Direct Neural Network Training on Encrypted Datasets

Alexander C Flint
Sydney, Australia
alexander_flint@alumni.brown.edu

9 October 2026

Abstract

The vast majority of the world's data is not available to be used for training neural networks because of concerns about privacy and protecting data value. The Tensorlock method, described here, transforms training, test, and inference plaintext vector examples into encrypted vector examples of higher dimension using padding (increasing vector dimensions with added values in additional dimensions), perturbation (adding noise to values across dimensions), fixed index shuffling, and random orthogonal matrix transformation. The resulting encryption process enables artificial neural network training and inference directly on the encrypted data. Training on datasets transformed by aggregate padding, perturbation, fixed index shuffling, and orthogonal transformation results in only slightly lower validation accuracies than those seen with unencrypted controls, with similar training time. Modeled attacks produced poor recovery in the absence of highly specific knowledge of plaintext covariance structure. Without such knowledge, inversion requires exhaustive search over an enormous hypothesis and transformation space. By enabling training and inference on encrypted data, Tensorlock maintains data privacy and also protects the business value and underlying intellectual property of data in machine learning.

Keywords: secure machine learning, secure artificial intelligence, data security, encrypted training data, encrypted test data, encrypted inference, non-orthogonal transformation, artificial neural networks, data value, intellectual property.

1 Introduction

Current AI models have been largely trained on publicly accessible data, but only a tiny fraction of the world's data is publicly accessible, raising concerns that scaling models will "run out of data" in the near future [Villalobos et al., 2024, Roded and Slattery, Hidary, 2025]. While use of machine-created synthetic data has been proposed as a solution to this problem [Villalobos et al., 2024, Jones, 2024, OpenAI], this approach cannot replace the intrinsic value of well-curated private data such as enterprise documents, financial data, or electronic health records [Theodos and Sittig, 2020, Phillips, 2018, Aubinais et al., 2025, Henderson et al., 2023]. These domains would greatly benefit from advances in machine intelligence, but maintaining data security requires new technologies that enable direct machine learning training and inference on encrypted datasets.

An ideal method for securing machine learning systems would encrypt training, test, and inference data to allow the direct use of existing architectures, with minimal loss of accuracy and comparable training times. Such an implementation would also require encrypting test and inference examples identically, thereby maintaining the business value of training data and protecting intellectual property in source data.

Here, the performance of Tensorlock, an encryption technique for direct use with neural networks [Flint, 2026], is described and tested. The method combines padding, perturbation, fixed index shuffling, and random orthogonal transformation to create an aggregate transformation applied across all training, test, and inference data. Neural networks are trained directly on the encrypted data—no decryption step is performed prior to training or inference. We find that network training on data encrypted by this aggregate transformation achieves a level of validation accuracy that is similar to results obtained on unencrypted datasets, with similar training times. Direct inversion of the transformation involves prohibitively large transformation and coordinate hypothesis spaces in the absence of very specific knowledge of plaintext covariance structure.

2 Threat model

The novel encryption transformations described in detail below are designed to protect data privacy in the context of a specific threat model, shown in Figure 1.

