathcal{L}_{\text{index}}\big), \tag{1}$$

with $\lambda_{\text{gen}} = 0.6$, $\lambda_{\text{cls}} = 0.4$ (Adam, learning rate 10^{-3}, 70 epochs). We first validate this on MNIST and FashionMNIST (latent dimension 128), where generated samples closely resemble their retrieved nearest training exemplars (Appx. 9.16) – confirming the mechanism transfers to a generative objective before testing whether it survives a substantially harder, higher-dimensional setting. To test whether it holds at larger scale, we apply the same architecture to the LFW face dataset (Huang et al., 2008) (13,233 images, 5,749 identities), with one index-prediction head per identity.

Figure 6 shows the model attains high classification/index accuracy even at this scale; even without extensive hyperparameter tuning, additional loss terms (*e.g.* perceptual loss by Johnson et al. (2016)), or exhaustive optimization, the generated faces exhibit reasonable fidelity and resemble their retrieved training samples. We also observe, qualitatively, that faces the index branch retrieves with higher confidence tend to decode more cleanly (architecture details in Appx. 9.16.1) – suggestive of index-prediction confidence as a free, unsupervised proxy for generation quality that requires no separate metric like FID, though we have not verified this quantitatively and report it only as a qualitative observation.

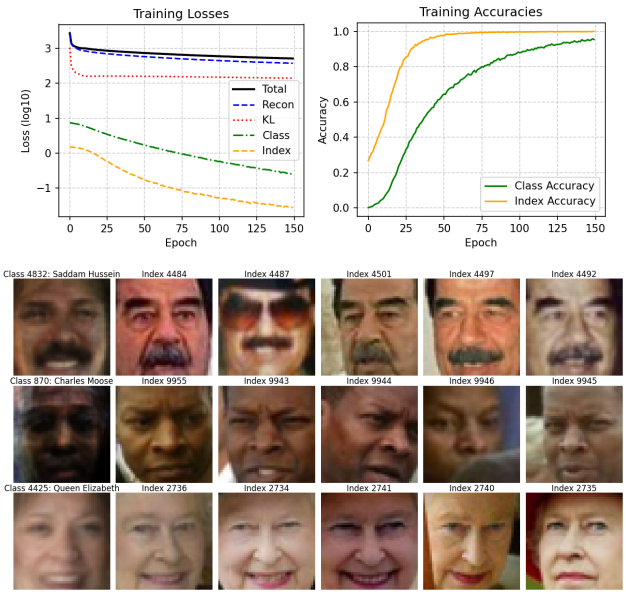


Figure 6: Image Generation with VAE on LFW faces. Top: training losses and class/index accuracies per epoch. Bottom: a generated face alongside its top-5 retrieved training exemplars from the index branch. Full architecture in Appx. 9.16.1; MNIST/FashionMNIST results in Appx. 9.16.

8 CONCLUSIONS

Provenance networks learn a representation that separates classes while also distinguishing individual samples, giving transparency into model decisions – but only once that transparency is tested against ground truth rather than assumed from training-set accuracy alone. Test-time attribution quality is real, but only once the index branch is trained with augmentation matched to the perturbations it will be evaluated under (Figure 4).

Limitations. Scalability: index-head accuracy drops as training data grows; subset selection (Section 5.2) and a factorized head (Section 5.3) both help but neither fully solves it. Deployment: the training data must be available and queryable at inference time, limiting use to settings where the deploying party already controls that data. Tunability: the classification/attribution loss-balancing weight is not a reliable dial (Table 7).

Additional findings. Section 7 covers three further exploratory applications (image-distortion robustness, membership inference, image generation), none validated to the standard of Section 6. We also directly tested, and reject, the stronger hypothesis that the provenance objective improves *adversarial* robustness (Appx. 9.14.1): if anything it reduces it, though branch disagreement gives a modest detection signal (AUROC up to 0.854).

REPRODUCIBILITY STATEMENT

We report architectural, training, and evaluation details throughout the paper and its appendix so that all results can be reproduced from the descriptions given. The single- and two-branch provenance-network architectures, including layer-by-layer specifications and optimizer settings, are detailed in Appx. 9.2 (and the corresponding single-stage subsection); backbone choices per dataset are given in Appx. 9.6; dataset statistics, splits, and preprocessing are in Appx. 9.5; the VAE architecture used for the image-generation application is in Appx. 9.16.1; and the training configuration for the membership-inference experiments is given in Appx. 9.15 (Table 13). The test-time attribution-quality evaluation protocol (perturbation types and strengths) is fully specified in Section 6.1. All datasets used (MNIST, FashionMNIST, CIFAR-10/100, Stanford Dogs, LFW) are standard, publicly available benchmarks. We are explicit throughout the paper about which results are single-run versus averaged over multiple seeds (*e.g.* the loss-balancing sweep, Appx. 9.9, and the CIFAR-10 seed-robustness check in Section 6), flagging single-seed results as such rather than presenting them with unwarranted precision. Code to reproduce the architectures, training recipes, and evaluation protocols described here will be released publicly upon publication.

AI USE STATEMENT

A large language model (Claude, Anthropic) was used as a writing and editing assistant during the preparation of this manuscript: restructuring sections, condensing prose, reorganizing content between the main text and appendix, and formatting figures, tables, and LaTeX source. It was not used to design experiments, generate or analyze data, produce the reported numerical results, or originate scientific claims; all experimental results, figures, and findings reported in this paper are the authors' own work. The authors have reviewed and take full responsibility for all content in the paper.

REFERENCES

Samyadeep Basu, Phillip Pope, and Soheil Feizi. On the causality and stability of influence functions. *arXiv preprint arXiv:2009.00329*, 2020.

Chaofan Chen, Oscar Li, Chaofan Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Vitaly Feldman. What neural networks memorize and why: Discovering the long tail via influence estimation. *arXiv preprint arXiv:2008.03703*, 2020.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 2242–2251, 2019.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Zayd Hammoudeh, Anant Mangalik, Han Zhang, Zhiwei Steven Yang, Shengjia Zhao, and Chelsea Finn. Training data influence analysis and estimation: A survey. *arXiv preprint arXiv:2212.04612*, 2023.

Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*, pp. 694–711. Springer, 2016. doi: 10.1007/978-3-319-46475-6_43.

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 2–1. IEEE, 2011.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 1885–1894, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. In *Proceedings of the ICML 2024 Workshop on Mechanistic Interpretability*, 2024.

Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2020.

Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Bhaskar Mitra and Nick Craswell. An introduction to neural information retrieval. *Foundations and Trends in Information Retrieval*, 13(1):1–126, 2018.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2023.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(1):e7, 2017. doi: 10.23915/distill.00007. URL `https://distill.pub/2017/feature-visualization`.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems*, pp. 19920–19930, 2020.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.

Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR) Workshop*, 2014.

Daniel Smilkov, Boris Hallen, and Fernanda Viégas. Smoothgrad: removing noise by adding noise. In *ICML Workshop on Visualization for Deep Learning*, 2017.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 3319–3328. PMLR, 2017.

Oriol Vinyals, Charles Blundell, Tim Lillicrap, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Jun Xu, Xiangnan He, and Hang Li. Deep learning for matching in search and recommendation. *Foundations and Trends in Information Retrieval*, 13(1):1–126, 2018.

Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep Ravikumar. Representer point selection for explaining deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 9291–9301, 2018.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE transactions on emerging topics in computational intelligence*, 5(5):726–742, 2021.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

9 APPENDIX

This appendix contains architectural detail, extended results for the core sections, and additional applications (robustness, membership inference, image generation) that we report as secondary, exploratory findings rather than core claims of the paper (see Section 8).

9.1 SINGLE-STAGE STANDALONE ARCHITECTURE

The standalone (*i.e.* single branch) architecture directly maps features to training sample indices ($\mathbb{R}^m \to \mathbb{R}^N$) without intermediate class structure, representing pure memorization where the model must learn to distinguish between all N training samples simultaneously (Figure 7).

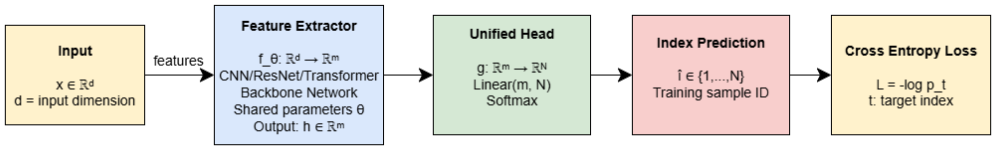


Figure 7: Single-Stage Standalone Architecture for Direct Provenance/Attribution.

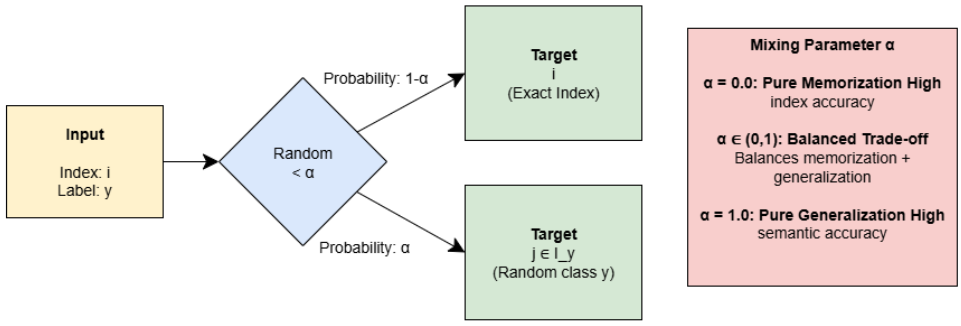


Figure 8: Training Strategy with Mixing Parameter α. During training, samples are probabilistically assigned to either exact index targets (memorization) or random class targets (generalization), controlled by mixing parameter $\alpha \in [0,1]$.

The standalone network consists of three primary components: feature extraction, unified attribution head, and direct index prediction, as illustrated in Figure 7.

Feature Extraction Backbone: The feature extractor employs a CNN architecture optimized for large-scale memorization tasks:

- **First Convolutional Block:** Conv2d$(1, 128, 3)$ with padding, BatchNorm2d, ReLU activation, and MaxPool2d(2) reducing spatial dimensions to 14×14
- **Second Convolutional Block:** Conv2d$(128, 256, 3)$ with padding, BatchNorm2d, ReLU activation, and MaxPool2d(2) reducing to 7×7
- **Third Convolutional Block:** Conv2d$(256, 512, 3)$ with padding, BatchNorm2d, ReLU activation, and AdaptiveAvgPool2d$(4, 4)$ producing fixed 4×4 spatial output

This configuration yields feature representations $h \in \mathbb{R}^{8192}$ where $8192 = 512 \times 4 \times 4$.

Unified Attribution Head: The classification head performs direct mapping from features to training sample probabilities through a deep fully-connected network:

$$h_1 = \text{ReLU}(\text{BN}(\text{Linear}(h, 4096))) \quad \text{with Dropout}(0.4) \tag{2}$$
$$h_2 = \text{ReLU}(\text{BN}(\text{Linear}(h_1, 2048))) \quad \text{with Dropout}(0.2) \tag{3}$$
$$\hat{y} = \text{Softmax}(\text{Linear}(h_2, N)) \quad \text{with Dropout}(0.1) \tag{4}$$

where $N = 60{,}000$ represents the total number of training samples, and $\hat{y} \in \mathbb{R}^N$ is the probability distribution over all training indices.

Model Capacity: The complete architecture contains approximately 129 million trainable parameters, with the final attribution layer contributing $2048 \times 60{,}000 = 122{,}880{,}000$ parameters alone, emphasizing the model's capacity for fine-grained memorization.

9.1.1 Training Objective and Loss Function

The training objective directly optimizes for exact training sample identification. For each input sample (x_i, y_i) with corresponding training index t_i, the model learns the mapping:

$$f_\theta : x_i \mapsto t_i \tag{5}$$

We employ cross-entropy loss with label smoothing ($\epsilon = 0.05$) to stabilize training on the large output space of $N = 60{,}000$ training samples.

9.1.2 Optimization Strategy

Optimizer Configuration: We employ AdamW optimizer with the following hyperparameters:

- Learning rate: $\eta = 0.002$
- Weight decay: $\lambda = 2 \times 10^{-5}$
- Momentum parameters: $\beta_1 = 0.9, \beta_2 = 0.999$
- Batch size: $B = 128$

Learning Rate Scheduling: We implement a warmup followed by step decay schedule. The learning rate gradually increases from zero to the base rate over the first 3 epochs. After warmup, we apply step decay every 8 epochs with a multiplicative factor of 0.6, allowing the model to converge effectively.

Weight Initialization: Critical for large-scale memorization, we use:

- Convolutional layers: Kaiming normal initialization with mode = fan_out
- Batch normalization: weights = 1, bias = 0
- Final attribution layer: $\mathcal{N}(0, 0.01)$ for enhanced stability
- Other linear layers: $\mathcal{N}(0, 0.02)$

9.1.3 Experimental Setup and Evaluation Metrics

We monitor two complementary accuracy metrics during training:

Index Accuracy: Measures exact memorization capability **Digit Accuracy:** Measures semantic understanding

Test Evaluation: For test samples not present during training, we evaluate both Top-1 and Top-5 digit accuracy.

Training conducted on A100 NVIDIA GPUs with mixed precision (FP16) using PyTorch, enabling efficient memory utilization for the large output space ($N = 60{,}000$). Models trained for 80 epochs per experiment with comprehensive monitoring of memorization dynamics, convergence patterns, and generalization behavior across different mixing ratios.

Memorization Experiments: We conduct comparative analysis between:

- **100% Memorization** ($\alpha = 0.0$): Pure index-level learning
- **50% Memorization** ($\alpha = 0.5$): Balanced memorization-generalization

This experimental design enables systematic investigation of the memorization-generalization trade-off in neural attribution networks and provides insights into the model's capacity for fine-grained training sample identification versus semantic feature learning.

9.2 TWO-STAGE PROVENANCE NETWORK

The two-stage provenance network addresses the computational challenges of large-scale attribution by decomposing the problem into hierarchical stages: digit classification followed by instance-level attribution within the predicted class. This approach significantly reduces parameter complexity while enabling conditional attribution based on semantic class structure.

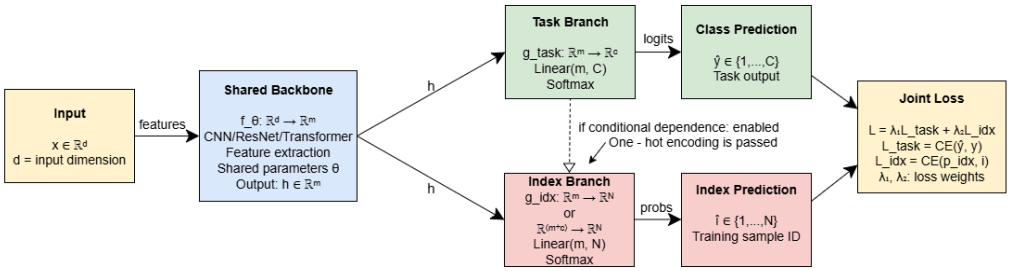


Figure 9: Two-Stage Provenance Network with Optional Class Conditional Dependence. The shared backbone extracts features, which feed into both the task branch (digit classification) and index branch (training sample attribution). When conditional dependence is enabled, the index branch receives concatenated features and one-hot encoded class predictions, allowing instance-level attribution within the predicted class. When disabled, the index branch operates on features alone, performing attribution across all training samples without class-specific guidance

9.2.1 ARCHITECTURE OVERVIEW

The two-stage architecture consists of a shared feature extraction backbone feeding into two specialized branches: the task branch for digit classification and the index branch for training sample attribution, as illustrated in Figure 9. The key innovation lies in the conditional dependence mechanism that allows the index branch to leverage class predictions for more focused attribution.

Shared Feature Backbone: The feature extraction employs a lightweight CNN architecture:

- **First Block:** Conv2d$(1, 64, 3)$ with padding, BatchNorm2d, ReLU, MaxPool2d$(2) \rightarrow 14 \times 14 \times 64$
- **Second Block:** Conv2d$(64, 128, 3)$ with padding, BatchNorm2d, ReLU, MaxPool2d$(2) \rightarrow 7 \times 7 \times 128$
- **Third Block:** Conv2d$(128, 256, 3)$ with padding, BatchNorm2d, ReLU, AdaptiveAvgPool2d$(4, 4) \rightarrow 4 \times 4 \times 256$

The shared backbone produces feature representations $h \in \mathbb{R}^{4096}$ where $4096 = 256 \times 4 \times 4$, which are then projected to $h' \in \mathbb{R}^{2048}$ through a feature projection layer with BatchNorm and Dropout(0.3).

Task Branch (Stage 1): The digit classification branch performs standard 10-class classification:

$$h_{\text{task}} = \text{ReLU}(\text{BN}(\text{Linear}(h', 512))) \quad \text{with Dropout}(0.2) \tag{6}$$

$$\hat{y}_{\text{digit}} = \text{Softmax}(\text{Linear}(h_{\text{task}}, 10)) \tag{7}$$

where $\hat{y}_{\text{digit}} \in \mathbb{R}^{10}$ represents the digit class probability distribution.

Index Branch (Stage 2): The instance attribution branch operates conditionally based on the predicted digit class. The branch architecture depends on whether conditional dependence is enabled:

Without Conditional Dependence:

$$h_{\text{idx}} = \text{ReLU}(\text{BN}(\text{Linear}(h', 2048))) \quad \text{with Dropout}(0.2) \tag{8}$$

$$h'_{\text{idx}} = \text{ReLU}(\text{BN}(\text{Linear}(h_{\text{idx}}, 1024))) \quad \text{with Dropout}(0.1) \tag{9}$$

$$\hat{y}_{\text{idx}} = \text{Softmax}(\text{Linear}(h'_{\text{idx}}, M)) \tag{10}$$

where M is the maximum number of samples per class across all digit classes.

With Conditional Dependence:

$$h_{\text{concat}} = \text{Concat}(h', \text{OneHot}(\arg\max(\hat{y}_{\text{digit}}))) \tag{11}$$

$$h_{\text{idx}} = \text{ReLU}(\text{BN}(\text{Linear}(h_{\text{concat}}, 2048))) \quad \text{with Dropout}(0.2) \tag{12}$$

$$h'_{\text{idx}} = \text{ReLU}(\text{BN}(\text{Linear}(h_{\text{idx}}, 1024))) \quad \text{with Dropout}(0.1) \tag{13}$$

$$\hat{y}_{\text{idx}} = \text{Softmax}(\text{Linear}(h'_{\text{idx}}, M)) \tag{14}$$

The concatenated input $h_{\text{concat}} \in \mathbb{R}^{2058}$ combines the projected features (2048) with the one-hot encoded predicted digit class (10), enabling class-conditioned attribution.

Model Capacity: The two-stage architecture contains approximately 8.7 million parameters, representing a 93.3% reduction compared to the standalone 60K-output model. The parameter distribution includes shared backbone (1.2M), task branch (0.3M), and index branch (7.2M) parameters.

9.2.2 Training Objective and Multi-Task Loss

The training objective combines digit classification and instance attribution through a weighted multi-task loss function:

$$\mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}_{\text{task}} + \beta \cdot \mathcal{L}_{\text{idx}} \tag{15}$$

where $\alpha = 0.3$ and $\beta = 0.7$ balance the contribution of each task.

Task Branch Loss: Standard cross-entropy for digit classification
Index Branch Loss: Cross-entropy with label smoothing ($\epsilon = 0.05$) for instance attribution:

9.2.3 Class-Conditioned Attribution Mechanism

Index Mapping Strategy: The two-stage approach requires bidirectional mapping between global training indices and class-local indices:

$$\text{global_to_local} : \{0, 1, ..., N-1\} \to \{0, 1, ..., 9\} \times \{0, 1, ..., M_c - 1\} \tag{16}$$

$$\text{local_to_global} : \{0, 1, ..., 9\} \times \{0, 1, ..., M_c - 1\} \to \{0, 1, ..., N-1\} \tag{17}$$

where M_c is the number of samples in digit class c, and $M = \max_c M_c$.

Validity Masking: During inference, the index branch output is masked to prevent invalid predictions:

$$\hat{y}_{\text{idx}}^{\text{masked}}[j] = \begin{cases} \hat{y}_{\text{idx}}[j] & \text{if } j < M_{\hat{c}} \\ -\infty & \text{otherwise} \end{cases} \tag{18}$$

where $\hat{c} = \arg\max(\hat{y}_{\text{digit}})$ is the predicted digit class and $M_{\hat{c}}$ is the number of training samples in that class.

Teacher Forcing: During training, we employ teacher forcing where the index branch uses ground truth digit labels rather than predictions:

$$h_{\text{concat}}^{\text{train}} = \text{Concat}(h', \text{OneHot}(y_i^{\text{digit}})) \tag{19}$$

This stabilizes training by providing accurate class information to the attribution branch.

9.2.4 CONDITIONAL VS. NON-CONDITIONAL MODES

The architecture supports two operational modes:

Non-Conditional Mode: The index branch operates independently of class predictions, performing attribution across all training samples without class-specific guidance. Input dimensionality to the index branch remains 2048.

Conditional Mode: The index branch receives concatenated features and class information, enabling class-conditioned attribution. Input dimensionality increases to 2058, allowing the model to focus attribution within the predicted semantic class.

The conditional dependence mechanism provides several advantages:

- **Focused Attribution:** Restricts search space to semantically relevant training samples
- **Improved Accuracy:** Leverages class structure for more precise instance matching
- **Computational Efficiency:** Reduces effective output space from N to $\max_c M_c$
- **Interpretability:** Attribution results are constrained to the predicted semantic class

9.2.5 OPTIMIZATION STRATEGY

Optimizer Configuration: AdamW with identical hyperparameters to the standalone model described in Section 9.1.2.

9.2.6 EVALUATION METRICS

The two-stage architecture requires specialized evaluation metrics for each stage:

Stage 1 (Digit Accuracy): Standard classification accuracy **Stage 2 (Instance Accuracy):** Local index prediction accuracy within the ground truth class **End-to-End Attribution Accuracy:** Overall system performance combining both stages

9.2.7 EXPERIMENTAL CONFIGURATION

Architecture Variants: We evaluate both conditional and non-conditional modes to assess the impact of class-guided attribution on overall system performance.

Computational Efficiency: The two-stage approach enables efficient batch processing with masked outputs, avoiding the computational overhead of the full 60K-dimensional softmax in the standalone architecture.

Training Paradigm: Joint end-to-end training of both branches with shared backbone parameters, enabling the model to learn complementary representations for classification and attribution tasks simultaneously.

This hierarchical decomposition enables scalable attribution learning while maintaining semantic coherence through class-conditioned instance matching, providing a computationally efficient alternative to direct large-scale memorization approaches.

9.3 Analysis of learned embeddings

Figure 10 displays a t-SNE visualization of k-means clusters generated from the penultimate layer of the index branch of a two-branch, class-conditional network. The architecture is detailed in Section 9.2. The analysis was conducted on 5,842 training samples of the digit '4'. For each cluster center, the four closest training data points, selected based on Euclidean distance in the feature space, are shown to illustrate the cluster's composition.

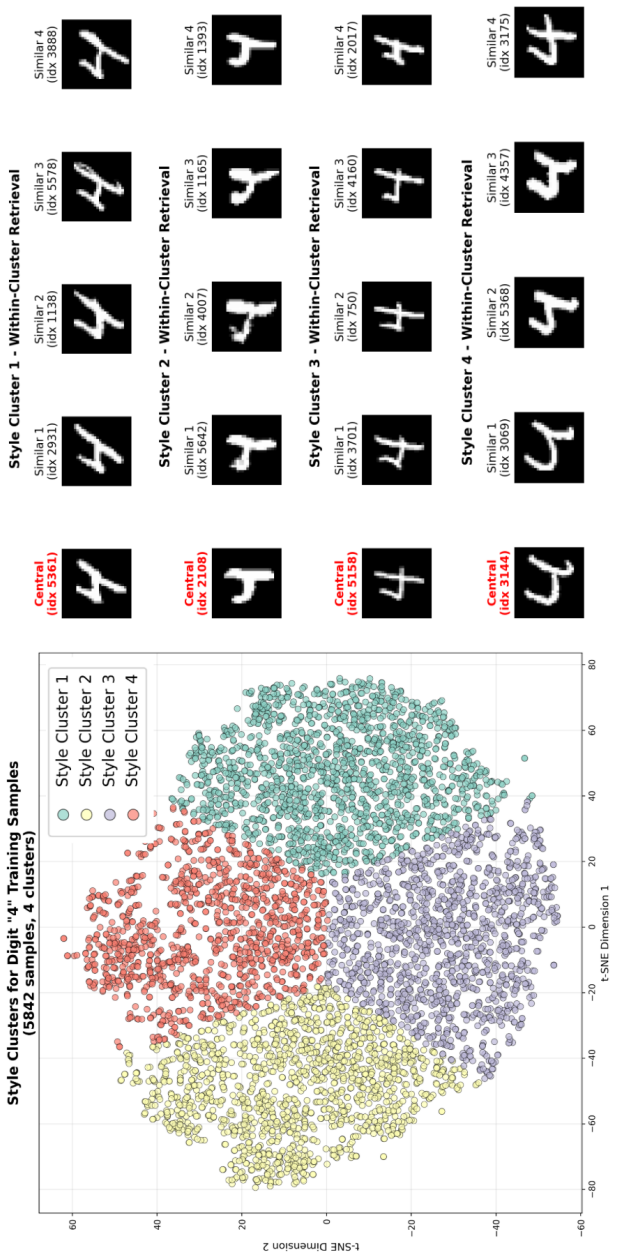


Figure 10: Different styles for digit 4 derived from K-means clustering of penultimate layer of the index branch of a two-branch class-conditional network.

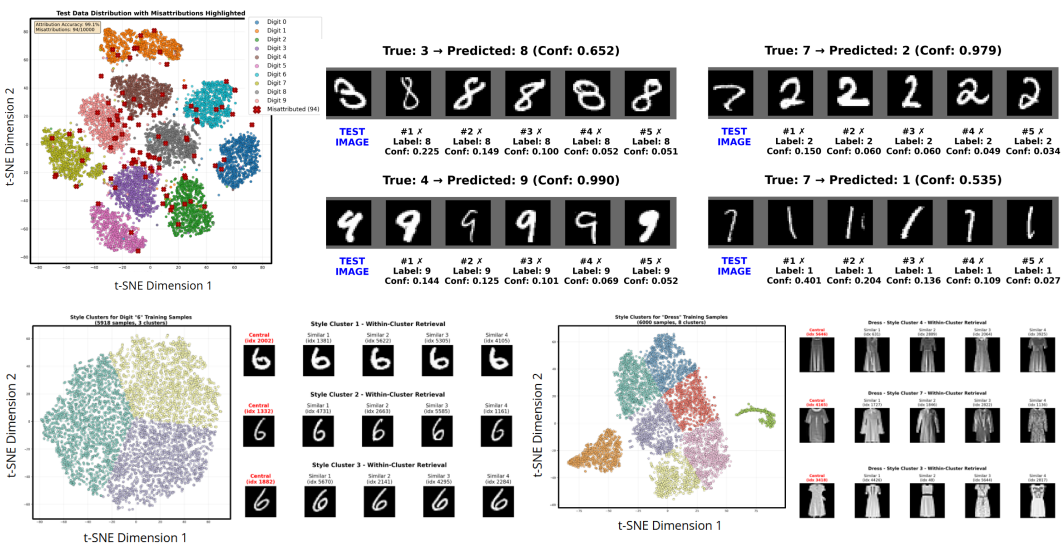


Figure 11: Top (left): t-SNE visualization of the penultimate layer in the index branch of a two-branch class-conditional network. Top (right): Misattributed test samples alongside their five nearest training samples in the index branch. Bottom: t-SNE visualization of k-means clusters from the same layer, with corresponding training samples for digits 6 (left) and FashionMNIST dresses (right).

9.3.1 Qualitative support: representation structure

We analyze representations learned in the 2048D last embedding layer of the index branch. Figure 11 (top left) shows t-SNE visualizations where misattributed samples (94 out of 10K) lie near cluster boundaries—these samples are visually ambiguous, resembling multiple digit classes. The top-5 retrieved training samples confirm the network organizes data by visual similarity rather than ground truth labels. The bottom row demonstrates instance-level structure: k-means clustering reveals distinct writing styles within digit 6 and dress styles within FashionMNIST, showing the embedding captures fine-grained intra-class variation beyond simple class separation.

9.4 Memorization vs. Generalization Trade-off: FashionMNIST

Figure 12 reports the FashionMNIST counterpart of the MNIST single-branch memorization/generalization sweep of Section 5.1 (Figure 2): index prediction accuracy on the training set, and class prediction accuracy (derived from the retrieved index) on both training and test sets, as the index mixing ratio α varies from 0–100%. The trend matches MNIST – low α favors memorization at the cost of test-set generalization, while higher α trades memorization for generalization – but memorization hurts classification accuracy more on FashionMNIST, consistent with it being the harder of the two datasets.

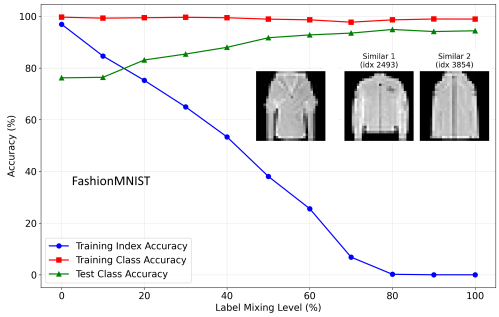


Figure 12: Trade-off between generalization and memorization in the single-branch network on FashionMNIST, with test samples overlaid alongside their two most similar training examples.

9.5 Datasets

We evaluate provenance networks across a diverse collection of datasets spanning different visual domains, complexity levels, and dataset sizes. Our experimental design progresses from simple grayscale digit recognition to complex natural image classification, enabling systematic analysis of how provenance networks scale across different visual domains and dataset complexities.

9.5.1 Computer Vision Datasets

MNIST (LeCun et al., 1998): The Modified National Institute of Standards and Technology database contains 70,000 grayscale images of handwritten digits (0-9) at 28×28 pixel resolution. We use the standard split of 60,000 training samples and 10,000 test samples. Each digit class contains approximately 6,000 training examples, with slight variations across classes. The dataset serves as our primary testbed for fundamental provenance network analysis due to its manageable size and clear class structure.

Fashion-MNIST (Xiao et al., 2017): A direct replacement for MNIST consisting of 70,000 grayscale images of fashion items across 10 categories (T-shirts, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, ankle boots). The dataset maintains the same 28×28 resolution and 60,000/10,000 train/test split as MNIST but presents significantly higher visual complexity with greater intra-class variation and inter-class similarity, making it more challenging for both classification and attribution tasks.