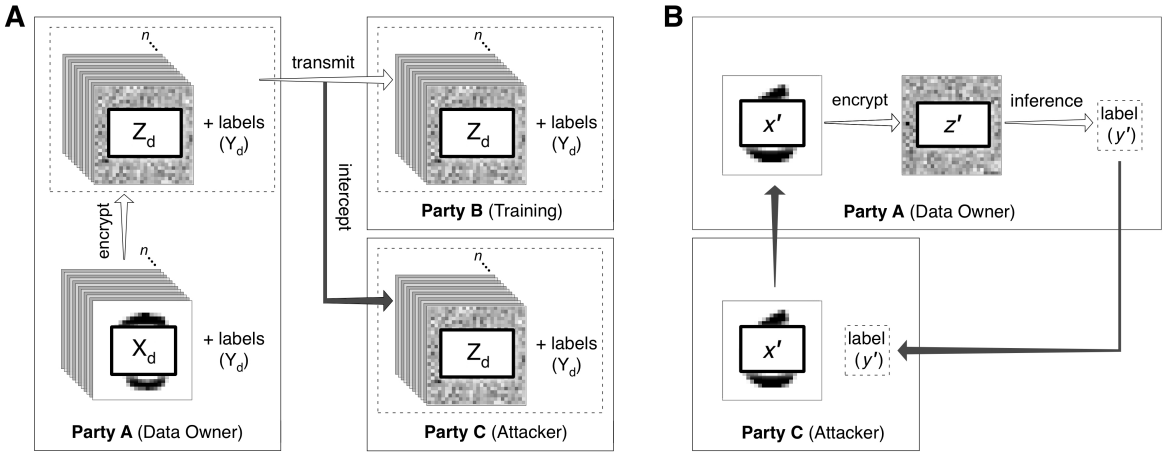

Figure 1: Threat model.
A. *Training stage.* Party A (the data owner) possesses the original dataset (Xd) and a corresponding set of labels (Yd). Party A generates an encrypted dataset (Zd), which is transmitted along with Yd to Party B for model training. Party C (attacker) intercepts transmission of Zd and Yd. The attacker thus possesses an encrypted dataset (Zd) and labels (Yd) but no original data (Xd).
B. *Inference stage.* Party C (attacker) submits an original (unencrypted) example (x') that is *not* a member of the original dataset (Xd) from training to an *inference oracle* controlled by Party A. The Party A (data owner) inference oracle receives (x') and *privately* encrypts it to (z') with the same algorithm used during training. Inference performed on (z') using the trained neural network yields label (y'), which is returned to Party C. Only the inference result label (y') is transmitted from the inference oracle back to Party C, and after inference is performed, the encrypted example created in the inference oracle is then deleted (z').

As illustrated in Figure 1A, a data owner (Party A) possesses an original dataset (Xd) with n examples and a corresponding set of n labels (Yd). Party A applies the set of transformations described below to generate an encrypted dataset (Zd) with n examples and an unchanged set of n labels (Yd). Party A transmits Zd and Yd to Party B for performance of ANN training, and Party B uses Zd and Yd to train a neural network (Figure 1A).

Party C intercepts Zd and Yd during transmission (or because Party C is part of Party B). As

shown in Figure 1B, during inference, an attacker (Party C) may present, as a user of software designed to perform inference, an original data example (x') to Party A's *inference oracle*. The inference oracle then privately encrypts this example (x') using the same encryption process used for training to yield an encrypted example (z'). Because Party C does not have access to original examples in the training dataset (Xd), this original example (x') presented by Party C is *not* a member of Xd and must be generated *a priori* for this purpose by Party C without knowledge of the original examples in Xd. Inside the inference oracle, the encrypted example (z') is presented to the trained neural network (obtained from Party B after the training process) to yield a label category prediction (y'). The inference oracle returns the label category prediction (y') to the user (in this scenario, the attacker, Party C), *without disclosing the encrypted example* (z'), and the inference oracle deletes the encrypted example (z') after inference is performed.

3 Description of the encryption methods

The central method of the encryption approach presented here is the combination of one or more non-orthogonal transformations (which are used to increase vector dimensions by an amount unknown to a potential attacker and add noise) with one or more random orthogonal transformations (which rotate vectors in multidimensional space).

For a tensor of rank N, sequential non-orthogonal transformations (padding and/or perturbation) are applied to the input tensor, followed by a random orthogonal transformation, which can be an example-wise fixed index shuffling transformation, a random orthogonal matrix transformation, or both. All tensors in a given project's training and test sets, as well as any inference examples, are transformed in the same fashion.

For the purpose of testing and security analysis, we present encryption consisting of four transformations applied serially: padding to increase vector dimensions, perturbation to add noise across values in all dimensions, fixed index shuffling, and random orthogonal matrix transformation. While each of these transformations may be applied to tensors of arbitrary rank with any data type suitable for any type of machine learning operation, in this paper, method illustrations and performance experiments will use supervised training of fully-connected artificial neural networks (ANNs) on the MNIST dataset of grayscale (n,28,28) handwritten digits [LeCun et al.] as an exemplar, with additional analyses performed on Fashion-MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky], EMNIST Digits [Cohen et al., 2017], and EMNIST Letters [Cohen et al., 2017].