CIFAR-10 (Krizhevsky & Hinton, 2009): A collection of 60,000 32×32 color images across 10 object classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). The standard split provides 50,000 training images and 10,000 test images, with 5,000 training samples per class. CIFAR-10 represents a significant complexity increase from the grayscale datasets, featuring natural images with complex backgrounds, lighting variations, and object poses.

CIFAR-100 (Krizhevsky & Hinton, 2009): An extension of CIFAR-10 containing 60,000 32×32 color images across 100 fine-grained classes grouped into 20 coarse categories. With only 500 training samples per class, CIFAR-100 presents substantial challenges for memorization-based approaches while testing the scalability of provenance networks to larger class vocabularies and reduced per-class sample sizes.

Stanford Dogs (Khosla et al., 2011): A fine-grained classification dataset containing approximately 20,580 images across 120 dog breeds. Images vary significantly in resolution and aspect ratio,

presenting challenges in both visual complexity and fine-grained discrimination. The dataset tests provenance networks' ability to handle real-world image variation and subtle inter-class differences that require detailed visual understanding.

Labeled Faces in the Wild (LFW) (Huang et al., 2008): A face recognition dataset containing over 13,000 images of faces collected from the web, with significant variation in pose, lighting, expression, and image quality. We use LFW to evaluate provenance networks in generative modeling tasks, specifically testing whether generated faces can be traced back to their most similar training examples.

9.5.2 Dataset Statistics and Characteristics

Table 3 summarizes the key characteristics of each dataset used in our experiments. The progression from MNIST to Stanford Dogs represents increasing visual complexity, class granularity, and real-world applicability.

Table 3: Dataset statistics and characteristics for provenance network evaluation

Dataset	Classes	Train Size	Test Size	Resolution	Channels	Complexity
MNIST	10	60,000	10,000	28×28	1	Simple
Fashion-MNIST	10	60,000	10,000	28×28	1	Moderate
CIFAR-10	10	50,000	10,000	32×32	3	Moderate
CIFAR-100	100	50,000	10,000	32×32	3	High
Stanford Dogs	120	12,000	8,580	Variable	3	High
LFW	5,749	13,000	Variable	Variable	3	High

Complexity Considerations: The datasets are strategically selected to evaluate different aspects of provenance networks:

- **Scale Testing:** MNIST and Fashion-MNIST provide controlled environments for fundamental algorithm development with manageable computational requirements.
- **Class Granularity:** The progression from 10 classes (MNIST, Fashion-MNIST, CIFAR-10) to 100+ classes (CIFAR-100, Stanford Dogs) tests scalability of both standalone and two-stage architectures.
- **Visual Complexity:** Moving from grayscale digits to natural color images evaluates the robustness of learned representations across visual domains.
- **Sample Density:** CIFAR-100's 500 samples per class versus MNIST's 6,000 samples per class tests performance under varying data availability.
- **Fine-grained Recognition:** Stanford Dogs' subtle inter-class differences challenge the attribution system's ability to capture discriminative features.

9.5.3 Data Preprocessing and Normalization

All datasets undergo consistent preprocessing to ensure fair comparison across architectures:

Normalization: Images are normalized using dataset-specific statistics:

- MNIST/Fashion-MNIST: $\mu = 0.1307, \sigma = 0.3081$
- CIFAR-10/100: $\mu = (0.4914, 0.4822, 0.4465), \sigma = (0.2023, 0.1994, 0.2010)$
- Stanford Dogs/LFW: ImageNet statistics for transfer learning compatibility

Data Augmentation: We deliberately avoid extensive data augmentation in our primary experiments to maintain direct correspondence between augmented samples and their training indices. This design choice preserves the integrity of the attribution task, where each training sample must maintain a unique, identifiable index.

Resolution Handling: For datasets with variable resolutions (Stanford Dogs, LFW), images are resized to consistent dimensions while maintaining aspect ratios through center cropping or padding as appropriate.

9.5.4 EXPERIMENTAL PARTITIONS

Training Set Attribution: During training, each sample in the training set is assigned a unique index $i \in \{0, 1, ..., N-1\}$ where N is the total number of training samples. These indices remain constant throughout training, enabling the provenance network to learn stable index-to-sample mappings.

Validation and Testing: Test sets are used exclusively for evaluation, with provenance networks tasked to identify the most similar training samples for each test input. This setup simulates real-world scenarios where models must trace novel inputs back to their training data influences.

Cross-Dataset Generalization: While our primary focus is within-dataset attribution, the diverse dataset collection enables analysis of how provenance learning principles transfer across visual domains with different statistical properties and semantic structures.

This comprehensive dataset collection enables systematic evaluation of provenance networks across the spectrum from simple digit recognition to complex real-world visual understanding, providing robust evidence for the approach's broad applicability and scalability characteristics.

9.6 BACKBONE ARCHITECTURES PER DATASET

Appx. 9.1 and 9.2 fully specify the CNN backbone used for the grayscale (MNIST/FashionMNIST) experiments. Table 4 makes explicit which backbone was used for every dataset in Table 1. The CIFAR-10 backbone is directly corroborated by an independent reproduction in Section 6; the exact CIFAR-100 and Stanford Dogs backbones were not independently re-verified for this draft, and we report that as an open documentation gap (rows marked accordingly) rather than assert an architecture we have not confirmed.

Table 4: Backbone architecture used for each dataset's entry in Table 1.

Dataset	**Backbone**	**Status**
MNIST	3-block CNN (Appx. 9.1/9.2), 1-channel input	Confirmed
FashionMNIST	Same backbone as MNIST (1-channel input)	Confirmed
CIFAR-10	10-block ResNet (`dependence_1_cifar10_better.py`)	Plausible: reproduced in Sections 6/6.1 at 81.6
CIFAR-100	*Presumed same family as CIFAR-10*	*Not independently verified*
Stanford Dogs	*Either a custom CNN or an ImageNet-pretrained backbone*	*Not independently verified*
LFW (generation only)	3-layer fully-connected VAE encoder/decoder (Appx. 9.16.1)	Confirmed

9.7 Influence Estimation via Last-Layer Representations

Understanding which training examples are responsible for a model's prediction has become a central problem in interpretability. Classical influence functions estimate the effect of removing a training point (x_i, y_i) on the loss at a test point x_t via a second-order Taylor approximation of the ERM objective (Koh & Liang, 2017). However, computing influence scores requires Hessian–vector products and often suffers from numerical instability in deep networks (Basu et al., 2020; Feldman, 2020). This makes traditional influence estimation extremely expensive and frequently unreliable in practice.

In contrast, recent theoretical and empirical findings show that for deep networks with a linear final layer—which includes essentially all modern CNNs and transformers—**the influence of a training example can be well-approximated by nearest neighbors in the last-layer representation space** (Yeh et al., 2018; Pruthi et al., 2020). Let $f_\theta : \mathcal{X} \to \mathbb{R}^d$ denote the network embedding (all layers except the final classifier). The logit for class c is then

$$F_c(x) = w_c^\top f_\theta(x), \tag{20}$$

where $w_c \in \mathbb{R}^d$ are the classifier weights.

Influence as Representation Similarity. For a training example (x_i, y_i) define its embedding $z_i = f_\theta(x_i)$ and similarly $z_t = f_\theta(x_t)$ for a test example. Under mild regularity assumptions, the first-order approximation of the change in logits when replacing x_t with x_i in training yields an influence score proportional to the similarity of their representations:

$$I(i \to t) \;\propto\; \langle z_i, z_t \rangle. \tag{21}$$

Cosine similarity provides a numerically stable version:

$$S(i,t) = \frac{\langle z_i, z_t \rangle}{\|z_i\|_2 \, \|z_t\|_2}. \tag{22}$$

Thus, the **most influential training example** is simply the nearest neighbor of x_t in representation space:

$$i^* = \arg\max_{i \in \mathcal{D}_{\text{train}}} S(i,t). \tag{23}$$

Assigning the label of the most influential training point yields the influence-based prediction:

$$\hat{y}_t = y_{i^*}. \tag{24}$$

Representer Theorem Connection. Yeh et al. (2018) show that for deep networks trained with weight decay, each prediction can be decomposed as a weighted sum of training example similarities:

$$F_c(x_t) = \sum_{i=1}^{n} \alpha_{i,c} \langle z_i, z_t \rangle, \tag{25}$$

where $\alpha_{i,c}$ depend on the classifier weights and loss. In many classification settings, the coefficients $\alpha_{i,c}$ are approximately uniform within a class, yielding a natural justification for nearest-neighbor influence estimation.

Practical Advantages. The last-layer influence approximation offers:

- **No gradients or Hessians**: only forward passes.
- **Low memory**: store d-dim embeddings, not full gradients.
- **Stability**: representation similarity is smooth and robust.
- **Faithfulness**: provably consistent with linear last layers.

Empirically, on MNIST and CIFAR-10, the influence-NN classifier achieves accuracy close to classical nearest neighbors in the learned embedding space, far exceeding gradient-based influence methods which often perform near chance level due to noise.

Algorithm 1 Influence-Based Prediction via Last-Layer Nearest Neighbors

Require: Trained network f_θ, training set $\{x_i, y_i\}$, test input x_t

1: Precompute embeddings $z_i = f_\theta(x_i)$ for all training points.

2: Compute test embedding $z_t = f_\theta(x_t)$.

3: Compute similarities $S(i,t)$ for all i:

$$S(i,t) = \frac{\langle z_i, z_t \rangle}{\|z_i\|_2 \|z_t\|_2}.$$

4: Identify the most influential example:

$$i^* = \arg\max_i S(i,t).$$

5: **return** predicted label $\hat{y}_t = y_{i^*}$.

Table 5: Top-1 retrieval accuracy (%) as perturbation strength increases (None/Tiny/Small/Medium/Full = identity, noise-only, 5°+1px, 10°+1.5px, 15°+2px+noise; Section 6.1), before the augmentation fix (cf. Figure 4 after the fix); "–" = not evaluated at that level.

	MNIST				FMNIST		CIFAR-10	
Pert.	**Prov.**	**kNN**	**Infl.**	**PPNet**	**Prov.**	**kNN**	**Prov.**	**kNN**
None	99.8	100.0	–	–	98.5	100.0	77.4	100.0
Tiny	99.4	98.9	–	–	96.0	99.4	73.2	78.1
Small	33.2	57.6	–	–	33.3	47.9	**14.6**	11.8
Medium	13.5	33.5	–	–	14.0	18.6	**5.8**	3.2
Full	5.3	15.5	11.9	0.0	5.2	6.0	**1.7**	0.4

This approach therefore serves as an inexpensive and theoretically grounded proxy for influence functions, while avoiding the instability, memory cost, and high computational burden of gradient-based methods.

Behavior under misclassification (does the index branch's retrieval remain informative when the task branch's own prediction is wrong?) is reported in the main body, Section 6.4, since it speaks directly to using the provenance mechanism for model debugging.

9.8 Extended Results: Test-Time Attribution Quality (Pre-Augmentation Sweep)

Section 6.3 summarizes the headline (full-perturbation) numbers from the pre-augmentation sweep in text; Table 5 gives the complete sweep across all perturbation levels on all three datasets, together with the two MNIST-only baselines (Infl. = influence proxy, PPNet = ProtoPNet-style).

Table 6 gives the corresponding classification (task) accuracy sweep referenced in Section 6.3, showing task accuracy is far more robust to the same perturbations than the index branch's retrieval accuracy above.

9.9 Additional Analysis: Loss-Balancing Sweep

The loss-balancing hyperparameter does not reliably control attribution quality. A natural follow-up question is how λ_{class} and λ_{index} (denoted α, β here, with $\alpha + \beta = 1$) should be chosen, and whether classification and attribution trade off against each other as the weighting changes – ideally traced out as a Pareto front. We swept $(\alpha, \beta) \in \{(0.1, 0.9), (0.3, 0.7), (0.5, 0.5), (0.7, 0.3), (0.9, 0.1)\}$ on MNIST with the augmented two-branch network, training 3 seeds per setting (40 epochs each) to separate a genuine trend from single-run training variance. Table 7 reports test classification accuracy and test-time attribution accuracy (top-1, full perturbation protocol), averaged over seeds.

Classification accuracy is essentially flat across the sweep (99.05–99.47%), which is expected since MNIST classification is easy regardless of how much loss weight it receives. Attribution accuracy is

Table 6: Classification (task) accuracy (%) across the same perturbation sweep as Table 5, before vs. after augmentation training (No/Aug); perturbation abbreviations as above.

	MNIST		FMNIST		CIFAR-10	
Pert.	**No**	**Aug**	**No**	**Aug**	**No**	**Aug**
None	100.0	99.9	98.7	95.3	91.8	94.1
Tiny	100.0	99.9	97.7	95.0	91.8	93.2
Small	98.9	99.6	90.3	95.1	77.3	91.8
Medium	97.2	99.5	81.1	92.9	66.7	90.9
Full	92.6	99.5	71.5	90.0	58.6	82.5

Table 7: Test classification accuracy and test-time attribution accuracy (top-1, %) on MNIST as the loss-balancing weights vary, mean $\pm$ std over 3 seeds per setting.

(α, β)	**Class. acc.**	**Attribution top-1**
(0.1, 0.9)	99.05 ± 0.08	28.0 ± 10.5
(0.3, 0.7)	99.32 ± 0.05	23.8 ± 10.5
(0.5, 0.5)	99.39 ± 0.06	28.2 ± 13.1
(0.7, 0.3)	99.37 ± 0.08	26.3 ± 10.1
(0.9, 0.1)	99.47 ± 0.07	48.2 ± 7.2

where a clean trade-off should appear if β (the instance-loss weight) directly controlled attribution quality – but it does not: the standard deviation across seeds within a single setting (7–13 points) is comparable to or larger than the range across settings (23.8–48.2 points), so the five means do not trace out a reliable, monotonic curve. If anything, the weak signal that survives points the *opposite* direction from the naive prediction: $(\alpha, \beta) = (0.9, 0.1)$, which weights the instance loss least, has the highest mean attribution accuracy and the lowest cross-seed variance, while intermediate weightings are noisier and not clearly better. We do not have a confident mechanistic explanation for this; it may reflect that, once augmentation is present, the class branch's accuracy (which conditions the index search space) matters more for attribution accuracy than the nominal instance-loss weight itself, or that 40-epoch single-seed training on this architecture is simply too noisy an instrument to resolve the trade-off ogDP asked about. Either way, we report it as an honest answer rather than fitting a curve to noise: *at this scale, the loss-balancing hyperparameter does not give practitioners reliable, predictable control over the classification/attribution trade-off*, which is itself useful for anyone tuning this architecture, and a concrete target for follow-up work (more seeds, more epochs, or a different parameterization of the trade-off).

9.10 PROTOPNET-STYLE BASELINE: IMPLEMENTATION DETAILS

We implement a scoped-but-faithful version of ProtoPNet (Chen et al., 2019) for the comparison in Section 6.2, on MNIST.

Architecture. The backbone is identical to the provenance network's shared feature extractor (Appx. 9.2): three conv blocks reducing to a 4×4 spatial feature map, followed by a 1×1 conv projecting to 512 channels (the prototype dimension). We use 10 prototype vectors per class (100 total for MNIST), each compared against every spatial location of the feature map via squared Euclidean distance; the minimum distance over spatial locations, per prototype, is converted to a similarity score $\log\left(\frac{d^2+1}{d^2+\epsilon}\right)$ following the original paper, and a linear last layer maps the 100-dimensional similarity vector to 10 class logits.

Losses. Training minimizes cross-entropy plus a cluster cost (the nearest same-class prototype to each training patch should be close) plus a separation cost (the nearest other-class prototype should be far), plus ℓ_1 regularization on the last-layer weights, with coefficients $\lambda_{\text{cluster}} = 0.8$, $\lambda_{\text{sep}} = 0.08$, $\lambda_{\ell_1} = 10^{-4}$.

Training schedule (simplified from the original paper). We use a single joint-training phase (20 epochs, Adam, lr 10^{-3}) rather than the original's alternating warmup/joint/push/last-layer-only schedule, followed by one *push* step – each prototype is replaced by the feature vector of the closest actual training-image patch among same-class training images, computed over the full training set – and a short last-layer-only fine-tune (5 epochs, prototypes and backbone frozen). Test accuracy was 98.79% immediately before the push, 97.97% immediately after (the expected discrete accuracy drop from snapping continuous prototypes to real patches), and 98.83% after fine-tuning.

Explanation retrieval. For a test input, we identify the training image each of the predicted class's 10 prototypes was pushed to, rank them by similarity to the input, and treat the top-ranked training image as the model's explanation (top-1) or the top-5 as a relaxed variant – the same top-1/top-5 retrieval-accuracy protocol used for the provenance net and kNN baselines (Section 6.1), evaluated on the identical $1{,}000$ perturbed probes.

Why the score is structural, not incidental. ProtoPNet's attribution accuracy is essentially 0% at *every* perturbation level we tested, including with no perturbation at all (identity: 0.1% top-1), even though its classification accuracy stays reasonably robust (100% at identity, degrading gracefully to 92.1% at the strongest perturbation – comparable to the provenance net's own classification robustness in Table 6). This is not a training failure: with only 10 prototypes per class explaining behavior over $\sim 6{,}000$ same-class training images, the maximum possible top-1 hit rate under our exact-source-recovery metric is $10/6{,}000 \approx 0.167\%$ – matching the observed rate almost exactly. ProtoPNet's design point is fundamentally different from provenance networks' or the kNN baseline's: it summarizes each class with a handful of human-interpretable, representative prototypes rather than indexing every training sample, so it can never be exact-source-competitive under this metric regardless of how well it is trained. Final test classification accuracy after the push and last-layer fine-tune was 98.83%, closely comparable to the provenance net's own accuracy – so on the prediction-accuracy axis the two architectures are on par, but they answer a categorically different explainability question: ProtoPNet asks "which of a small set of interpretable patterns does this input resemble," while provenance networks and kNN retrieval ask "which specific training example is this," and only the latter is answerable by a 10-prototype-per-class summary in principle.

9.11 SCALABILITY THROUGH SUBSET SAMPLING: EXTENDED RESULTS

To comprehensively evaluate the scalability approach presented in Section 5.2, we conducted experiments across fine-grained subset ratios from 10% to 100% of the training data. Table 2 (in Section 5.2) presents complete results for MNIST and FashionMNIST; the observations below give the full discussion behind that section's summary.

Experimental Setup: We use stratified sampling to maintain class proportions when selecting subsets. Both CNN1 (classification branch) and CNN2 (index branch) are trained on the same subset of training data, with CNN2 predicting indices only within the selected subset. Both models share the initial convolutional layer and are trained jointly for 100 epochs. All models are evaluated on the complete 10,000-sample test set.

Table 8: Index-head final-layer parameters, flat vs. factorized (K=2), at increasing scale (extrapolated).

M	Flat	Factorized	Ratio
6,742	6.90M	170K	41×
60,000	61.4M	502K	122×
1,000,000	1.02B	2.05M	500×

Key Observations:

General trends with data scale: As expected, increasing the subset size generally improves performance, with CNN1 test accuracy improving from 98.27% (10% subset) to 99.26% (90% subset) on MNIST. However, the improvements plateau beyond 50-70%, demonstrating diminishing returns. Notably, even with severely limited subsets (10% = 6,000 samples), CNN1 achieves respectable accuracy (86.66% on FashionMNIST, 98.27% on MNIST), validating that provenance networks can operate effectively when trained on substantially reduced data.

Non-monotonic index prediction: CNN2 Top-1 accuracy does not increase monotonically with subset size. For MNIST, Top-1 accuracy peaks at 30% (79.72%) and 90% (83.26%), while dropping at intermediate points (e.g., 50% → 69.94%). This counterintuitive pattern suggests that adding more training samples to the index vocabulary introduces confusion between visually similar examples, making exact index prediction harder even as the model has more data. The effect is less pronounced in Top-5 and Top-10 metrics, which remain more stable.

Stable semantic retrieval: Top-5 and Top-10 accuracies show much more consistent trends across subset sizes, indicating the network successfully identifies semantically relevant training samples regardless of exact index prediction difficulty. For instance, MNIST Top-5 accuracy ranges from 92.05% to 96.53% across all subsets, with no dramatic drops. This validates the Top-K retrieval strategy for provenance tracking—the network learns to map test samples to their nearest neighbors in the training set, even when pinpointing the exact closest sample proves difficult.

Dataset complexity effects: FashionMNIST shows performance saturation beyond 50%, with minimal improvement from additional subset samples (90%: 92.19% vs 50%: 90.86%). This suggests that for more complex datasets with higher intra-class variation, carefully selected representative samples (prototypes, boundary cases, or diversity-maximizing selections) may be more effective than random stratified sampling. The marginal gains from 50% to 90% (1.33 percentage points) come at the cost of nearly doubling the index head size.

Practical deployment: For MNIST, training on 30% of data achieves 98.87% classification accuracy with 95.49% Top-5 retrieval, representing 70% parameter reduction in the index head (17,995 vs 60,000 output neurons). For FashionMNIST, training on 50% achieves 90.86% classification with 89.71% Top-5 retrieval and 50% parameter reduction. These results demonstrate practical scalability improvements while maintaining competitive performance. The trade-off between parameter efficiency and accuracy allows practitioners to select operating points based on deployment constraints: resource-constrained settings can use 30-50% subsets with minimal accuracy loss, while applications requiring maximum accuracy can use 70-90% subsets while still achieving meaningful compression compared to full indexing.

9.12 Extended Results: Factorized Index Head

Section 5.3 summarizes the headline parameter-reduction and convergence numbers for the factorized index head in text. Table 8 gives the full parameter-count scaling table underlying the parameter-reduction claim.

Table 9 gives the full epoch-budget convergence sweep, from-scratch and warm-started, underlying the convergence summary. Per-digit accuracy at the 40-epoch budget (20.8% and 36.1% individually, 10.2% joint vs. 7.5% expected under independence, measured without augmentation) confirms the two digit heads are learning close to independently, rather than one head's difficulty masking the other's.

Warm-starting the backbone and instance-branch trunk from the already-trained flat model – so the digit heads are the only randomly-initialized component – gives roughly a 2× speedup early

Table 9: Exact training-index-match accuracy at increasing epoch budgets, factorized index head, MNIST (single run per configuration).

Epochs	From scratch	Warm-started
40	8.2%	6.8%
160	53.8%	31.9%
320	78.0%	48.4%
400	–	50.7% (plateau)

on (6.8% vs. 3.0% at 40 epochs, augmented recipe), and, extended to $10\times$ the usual budget (400 epochs), converges to a genuine plateau at $\sim$50.7% (epochs 320–400 hover within 48–51% with no further net gain) – still well short of the flat head's 89.8%, and, notably, short of the 78.0% reached by the non-augmented from-scratch run at 320 epochs, consistent with augmentation making an already-hard optimization problem harder still. Warm-starting speeds up early training and shifts where the trajectory plateaus, but does not close the gap to the flat head, confirming the bottleneck is the digit-head readout itself rather than the shared representation.

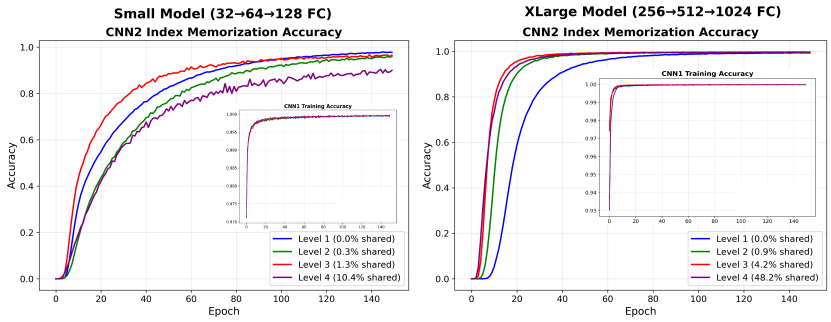


Figure 13: Accuracy per epoch for the index branch and class branch (insets) of Small (top) and XLarge (bottom) models. Each curve represents a different level of parameter sharing (on MNIST). See also Appx. 9.13.

9.13 Analysis of network size and parameter sharing

Section 5.4 in the main body summarizes the headline parameter-sharing result; here we give the full experimental setup and results across both datasets and all four sharing levels.

We conducted systematic parameter sharing experiments across multiple model scales and datasets to understand how different levels of parameter sharing affect task performance. All models follow a convolutional neural network architecture with progressive channel expansion, consisting of three convolutional layers followed by fully connected layers.

Table 10: MNIST results across all model sizes and sharing levels. C2 Cls-T1/T5 denotes CNN2 Class Consistency Top-1/Top-5 accuracy.

Model	Level	Sharing%	Total Params	CNN1 Test	CNN2 Memo (training)	C2 Cls-T1/T5
Small (32→64→128)	1	0.0%	4.0M	0.992	0.987	0.817 / 0.972
	2	0.3%	4.0M	0.992	0.982	0.820 / 0.972
	3	1.3%	4.0M	0.991	0.987	0.828 / 0.975
	4	10.4%	3.9M	0.991	0.919	0.766 / 0.944
Medium (64→128→256)	1	0.0%	17.2M	0.993	1.000	0.850 / 0.977
	2	0.4%	17.2M	0.993	1.000	0.847 / 0.977
	3	2.1%	17.1M	0.993	1.000	0.855 / 0.978
	4	18.8%	16.1M	0.993	0.963	0.827 / 0.965
Large (128→256→512)	1	0.0%	35.8M	0.993	0.997	0.863 / 0.981
	2	0.7%	35.7M	0.994	0.997	0.861 / 0.981
	3	3.1%	35.5M	0.993	0.997	0.862 / 0.981
	4	31.8%	33.0M	0.992	0.994	0.848 / 0.976
XLarge (256→512→1024)	1	0.0%	79.6M	0.993	0.992	0.869 / 0.982
	2	0.9%	79.5M	0.993	0.992	0.866 / 0.981
	3	4.2%	79.0M	0.993	0.993	0.866 / 0.981
	4	48.2%	71.9M	0.992	0.994	0.852 / 0.978

Table 11: Fashion-MNIST results across all model sizes and sharing levels. C2 Cls-T1/T5 denotes CNN2 Class Consistency Top-1/Top-5 accuracy.

Model	Level	Sharing%	Total Params	CNN1 Test	CNN2 Memo (training)	C2 Cls-T1/T5
Small (32→64→128)	1	0.0%	4.0M	0.895	0.933	0.555 / 0.820
	2	0.3%	4.0M	0.893	0.944	0.568 / 0.832
	3	1.3%	4.0M	0.895	0.998	0.624 / 0.877
	4	10.4%	3.9M	0.887	0.826	0.521 / 0.787
Medium (64→128→256)	1	0.0%	17.2M	0.908	0.757	0.439 / 0.717
	2	0.4%	17.2M	0.906	0.977	0.595 / 0.850
	3	2.1%	17.1M	0.906	0.982	0.611 / 0.860
	4	18.8%	16.1M	0.893	0.950	0.565 / 0.827
Large (128→256→512)	1	0.0%	35.8M	0.909	0.987	0.618 / 0.868
	2	0.7%	35.7M	0.912	0.996	0.629 / 0.876
	3	3.1%	35.5M	0.910	0.998	0.633 / 0.879
	4	31.8%	33.0M	0.905	0.998	0.622 / 0.872
XLarge (256→512→1024)	1	0.0%	79.6M	0.914	0.835	0.504 / 0.768
	2	0.9%	79.5M	0.913	0.998	0.631 / 0.879
	3	4.2%	79.0M	0.914	0.998	0.635 / 0.882
	4	48.2%	71.9M	0.908	0.998	0.623 / 0.873

We evaluated four model sizes with increasing capacity. The Small Model uses a 32→64→128 channel progression with 128 FC units, totaling approximately 4M parameters. The Medium Model

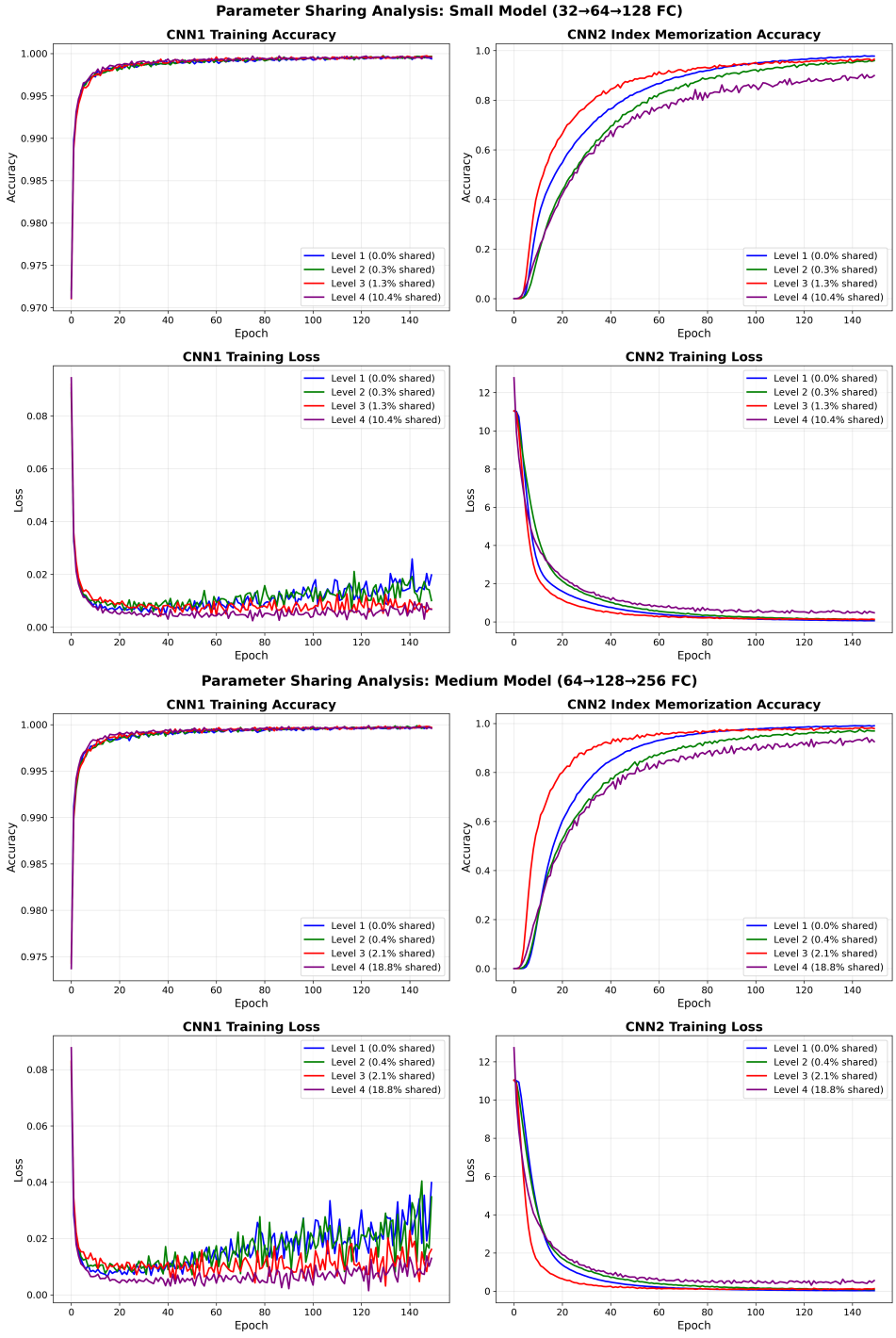


Figure 14: Convergence plots for class branch (CNN1) and index branch (CNN2) for a small (top) and medium (bottom) size CNNs over MNIST dataset.