3.1 Padding

Padding of a dataset involves adding data in additional dimensions to input vectors such that the examples of the source tensor are increased in size (increasing the vector dimensions of the flattened examples), with resulting examples of uniform shape across the resulting tensor. In the particular case of 2D image data, random padding may represent one or more rows of data (Figure 2A), or one or more columns of data, a combination of one or more rows and columns, or may be an addition of any number of vector dimensions with values in fixed locations inside or outside of the examples: e.g., interlaced padding (Figure 2B). In the present work, a different pad of data is applied to every example, with different pads generated for every example across the training, test, and inference tensors. The padding values may be randomly sampled from a uniform distribution or from any other distribution, or may be algorithmically determined from the source plaintext data.

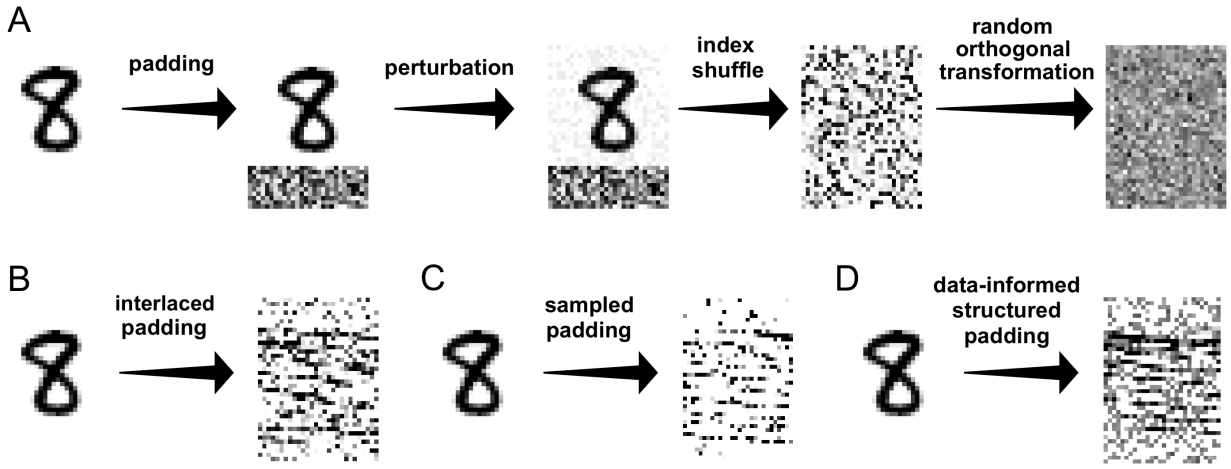

Figure 2: Serial transformations performed prior to network training.
A. Illustration of a sequence of padding, perturbation, index shuffling, and random orthogonal matrix transformation. An example from an original plaintext dataset (Xd), the MNIST dataset with n examples in a n,28,28) tensor. Padding as illustrated here is a 10-row pad of randomly chosen pixels, with a different random pad appended to each of the examples [a random (n,10,28) pad appended to the original (n,28,28) tensor]. The padded example is then subjected to perturbation at the pixel level, in this case, uniform random adjustment from the range of -5% to +5%). An index shuffle is then applied to all examples, with the same index shuffle bijection applied to each of the n examples. A random orthogonal matrix transformation is then applied by dotting all examples by the same random orthogonal matrix of appropriate size (e.g. for a 38x28 input, a [(38x28) x (38x28)] = [1064 x 1064] random orthogonal matrix).
B. Interlaced padding. Instead of placing the random padding pixels to the end of the examples, the random pixels can be appended/inserted in the same randomly chosen locations in the input vectors.
C. Sampled padding. Padding values may be sampled from the distribution of values in the source plaintext data to help obscure the distinction between padding and plaintext dimensions. Here, random sampled padding values are applied in interlaced form—note the difference in pixel values when compared to the uniform interlaced values illustrated in (B).
D. Data-informed structured padding. The choice of padding values can also be determined algorithmically based on the plaintext data structure to further obscure the distinction between padding and plaintext dimensions. One such type of data-informed structured padding, principal component analysis-score bootstrap padding, is illustrated here in interlaced form.

3.2 Perturbation

Perturbation of a dataset involves applying a set of functions that alter the value in each dimension in the examples of an input tensor. The standard version of perturbation illustrated here (Figure 2A) is an alteration of the pixel values by different randomly chosen percentages of the underlying value range in each dimension.