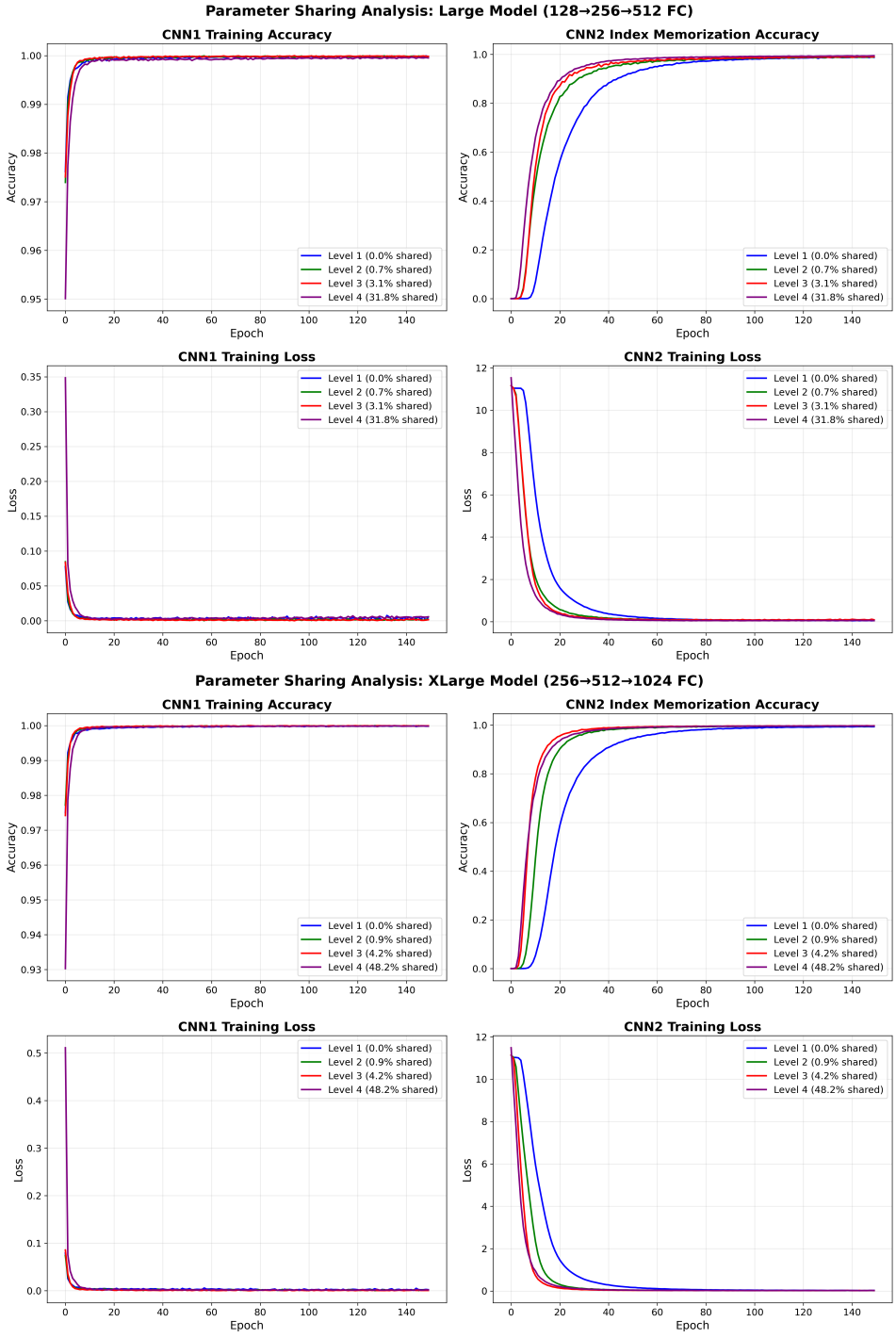


Figure 15: Convergence plots for class branch (CNN1) and index branch (CNN2) for a large (top) and xlarge (bottom) size CNNs over MNIST dataset.

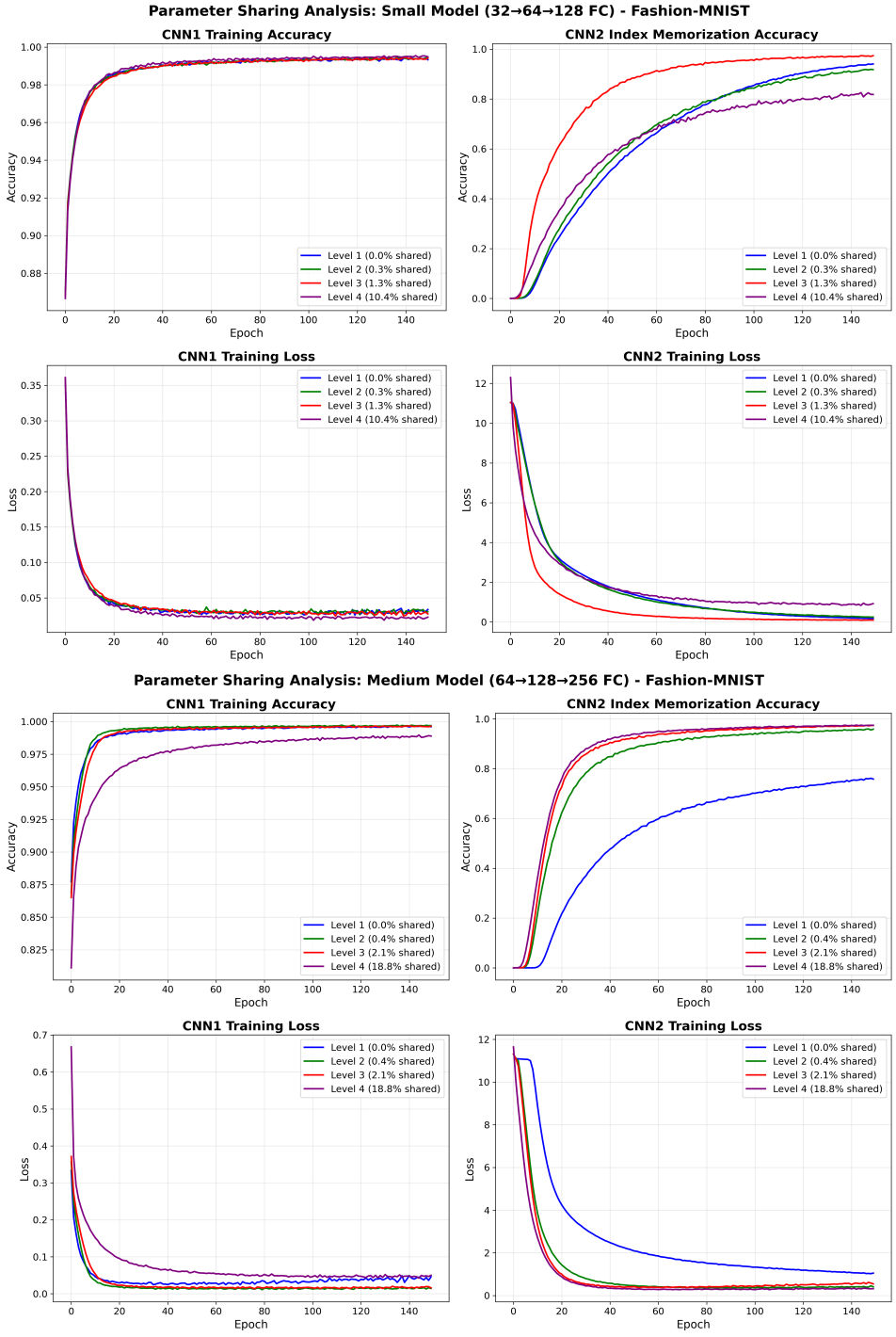


Figure 16: Convergence plots for class branch (CNN1) and index branch (CNN2) for a small (top) and medium (bottom) size CNNs over FashionMNIST dataset.

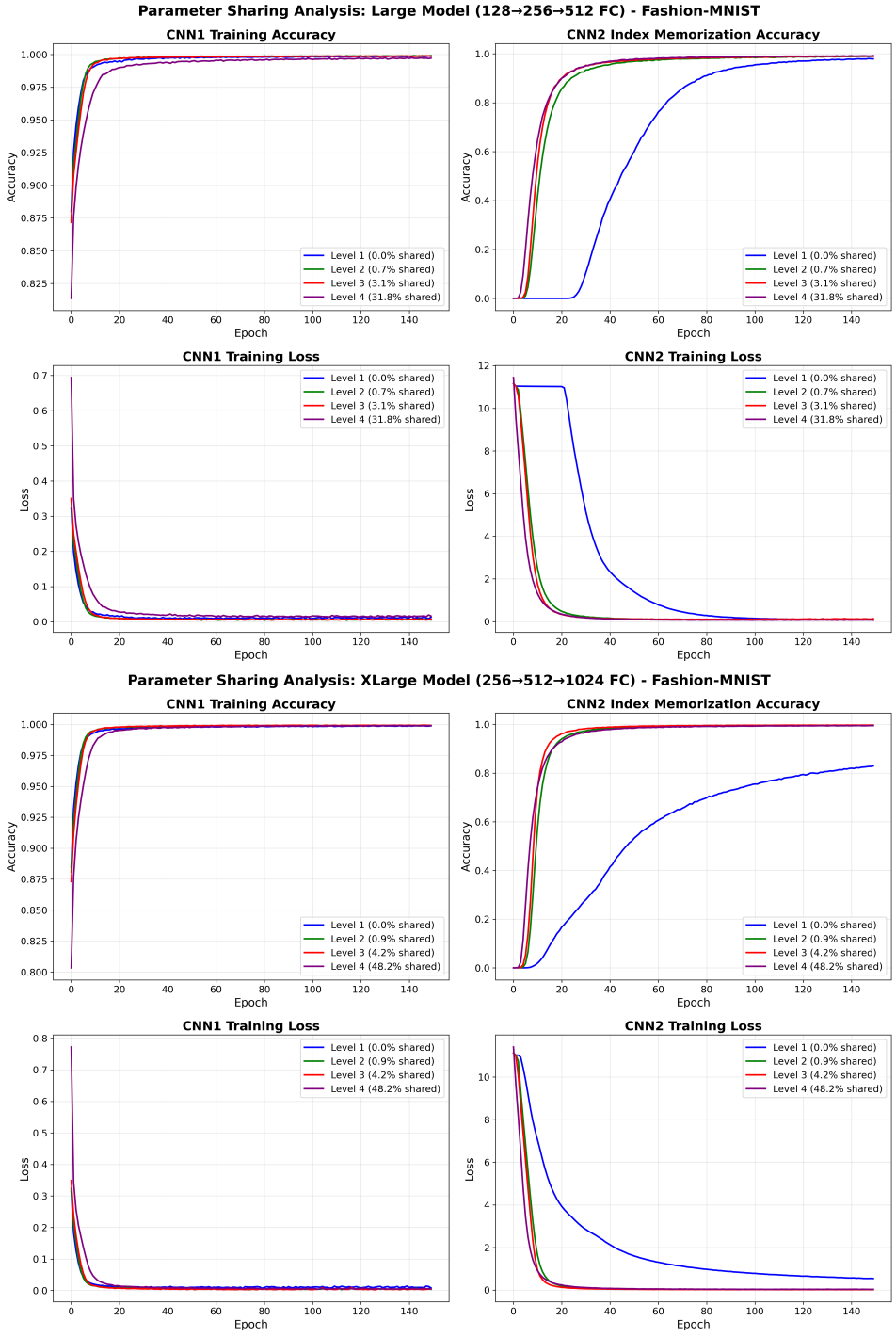


Figure 17: Convergence plots for class branch (CNN1) and index branch (CNN2) for a large (top) and xlarge (bottom) size CNNs over FashionMNIST dataset.

expands to 64→128→256 channels with 256 FC units, reaching approximately 17M parameters. The Large Model further scales to 128→256→512 channels with 512 FC units, comprising approximately 35M parameters. Finally, the XLarge Model uses 256→512→1024 channels with 1024 FC units, totaling approximately 80M parameters. All models include dropout (0.5) and ReLU activations.

We implemented four levels of parameter sharing between two networks. Level 1 shares only the first convolutional layer. Level 2 shares the first two convolutional layers. Level 3 shares all three convolutional layers. Level 4 shares all convolutional layers plus the first fully connected layer. This progressive sharing design allows us to study how increasing amounts of shared representations affect task interference and performance.

9.13.1 Tasks and Training Protocol

We trained a Two-Staged provenance Network (non-class conditional) for this task.

Batch sizes varied by model scale for memory efficiency. Small, Medium, and Large models used batch sizes of 64 for CNN1 and 32 for CNN2. The XLarge model required reduced batch sizes of 32 and 16 respectively to fit in GPU memory. Learning rates were adjusted based on model capacity: Small and Medium models used 0.001 for CNN1 and 0.0001 for CNN2; Large models used 0.0005 for both networks; XLarge models used 0.0002 for both networks to ensure stable training at massive scale.

9.13.2 Evaluation Metrics

We evaluated models using 4 key metrics as diplayed in Tables 10, 11. CNN1 test accuracy measures classification performance on the held-out test set, indicating generalization capability. CNN2 training accuracy measures index memorization performance on training data, showing the network's capacity to memorize individual instances. CNN2 class consistency (Top-1 and Top-5) evaluates whether memorized training indices preserve semantic structure: when shown a test image, do the top-1 or top-5 predicted training sample indices belong to the correct class? This metric reveals whether memorization captures class-level patterns beyond pure instance recall. Finally, we computed the sharing ratio as the percentage of shared parameters relative to total unique parameters across both networks.

9.13.3 Capacity and Sharing Dynamics

Model capacity fundamentally shapes parameter sharing dynamics. Small models showed the strongest interference effects, particularly at Level 4 sharing where limited capacity forced direct competition between tasks. Medium and Large models demonstrated that increased capacity reduces interference, enabling near-perfect performance on both tasks even with substantial sharing. The XLarge Model revealed that massive capacity (approximately 80M parameters) can accommodate up to 48% parameter sharing with minimal degradation, suggesting that capacity-constrained interference diminishes as models scale.

Dataset complexity interacted with model capacity in predictable ways. MNIST's simpler visual patterns allowed even Small models to achieve strong performance across sharing levels. Fashion-MNIST's increased complexity revealed clearer capacity constraints: Small models showed significant memorization degradation at high sharing levels, while larger models maintained strong performance. This pattern suggests that complex datasets require proportionally more capacity to support parameter sharing without interference.

9.13.4 Optimal Sharing Levels

Level 3 sharing (all convolutional layers) emerged as optimal for most configurations, particularly on complex datasets. This level provided sufficient shared feature extraction while preserving task-specific capacity in FC layers. Level 4 sharing (including first FC layer) created the highest parameter overlap but showed performance degradation in capacity-constrained settings, especially for Small models on Fashion-MNIST.

Interestingly, Level 1 sharing (first conv layer only) sometimes underperformed on Fashion-MNIST, particularly for Medium and XLarge models. This suggests that minimal sharing provides insufficient feature extraction capacity for complex visual tasks, and that intermediate sharing levels enable better learned representations through multi-task pressure on shared parameters.

9.13.5 Memorization and Semantic Structure

Class consistency metrics revealed that memorization preserves semantic structure beyond pure instance recall. Top-5 class consistency substantially exceeded Top-1 across all configurations, indicating that memorized indices cluster by class even when exact matches are imperfect. This demonstrates that the memorization task implicitly learns class-level representations.

Higher sharing levels generally improved class consistency, suggesting that shared representations encode semantic information more effectively than task-specific features. This pattern was most pronounced in larger models, where Level 3-4 sharing achieved the highest class consistency despite having the greatest parameter overlap. This finding suggests that forcing networks to share representations encourages learning of generalizable semantic features.

9.13.6 Training Dynamics

Analysis of training curves revealed distinct convergence patterns. CNN1 classification typically plateaued within 20 epochs, indicating rapid learning of discriminative features. CNN2 memorization exhibited slower, more gradual improvement throughout the 150-epoch training period, reflecting the difficulty of learning 60,000-way classification.

Level 1-3 sharing produced smooth, stable loss curves across all model sizes. Level 4 sharing introduced instability in Small models, manifested as oscillating training loss, particularly on Fashion-MNIST. This instability disappeared in larger models, confirming that capacity constraints drive interference effects at high sharing levels.

Higher sharing levels accelerated CNN2 convergence in larger models, suggesting that shared task-relevant features bootstrap memorization learning. This effect was absent in Small models, where capacity constraints prevented efficient feature sharing.

9.13.7 Implications

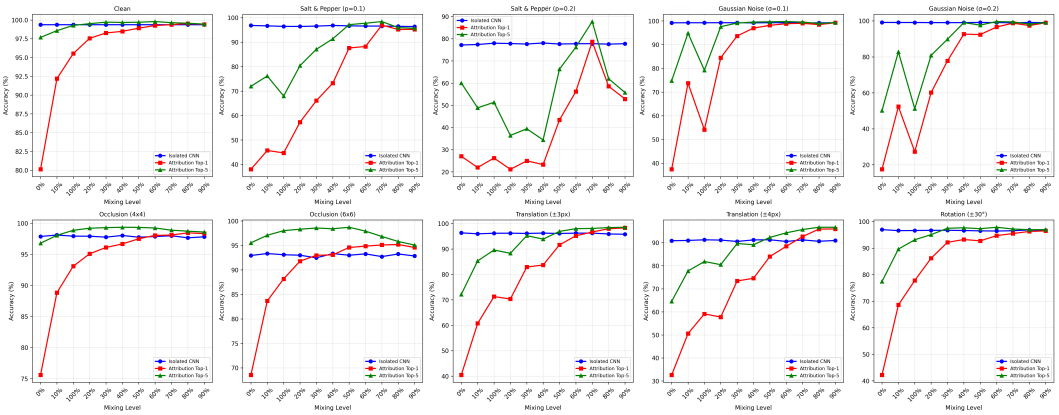


Figure 18: Comparison of the single-branch index-prediction network with varying levels of label mixing against an isolated CNN. Plots show Top-1 and Top-5 accuracy under 9 distortion types plus a baseline without distortion. The variation in the isolated CNN (blue curves) across different index-mixing levels arises from the use of different test sets at each level. Intermediate levels of memorization improve robustness: for distortions like occlusion and blur, partial label mixing (20–30%) yields higher accuracy than the isolated CNN. Extended results (remaining distortions, FashionMNIST) are in Appx. 9.14.2.

9.14 Extended Results: Robustness to Image Distortions

Section 7 introduces this application in the main body. Here we give the full setup, the 9-distortion MNIST view (Figure 18), the extended sweep – all 14 distortion types, plus FashionMNIST (Appx. 9.14.2) – and a direct test of the stronger, adversarial-robustness version of the underlying hypothesis (Appx. 9.14.1).

We compare a single-branch index-prediction network with varying levels of label mixing (similar to label smoothing) against a standard CNN trained independently ("isolated CNN"); while the index-prediction network infers indices, the isolated CNN directly predicts class labels. Performance is evaluated under 14 conditions covering 8 distortion types – salt-and-pepper noise, Gaussian noise, occlusion, translation, rotation, scaling, Gaussian blur, and motion blur – along with a baseline condition without distortion.

Results on MNIST are presented in Figure 18. As observed earlier (Section 5.1), increasing the level of mixing (hence reducing memorization) generally improves classification accuracy. Interestingly, for certain distortions – such as occlusion, translation, Gaussian, and motion blur – intermediate levels of index mixing yield higher accuracy than the isolated CNN, as indicated by points where the red curve (Top-1 acc) crosses above the blue line (again Top-1 acc). This suggests that some degree of memorization ($\sim$20–30%, corresponding to label mixing $>$ 70–80%) can enhance robustness. A possible explanation is that, under certain distortions (*e.g.* occlusion), the index branch can still retrieve appropriate training samples, whereas a purely classification-based net (isolated CNN) loses this information. This should be read alongside Appx. 9.14.1 below: the same architecture tested directly against adversarial (rather than incidental) perturbations shows the opposite effect.

9.14.1 Adversarial Robustness: A Direct Test

The distortion-based result above is suggestive but does not test the stronger, more specific hypothesis that the index branch provides useful *adversarial* robustness by grounding predictions in retrievable training evidence, or that agreement between the main and index branches can serve as an adversarial-example detector. We tested this directly on MNIST, comparing the augmented two-stage network's main branch against a matched, independently-trained classifier baseline (same backbone, task loss only), under FGSM ($\epsilon \in \{0.05, 0.1, 0.2\}$) and PGD (40 steps, step size 0.01, $\epsilon = 0.2$) attacks, plus Gaussian and salt-and-pepper noise, on a fixed 2,000-image test subset. Because the class-conditional index branch's search space is gated by the main branch's own class prediction (Section 3), its retrieved exemplar's label trivially always matches that prediction under this architecture; we instead use kNN retrieval in the model's own embedding space (Section 6.2's influence-proxy construction) as the index-branch signal, since it can genuinely disagree with the main branch, which the detection mechanism requires.

The main hypothesis does not hold: the auxiliary index branch does not make the main task more robust, and if anything makes it less so. Table 12 shows the classifier baseline consistently outperforms the two-stage network's main branch under every attack and noise condition we tested, except at the strongest attack (PGD), where both are near floor. The gap is largest at moderate perturbation strength (FGSM $\epsilon = 0.05$: 81.9% vs. 49.3%; salt-and-pepper $p = 0.05$: 95.9% vs. 62.0%). The kNN-embedding top-K vote, which we hoped might offer redundancy, is consistently the *least* robust of the three signals rather than an improvement. A plausible explanation, consistent with our own model-size analysis (Section 5.4) and the classification/attribution decoupling in Section 6: the instance-prediction loss competes with the classification loss for representational capacity and gradient budget, so a jointly-trained main branch ends up less hardened for pure classification than a dedicated classifier trained without that competition.

Table 12: Classification accuracy (%) under adversarial attacks and random noise, classifier baseline vs. the two-stage network's main branch, MNIST, n=2,000.

Condition	**Classifier**	**Two-stage (main)**	**kNN top-K vote**
Clean	99.6	99.5	98.5
FGSM ϵ=0.05	81.9	49.3	55.7
FGSM ϵ=0.1	37.7	25.5	24.7
FGSM ϵ=0.2	15.8	13.5	10.5
PGD ϵ=0.2, 40 steps	0.0	0.9	3.4
Gaussian σ=0.1	96.1	86.6	74.2
Gaussian σ=0.3	20.0	13.5	13.7
Salt-and-pepper p=0.05	95.9	62.0	42.2
Salt-and-pepper p=0.2	22.0	18.0	15.0

The one part of the original hypothesis that does hold: consistency-based adversarial detection. Using disagreement between the main branch's prediction and the kNN-embedding vote as a detection statistic, AUROC ranges from 0.657 (FGSM ϵ=0.05) to 0.854 (FGSM ϵ=0.2), and 0.682 for PGD – meaningfully better than chance, though far from perfect, and driven by consistency dropping from 98.5% on clean inputs to as low as 27.8% under strong FGSM. This is a real, if modest, capability: the two branches genuinely do disagree more often under attack, which is a usable (not perfect) detection signal, even though it does not translate into the main branch itself being more robust.

Conclusion. We reject the hypothesis that the provenance objective improves adversarial robustness, and instead find the opposite: the classifier baseline is consistently more robust than the two-stage network's main branch. We report both results plainly: the architecture provides a usable (if imperfect) adversarial-example detector via branch disagreement, but does not provide the robustness benefit that seemed plausible before testing it.

9.14.2 Extended distortion types

We evaluate robustness under a diverse set of input distortions, shown in Figure 19 over four samples. Results over all 14 distortion types are shown in Figure 20.

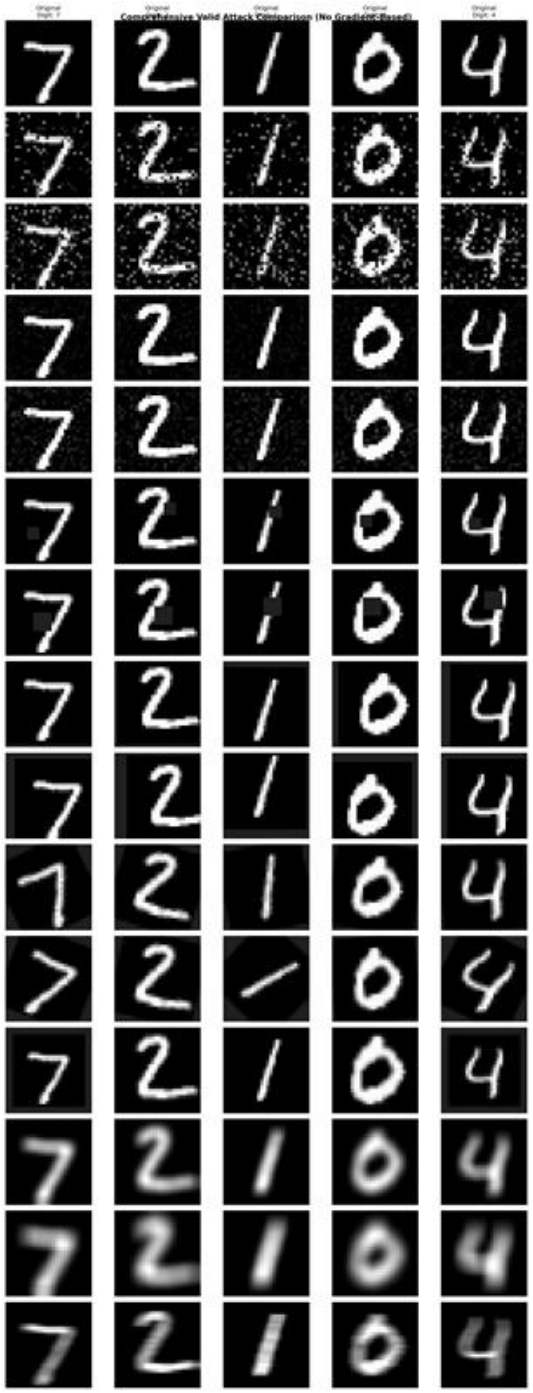


Figure 19: Distortion types in order from top to bottom: no distortion, salt-and-pepper noise (p=0.1, p=0.2), Gaussian noise ($\sigma = 0.1, \sigma = 0.2$), occlusion ($4 \times 4$, 6×6), translation ($\pm$ 3px, $\pm$ 4px), rotation ($\pm 30°, \pm 40°$), scaling (0.8-1.2x), Gaussian blur ($\sigma = 1, \sigma = 2$), and motion blur (5px).

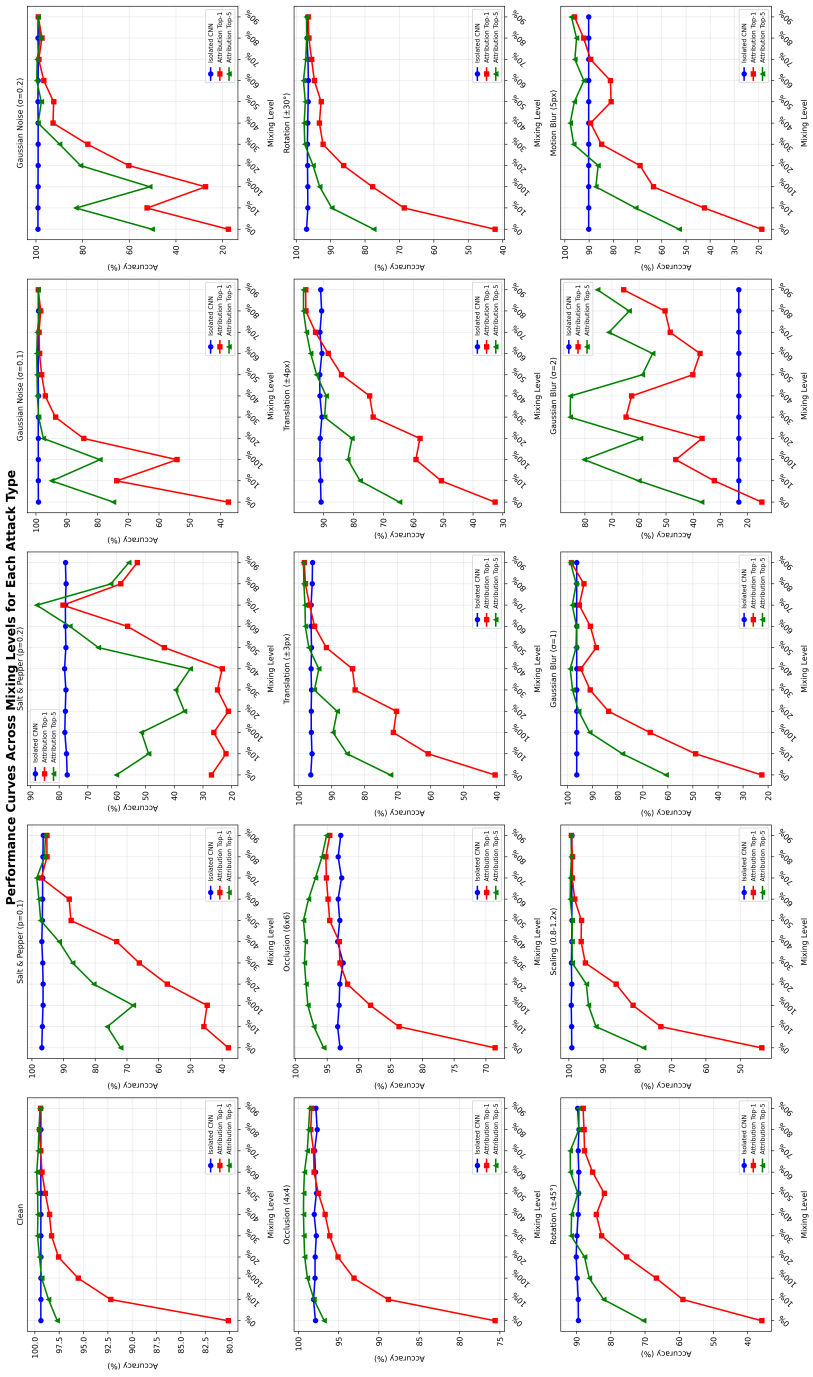


Figure 20: Comparison of the single-branch index-prediction network with varying levels of label mixing against an isolated CNN. Plots show Top-1 and Top-5 accuracy under 14 distortion types plus a baseline without distortion. The variation in the isolated CNN (blue curves) across different index-mixing levels arises from the use of different test sets at each level. Intermediate levels of memorization improve robustness: for distortions like occlusion and blur, partial label mixing (20–30%) yields higher accuracy than the isolated CNN.