First, an array of random values from a chosen minimum to a chosen maximum (e.g., min = -5% to max = +5%) is generated, where the array size is matched to the size of an example in the input tensor. In the experiments shown in this paper, 8-bit source data are converted to 64-bit floating-point precision [range 0.0 to 1.0] to enable graded changes between the original 8-bit values. For each example vector, the value in a given dimension is perturbed by the randomly chosen percentage amount at the corresponding position in the perturbation array. Different perturbation arrays are applied to each of the examples of the training tensor, test tensor, and any inference examples.

As pixel values are randomly perturbed upward or downward, perturbation beyond the relevant minimum and maximum values must be explicitly handled. Two approaches used here are *clipping* and *reflection*. Clipping assigns the minimum or maximum value to any pixel values brought beyond these values by the perturbation algorithm. For example, if a pixel value on a float scale from 0.0 to 1.0 started at 0.99 and would be brought to a value of 1.03 by a perturbation, clipping changes the resulting value to 1.0. Reflection brings the value to a corresponding amount within the min/max boundaries. For example, if a pixel value on a float scale from 0.0 to 1.0 started at 0.99 and would be brought to 1.03 by a perturbation, the amount exceeding the maximum value is (1.03 - 1.0 = 0.03) and the assigned value using reflection would be (1.0 - 0.03 = 0.97). Figure 2A illustrates the effect of applying a -5% to +5% bounded percentage perturbation with clipping to the examples of an input tensor. Throughout this paper, the setting for handling perturbations beyond the maximum and minimum bounds will be clipping unless otherwise stated.

Bounded perturbation as above can be sampled from a uniform or any other distribution, or based on a measured distribution of the plaintext data structure or local plaintext data structure. Perturbation can be produced by any function or functions applied to the values in each dimension in each example.

3.3 Orthogonal transformations

Two types of orthogonal transformation are used here in combination with the transformations of padding and perturbation described above. In *fixed index shuffling*, the index position of pixels of input vectors are shuffled, with the same index shuffling applied to each of the examples. In *random orthogonal matrix transformation*, a random orthogonal matrix of appropriate size is used to dot input vectors.

3.3.1 Fixed index shuffling

Fixed index shuffling of a dataset involves shuffling the pixel index locations of the examples of an input tensor such that the same shuffling rearrangement is applied in identical fashion to each example in the tensor. This represents a bijection of dimensions of flattened example vectors, and is thus a subset of orthogonal matrix transformation for which there are, for n-dimensional vectors, $n!$ possible bijections. Fixed index shuffling can be accomplished by randomly generating a shuffling index array that matches the shape of an example in the input tensor, which stores the instructions for rearranging the pixel locations in each example, or by dotting example vectors with a square

random binary permutation matrix. The same fixed shuffling transformation is applied to every example of the training tensor and the test tensor, as well as to any inference examples.

3.3.2 Random orthogonal matrix transformation

To transform input vectors by random orthogonal transformation, a random orthogonal matrix is generated by random choice of an orthogonal matrix of appropriate shape drawn from the infinite O(N) Haar distribution. Once a random orthogonal matrix is generated, it is used to transform input vectors by taking the (vector · matrix) dot product of the input vectors and the stored random orthogonal matrix. The same random orthogonal matrix is used to transform all training, test, and inference data in the same fashion.

3.4 Serial combinations of transformations

The transformations of padding, perturbation, shuffling and orthogonal transformation may be performed sequentially, in varying combinations. At least one random orthogonal transformation step is used, with an orthogonal transformation step deployed at the final stage of the chain of transformations, as illustrated in Figure 2A. Multiple rounds of functions may be applied in any order, including multiple versions of each category of transformation. Combining the non-orthogonal transformation steps of padding, perturbation, or a combination of these with any number of orthogonal transformations creates an aggregate non-orthogonal transformation. The resulting encrypted tensors may be left at 64-bit precision (float64) or alternatively cast to 32-bit or 16-bit precision.