9.15 Extended Results: Membership Inference

Section 7 introduces this application in the main body (Figure 5). Here we give the full experimental setup, training configuration, and additional per-dataset confidence/entropy figures.

9.15.1 Experimental setup

9.15.2 Training Configuration

We adopt a controlled training setup for membership inference analysis. The key hyperparameters are summarized in Table 13.

Table 13: Training configuration for membership inference analysis.

Setting	Value
Optimizer	AdamW (lr = 2×10^{-3}, weight decay = 2×10^{-5}, betas = (0.9, 0.999))
Schedule	Warmup (3 epochs) + Step decay ($\gamma = 0.6$, step = 8 epochs)
Epochs	40
Batch size	128
Mixed precision	Enabled (AMP)
Loss weights	$\alpha = 0.3$ (digit), $\beta = 0.7$ (index)
Other	Seed = 42, 4 workers, 5000 MIA samples
Evaluation	Top-k provenance accuracy with $k \in \{1, 5\}$

9.15.3 Model: Two-Stage CNN Attribution

The proposed **Two-Stage CNN Attribution** model jointly predicts class labels and training indices. It consists of:

- **Shared feature extractor:** Three convolutional blocks with BatchNorm, ReLU, and pooling, followed by adaptive pooling to a 4×4 grid.
- **Projection layer:** Fully connected projection to a 2048-d representation (ReLU, BatchNorm, Dropout).
- **Digit head (class prediction):** A two-layer MLP mapping to the number of classes.
- **Instance head (index prediction):** Conditioned on both the projected features and the class label. During training, the ground-truth label is used; otherwise, the predicted label (argmax) is used. The label is encoded as a one-hot vector and concatenated with the feature representation.

This design ensures that index attribution is conditioned on class identity, mimicking provenance behavior.

9.15.4 Loss Functions

We use two objectives:

- **Digit loss:** Standard cross-entropy on class prediction.
- **Instance loss:** Cross-entropy on index prediction with label smoothing (0.05).

The final training objective is a weighted sum:

$$\mathcal{L} = \alpha \, \mathcal{L}_{\text{digit}} + \beta \, \mathcal{L}_{\text{index}}.$$

9.15.5 Membership Inference Protocol

For membership inference, we query the index prediction head with candidate samples. Models trained on provenance information tend to assign higher confidence to training members than to non-members. We quantify this effect on held-out data, using 5,000 candidate samples.

Distribution of max confidence scores and entropies over train/member and test/non-member datapoint are shown next.

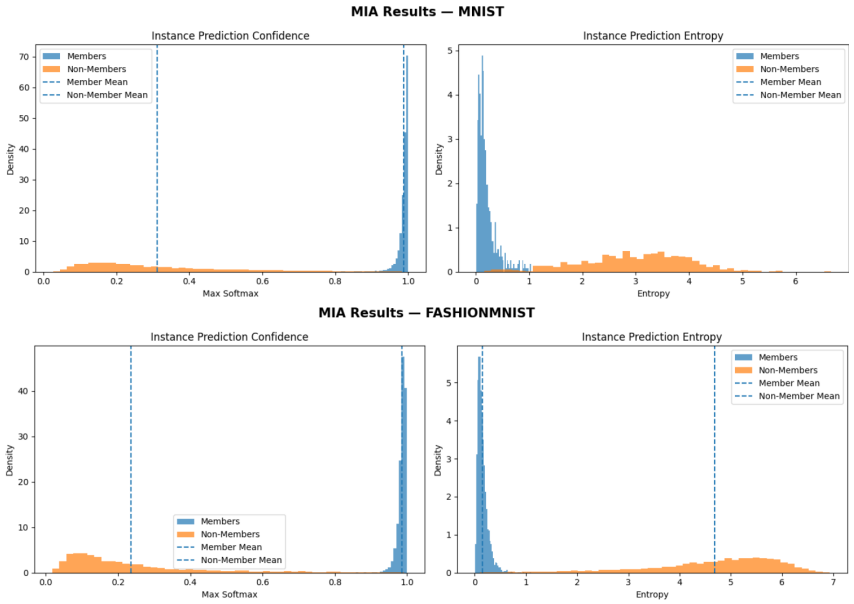


Figure 21: The left panel displays the distribution of maximum confidence scores, while the right panel shows the distribution of entropy, all from the **index branch** of a class-conditional two-branch network. Both distributions are plotted for 5K training samples (members) and 5K test samples (non-members). This data is used to generate the plot in Figure 5

.

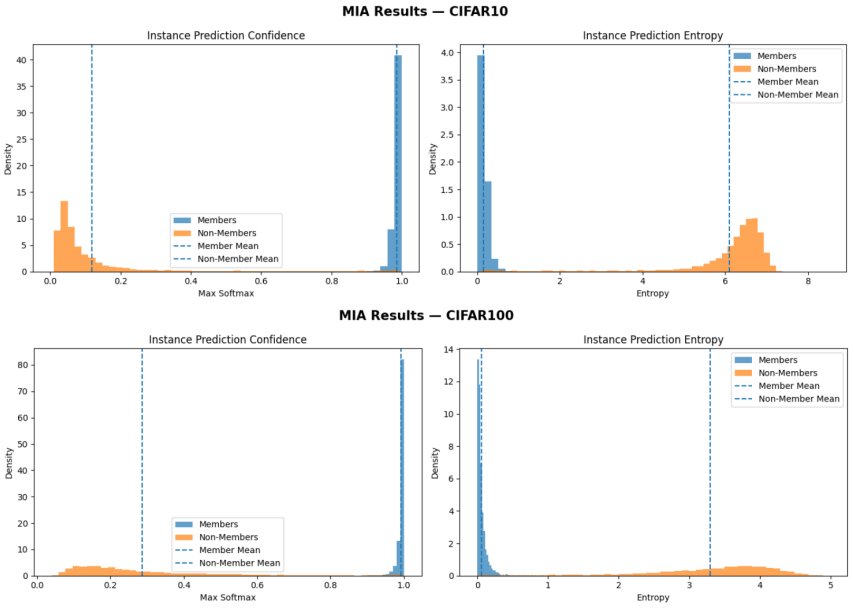


Figure 22: Same as above over CIFAR datasets.

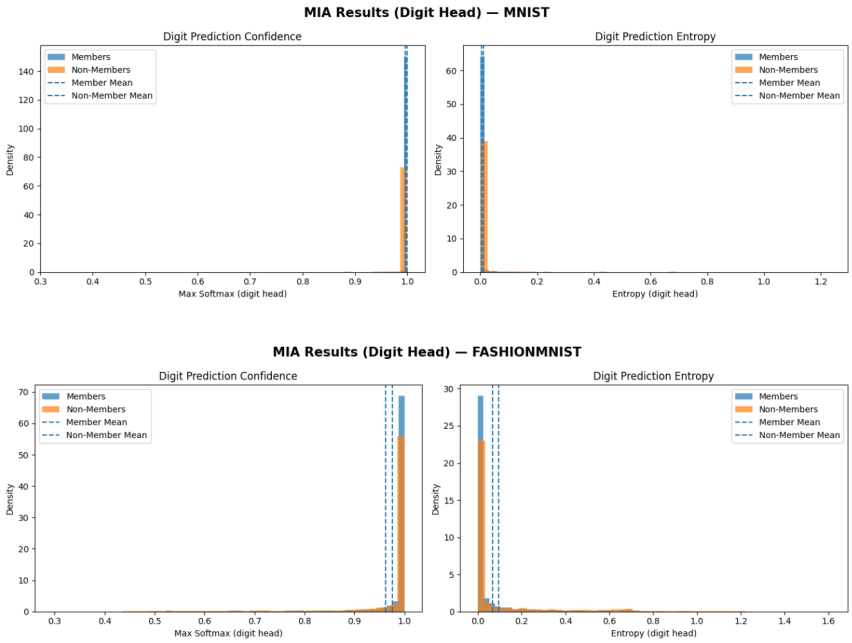


Figure 23: The left panel displays the distribution of maximum confidence scores, while the right panel shows the distribution of entropy, all from the **class branch** of a class-conditional two-branch network. Both distributions are plotted for 5K training samples (members) and 5K test samples (non-members).

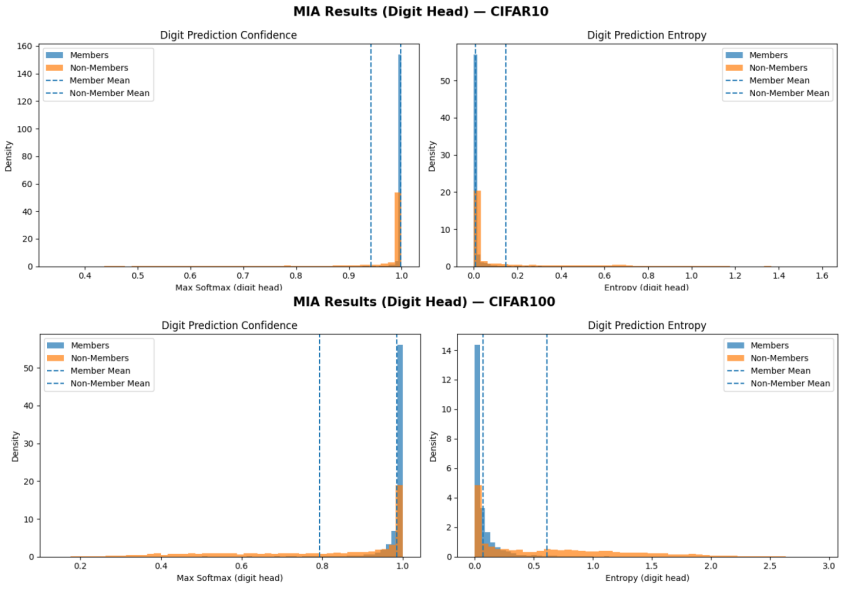


Figure 24: Same as above over CIFAR datasets.

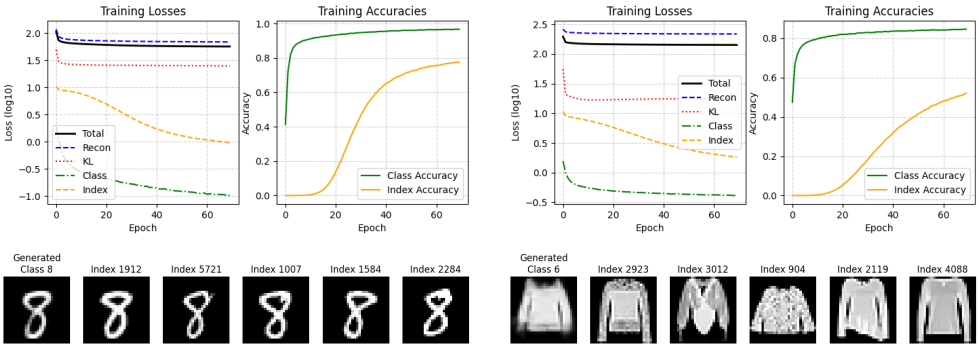


Figure 25: Image Generation with VAE. Top row: training losses and accuracies per epoch. Bottom row: generated samples alongside the top-5 predictions from the index network. Left column: MNIST; right column: FashionMNIST (latent_dim=128).

9.16 Extended Results: Image Generation

Section 7 introduces this application in the main body using the LFW face dataset (Figure 6). Here we give the MNIST/FashionMNIST validation that preceded it, at smaller scale; the full encoder/decoder architecture and training details are in Appx. 9.16.1.

Figure 25 shows loss curves, class and index prediction accuracies for MNIST and FashionMNIST, along with the five closest training samples retrieved by the index branch: the generated samples closely resemble their corresponding training examples, validating the architecture at smaller scale before the LFW extension in the main body.

9.16.1 Architecture and Training Details

Our model is a **Variational Autoencoder (VAE)** with an integrated classification module designed to jointly optimize for high-quality generative modeling and discriminative representation learning. The model has three main components: an **encoder**, a **latent reparameterization block**, and a **decoder with auxiliary heads**. Below, we describe each in detail.

9.16.2 Encoder

The encoder $f_\phi : \mathbb{R}^{1\times 28\times 28} \to \mathbb{R}^{2d}$ maps the input x to the parameters of a Gaussian distribution in a d-dimensional latent space (we use $d = 128$).

The encoder is implemented as a three-stage convolutional feature extractor followed by two parallel fully connected heads:

- **Conv Stage 1:** 3×3 convolution with 64 output channels, stride 2, and padding 1 $\to$ BatchNorm $\to$ ReLU. Reduces spatial resolution from 28×28 to 14×14.
- **Conv Stage 2:** 3×3 convolution with 128 channels, stride 2, padding 1 $\to$ BatchNorm $\to$ ReLU. Further reduces resolution to 7×7.
- **Conv Stage 3:** 3×3 convolution with 256 channels, stride 1, padding 1 $\to$ BatchNorm $\to$ ReLU. Maintains 7×7 spatial size while enriching representation depth.

The resulting $256 \times 7 \times 7$ feature map is flattened into a vector of size 12544 and passed through two linear layers:

$$\mu(x) = W_\mu h + b_\mu, \qquad \log \sigma^2(x) = W_{\log \sigma} h + b_{\log \sigma}.$$

9.16.3 Latent Reparameterization

We use the reparameterization trick to sample $z \sim q_\phi(z|x)$:

$$z = \mu(x) + \sigma(x) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

This allows gradients to propagate through stochastic sampling during training, enabling end-to-end optimization of both encoder and decoder parameters.

9.17 DECODER

The decoder $g_\theta : \mathbb{R}^d \to \mathbb{R}^{1\times 28\times 28}$ reconstructs the input image from z.

- **Fully Connected Projection:** The latent code is mapped back to a $256 \times 7 \times 7$ tensor.
- **Deconv Stage 1:** 3×3 transposed convolution to 128 channels, stride $1 \to$ BatchNorm $\to$ ReLU. (Resolution remains 7×7.)
- **Feature Pooling for Prediction:** We perform global average pooling over this intermediate 128-channel feature map, yielding $h_{\text{cls}} \in \mathbb{R}^{128}$. This vector is used for auxiliary prediction heads:
  - **Class Head:** A linear layer producing logits for K classes (we use $K = 10$).
  - **Index Heads (optional):** A list of linear layers, each predicting a separate categorical factor.
- **Deconv Stage 2:** 3×3 transposed convolution to 64 channels, stride 2, padding 1, output padding $1 \to$ BatchNorm $\to$ ReLU. (Upsamples to 14×14.)
- **Deconv Stage 3:** 3×3 transposed convolution to 32 channels, stride 2, padding 1, output padding $1 \to$ BatchNorm $\to$ ReLU. (Restores full 28×28 resolution.)
- **Output Layer:** 3×3 convolution producing a single channel, followed by a sigmoid nonlinearity to ensure pixel intensities lie in $[0, 1]$.

9.17.1 TRAINING SETUP

We train the model on the **IndexedMNIST** dataset, which augments the standard MNIST digits with class indices for auxiliary prediction tasks. Input images are normalized to $[0, 1]$. We use a batch size of 128 for training and 64 for evaluation.

9.17.2 OPTIMIZATION

The model parameters are optimized using the Adam optimizer with learning rate 10^{-3} for 70 epochs. For each minibatch, we compute:

- **Reconstruction loss:** Binary cross-entropy between input x and reconstruction $\hat{x}$.
- **KL divergence:** Regularizing the approximate posterior $q_\phi(z|x)$ toward the prior $p(z) = \mathcal{N}(0, I)$.
- **Classification loss:** Cross-entropy over the class logits.
- **Index loss (optional):** Cross-entropy over each auxiliary index head.

The total training objective is:

$$\mathcal{L} = \lambda_{\text{gen}}\Big(\underbrace{\mathbb{E}_{q_\phi(z|x)}[-\log p_\theta(x|z)]}_{\text{Reconstruction Loss}} + \underbrace{D_{\text{KL}}(q_\phi(z|x)\,\|\,p(z))}_{\text{KL Regularization}}\Big) + \lambda_{\text{cls}}\Big(\underbrace{\mathcal{L}_{\text{CE}}(y,\hat{y})}_{\text{Classification Loss}} + \underbrace{\sum_k \mathcal{L}_{\text{CE}}(y_k,\hat{y}_k)}_{\text{Index Losses}}\Big).$$

During training, we log the average total loss, its decomposition (reconstruction, KL, classification, index), and the classification and index prediction accuracies. This provides a clear picture of generative quality and discriminative performance over time.

PyTorch implementations are provided next.

```python
class Encoder(nn.Module):
    def __init__(self, latent_dim=64):
        super().__init__()
        self.enc = nn.Sequential(
            nn.Conv2d(1, 64, 3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(64, 128, 3, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 256, 3, stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
        )
        self.fc_mu = nn.Linear(256*7*7, latent_dim)
        self.fc_logvar = nn.Linear(256*7*7, latent_dim)

    def forward(self, x):
        x = self.enc(x)
        x = x.view(x.size(0), -1)
        mu = self.fc_mu(x)
        logvar = self.fc_logvar(x)
        return mu, logvar
```

Figure 26: Encoder of VAE on MNIST and FashionMNIST datasets

```python
class Decoder(nn.Module):
    def __init__(self, latent_dim=64):
        super().__init__()
        self.fc = nn.Linear(latent_dim, 256*7*7)
        self.up1 = nn.ConvTranspose2d(256, 128, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(128)
        self.up2 = nn.ConvTranspose2d(128, 64, 3, stride=2, padding=1, output_padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.up3 = nn.ConvTranspose2d(64, 32, 3, stride=2, padding=1, output_padding=1)
        self.bn3 = nn.BatchNorm2d(32)
        self.outc = nn.Conv2d(32, 1, 3, padding=1)

        self.pool = nn.AdaptiveAvgPool2d(1)
        self.class_head = nn.Linear(128, 10)
        self.index_heads = None

    def forward(self, z):
        x = self.fc(z).view(-1, 256, 7, 7)
        x = F.relu(self.bn1(self.up1(x)))
        h_cls = self.pool(x).flatten(1)
        class_logits = self.class_head(h_cls)
        index_logits = None
        if self.index_heads is not None:
            index_logits = [head(h_cls) for head in self.index_heads]
        x = F.relu(self.bn2(self.up2(x)))
        x = F.relu(self.bn3(self.up3(x)))
        recon = torch.sigmoid(self.outc(x))
        return recon, class_logits, index_logits
```

Figure 27: Decoder of VAE on MNIST and FashionMNIST datasets

```python
class VAE(nn.Module):
    def __init__(self, latent_dim=64, class_sizes=None):
        super().__init__()
        self.encoder = Encoder(latent_dim)
        self.decoder = Decoder(latent_dim)
        if class_sizes is not None:
            self.decoder.index_heads = nn.ModuleList([nn.Linear(128, sz) for sz in class_sizes])
        self.latent_dim = latent_dim

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std

    def forward(self, x):
        mu, logvar = self.encoder(x)
        z = self.reparameterize(mu, logvar)
        recon, class_logits, index_logits = self.decoder(z)
        return recon, mu, logvar, class_logits, index_logits
```

Figure 28: VAE Architecture on MNIST and FashionMNIST datasets

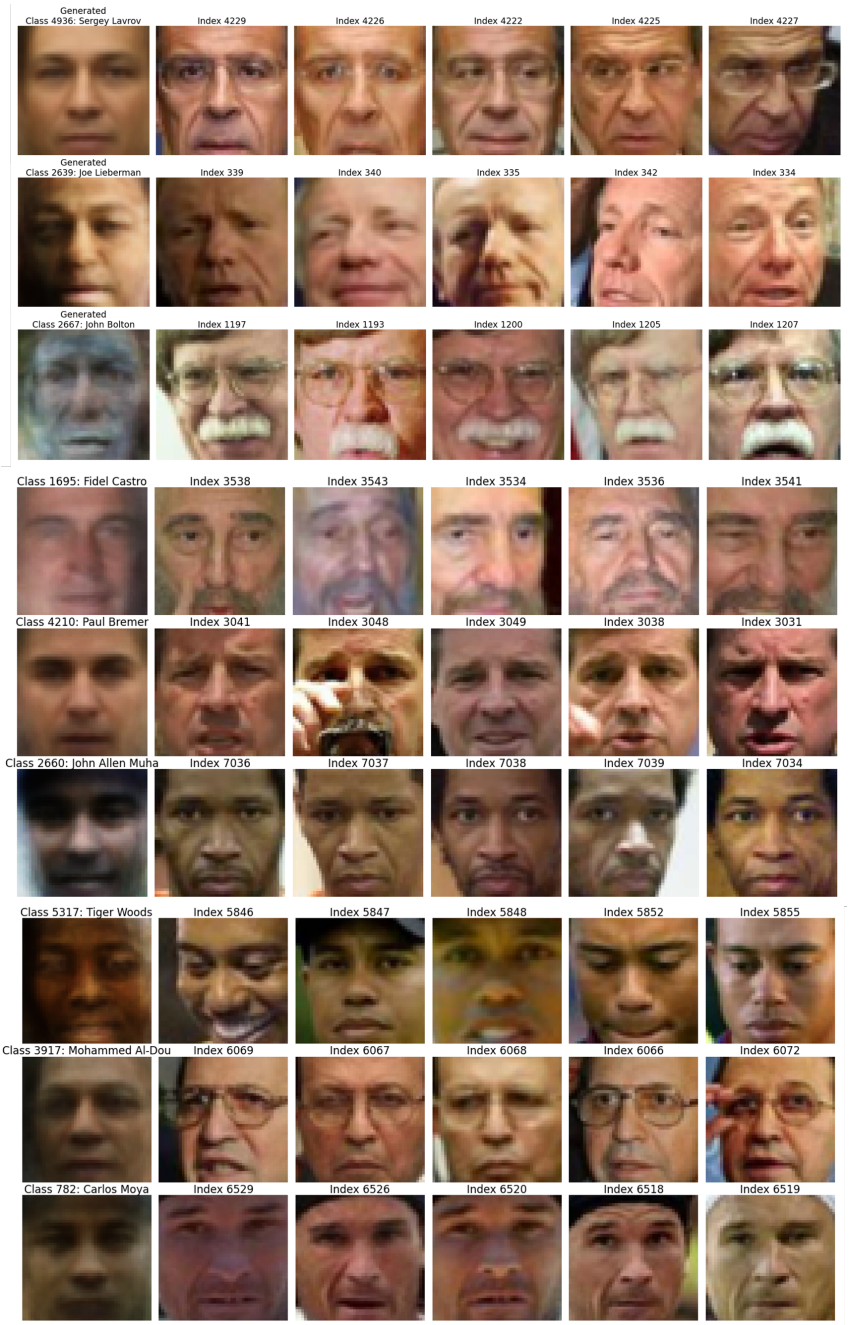


Figure 29: Additional generated faces by VAE (left column) along with five most similar training faces derived from the class and index branches.

9.17.3 ANALYZING THE RELATIONSHIP BETWEEN GENERATED IMAGE REALISM AND THE INDEX-PREDICTION BRANCH CONFIDENCE

We assessed the relationship between generated image realism and the index-prediction branch confidence. Specifically, we split generated samples by confidence (conf < 0.5 vs conf ≥ 0.5). To quantify the continuous association we report Spearman's rho between sample-level confidence and an individual-image quality proxy (Inception features distance to nearest training sample / reconstruction error). Across four runs, images from the high-confidence group had substantially lower FID than the low-confidence group:

Threshold	N_{high}	N_{low}	FID_{high} (95% CI)	FID_{low} (95% CI)	Spearman ρ (p)
0.5	432	568	28.7 (25.6–31.8)	65.4 (61.1–69.9)	-0.62 ($< 10^{-6}$)

Results show a consistent negative association: higher-confidence samples have substantially lower FID and smaller feature-space distances.

9.17.4 DATASET RECONSTRUCTION / NETWORK INVERSION (PRELIMINARY)

One of the intriguing aspects of provenance networks is that they implicitly encode information about the entire training dataset. This raises a natural question: can the original dataset be reconstructed from the network itself? If such reconstruction is feasible, it could enable valuable applications such as model retraining, data recovery, or transfer learning across related models. To explore this possibility, we fix the network weights and backpropagate gradients from each output index to the input space, iteratively updating the input to reconstruct the corresponding sample (Yosinski et al., 2015). This process is done in two phases:

Phase 1: Low-Resolution Optimization. The reconstruction process begins with a coarse low-resolution optimization designed to capture the global structure of the target concept while maintaining feature alignment and spatial smoothness. A small 14×14 image tensor $\mathbf{x}_s \in \mathbb{R}^{1\times1\times14\times14}$ is initialized with Gaussian noise and optimized for 15,000 iterations using the Adam optimizer with cosine-annealed learning rate decay. At each iteration, the image is bilinearly upsampled to 28×28 and passed through the pretrained classifier $f(\cdot)$ to compute the class logits. The objective combines three loss components:

$$\mathcal{L}_{\text{cls}} = \text{CrossEntropy}(f(\mathbf{x}), y_t), \tag{26}$$

$$\mathcal{L}_{\text{feat}} = \tfrac{1}{K}\sum_{k=1}^{K} \|\phi_k(\mathbf{x}) - \phi_k^*\|_2^2, \tag{27}$$

$$\mathcal{L}_{\text{tv}} = \sum_{i,j}(|x_{i,j+1} - x_{i,j}| + |x_{i+1,j} - x_{i,j}|), \tag{28}$$

corresponding to classification, feature-matching, and total variation (TV) regularization, respectively. The total loss is

$$\mathcal{L}_{\text{total}} = w_{\text{cls}}\mathcal{L}_{\text{cls}} + w_{\text{feat}}\mathcal{L}_{\text{feat}} + w_{\text{tv}}\mathcal{L}_{\text{tv}}, \tag{29}$$

where $w_{\text{cls}} = 15.0$, w_{feat} increases linearly from 0.5 to 2.0 over iterations, and $w_{\text{tv}} = 3 \times 10^{-5}$. Gradients are clipped to $\|\nabla \mathbf{x}_s\| \leq 1.0$, and pixel intensities are clamped to $[-3, 3]$ after each update. This phase produces a smooth, low-frequency prototype that roughly captures the spatial layout of the target class.

Phase 2: High-Resolution Refinement. The second phase refines the image at full 28×28 resolution to enhance fine-grained structure and texture. The upsampled output from Phase 1 initializes this stage, which continues optimization for an additional N_2 iterations using the same loss formulation (Eq. 4) but with dynamically adjusted weights to shift the balance from classification accuracy toward perceptual fidelity:

$$(w_{\text{cls}}, w_{\text{feat}}, w_{\text{tv}}) = \begin{cases} (10.0,\ 3.0,\ 0.0001 + 0.0001p), & p < 0.5, \\ (8.0,\ 5.0,\ 0.0002 + 0.0002(p - 0.5)), & p \geq 0.5, \end{cases}$$

where $p \in [0, 1]$ denotes normalized progress through Phase 2. This adaptive schedule encourages sharper spatial details and improved layerwise feature consistency as optimization proceeds. As before, gradients are clipped and pixel values clamped to stabilize training. Together, the two stages form a coarse-to-fine synthesis pipeline that progressively reconstructs semantically and perceptually plausible images corresponding to the target class.

For our experiments, we utilized a single-branch provenance network trained on the MNIST dataset with approximately 50% memorization capacity. Representative reconstructed samples are shown in Fig. 30. Although the reconstructions are not perfectly clear, they nonetheless exhibit recognizable patterns resembling the original inputs. Interestingly, our analysis indicates that moderate memorization levels (e.g., 10%) yield more faithful reconstructions compared to highly memorized networks (e.g., 100%), suggesting that excessive memorization may reduce the network's ability to generalize and reconstruct diverse inputs.

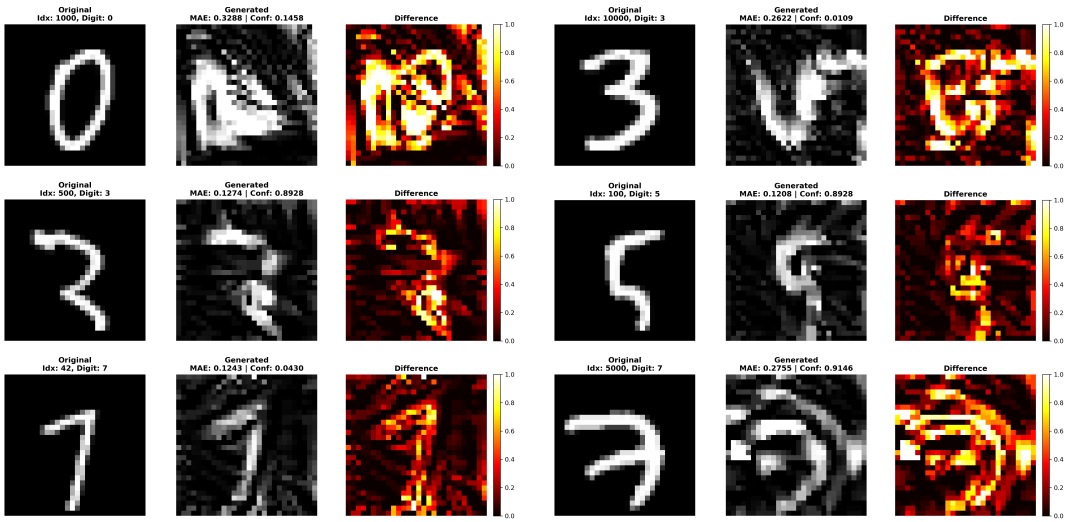


Figure 30: Results of network inversion for a single branch on the MNIST dataset with 50% memorization. The columns, from left to right, show: (1) the original training data point, (2) the reconstructed or generated image obtained by activating the corresponding index neuron, and (3) the difference between the original and reconstructed images.