4 Methods

4.1 Experimental setup

Training performance on data transformed by the encryption steps of padding, perturbation, fixed index shuffling, and random orthogonal transformation, alone and in combination, was tested by training and validation with the MNIST dataset [LeCun et al.], with a 60K training and 10K validation split. Additional analyses were performed with similar network structures using additional datasets [Fashion-MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky], EMNIST Digits [Cohen et al., 2017], and EMNIST Letters [Cohen et al., 2017]], with minor modifications to network structure as described in the Results. Training and validation were performed using the same fully-connected ANN structure and hyperparameters for all trained models, using PyTorch in Python 3.12 on Google Colab Pro, running on an NVIDIA A100 GPU. Fully-connected ANN models had (1) an input layer corresponding to the example data shape with or without change in shape produced by random padding, (2) two fully-connected hidden layers of 512 and 256 neurons with ReLU activation and 0.2 Dropout at each layer, and (3) an output layer corresponding to the labels of the dataset. The loss function was cross-entropy and the optimizer was Adam, with a learning rate of 0.001. Training batch size was 256, with 20 training epochs for all training runs. To facilitate appropriate comparisons of validation accuracy and training times, model parameters were fixed for all experiments and not individually optimized—the only difference from model to model was any required increase in input shape produced by random padding. To achieve appropriate precision of fixed perturbation transformations, all datasets were converted to floating point precision.

Each transformation process that was tested, whether individual steps or multiple steps in different combinations, was performed a total of 15 separate times (see *sample size calculations*

below), with separate fully-connected ANN models trained for each run. For example, to test orthogonal transformation alone, as shown in Figure 3D, a new random orthogonal matrix was generated to transform the training and validation splits of MNIST, and the model then underwent training. This was repeated 14 more times, each time deleting all model variables, generating a new random orthogonal matrix, transforming the tensors *de novo*, and training a new model at each run. During the same session, while connected to the same GPU, control models were independently trained without any transformation steps. Training runs between transformed and control datasets were performed in random order. For training time experiments, elapsed time was measured from the start of the first training epoch to the end of the last training epoch. For experiments examining varying floating-point precision and model accuracy, each of 15 transformations was recast and rerun from float64 to float32 and to float16, with float32 computation in the ANN networks throughout, to experimentally isolate ciphertext storage precision.

4.2 Statistics and sample size calculation

For comparison of validation accuracy and training times between controls and a given transformation process, the nonparametric Kruskal-Wallis equality-of-populations rank test was performed, with P value for the Chi-square test without ties reported for each comparison (similar results were obtained with the parametric Student's t-test). For matched runs between float64, float32, and float16 transformed datasets, one-way repeated measures ANOVA was used. For histogram analysis of coordinate value distribution in original and encrypted datasets, examples were flattened, with floating-point precision, scaled to 0.0 to 1.0, and histograms were generated with numpy.histogram in Python. Fitting of histogram data to the Gaussian distribution was performed NumPy and SciPy, with goodness-of-fit reported with the Kolmogorov-Smirnov statistic using SciPy, where higher P values indicate better fit to the Gaussian distribution.

Per-example agreement between encrypted and control models was quantified using correctness agreement, Jaccard similarity of correct sets, and the Q statistic. Correctness agreement was defined as the proportion of examples for which both models were either correct or incorrect. Jaccard similarity was calculated as the intersection-over-union of the sets of correctly classified examples. The Q statistic was used as a pairwise measure of classifier association based on concordant and discordant correctness outcomes, with larger values indicating more similar per-example prediction patterns.

For illustration of mixing across encrypted dimensions, random vectors were generated in NumPy, and vector word representations were generated from a dictionary of English words (https://github.com/comp-core/Think-Python/blob/master/words.txt).

Sample size power analysis was performed for two-sample means using a series of 20 control runs (unencrypted MNIST) on the same ANN, which showed a standard deviation of the validation accuracy of 0.001, which yields a very small sample size because of the extremely tight distribution of results. Conservatively assuming a standard deviation 50x larger than this (0.05) for both groups yields a sample size estimation of 10 model trainings needed per group (control vs. encrypted) to detect an absolute difference in accuracy of 0.007 with power of 0.8 and alpha of 0.05. Keeping the very low observed standard deviation for the control group (0.001) but conservatively assuming a 50x higher standard deviation for the encrypted group (0.05) for the same detection of a 0.007 difference in accuracy yields a sample size estimation of 7 model trainings per group. Based on the above calculations, 15 training runs per group were used.

Statistical tests and sample size calculations were performed with Stata/MP 16.1 (StataCorp, College Station, TX) and SciPy. Illustrations and graphs were generated using the Python libraries Seaborn and Matplotlib in Google Colab Pro.

5 Network training results

5.1 Single transformations

Figure 3 shows model training performance for each of the functions described above (padding, perturbation, fixed index shuffle, and random orthogonal transformation) applied as separate single-step transformations and compared to training with unencrypted MNIST (control). For each run, new models were trained on newly transformed datasets using new transformations. For padding applied alone, a very small but statistically significant difference is seen. For the other transformations applied individually, no difference in validation accuracy was seen between models trained on transformed data vs. controls (Figure 3, A-D).

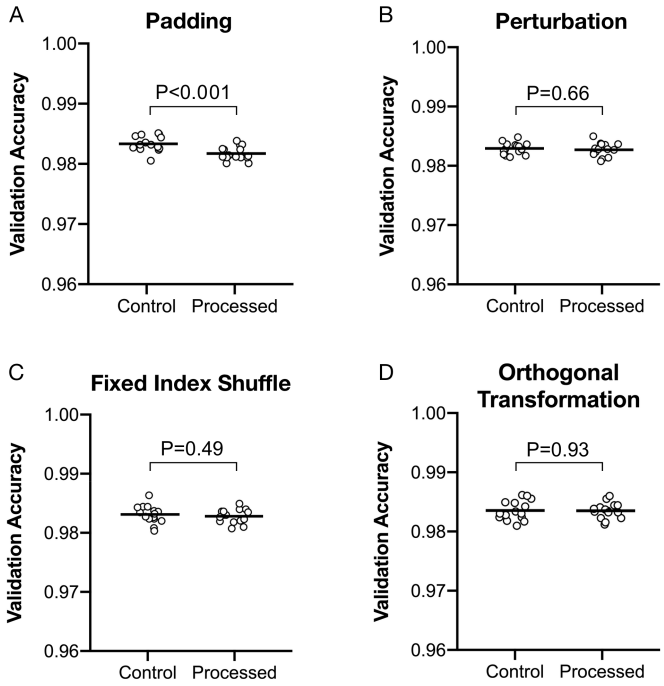

Figure 3: Single transformations.
A. Model training on padded data (a different random (2,28) pad for each example). Mean validation accuracy was 0.9818 ± 0.0009 compared with 0.9833 ± 0.0010 for control (P<0.001).
B. Model training after perturbation (a different array of random perturbations applied to each example, between min = -5% and max = +5%). Mean validation accuracy was 0.9827 ± 0.0009 (Processed) compared with 0.9829 ± 0.0008 (Control) (P=0.66).
C. Model training after fixed index shuffling. Mean validation accuracy was 0.9827 ± 0.0009 (Processed) compared with 0.9830 ± 0.0012 (Control) (P=0.49).
D. Model training after random orthogonal matrix transformation. Mean validation accuracy was 0.9835 ± 0.0011 (Processed) compared with 0.9835 ± 0.0013 (Control) (P=0.93).
Note that all y-axis scales run from 0.96 to 1.00.

5.2 Multiple transformations

The Tensorlock encryption method involves sequential application of padding, perturbation, fixed index shuffling, and random orthogonal transformation to yield an aggregate transformation. We

next tested the training performance of data encrypted with the particular sequence of padding → perturbation → fixed index shuffle → random orthogonal transformation.

With a combined transformation of random padding (2,28), random perturbation (min = -5% to max = +5%), fixed index shuffle, and random orthogonal matrix transformation, a small but statistically significant decrement in validation accuracy (0.9819 ± 0.0011) is seen when compared to controls (0.9831 ± 0.0009) (P<0.001) (Figure 4A). The absolute difference in validation accuracy between transformed datasets and control was 0.0012 and the relative percentage difference was 0.12%.

No difference in validation accuracy was seen between appended random padding (0.9821 ± 0.0011) and interlaced random padding (0.9820 ± 0.0014) in the combined transformations (P=0.72).

5.3 Training time

Training times were tested to compare computation time for control unencrypted MNIST with the same model specification trained using MNIST transformed with random padding (2,28), random perturbation (min = -5% to max = +5%), fixed index shuffle, and random orthogonal matrix transformation. A very small but statistically significant increase in training times was seen with models trained on encrypted data (18.567 ± 0.166 sec) compared to models trained on control MNIST (18.271 ± 0.162 sec), an absolute difference of 0.296 sec per model training, or a relative increase of 1.62% from control (Figure 4B).

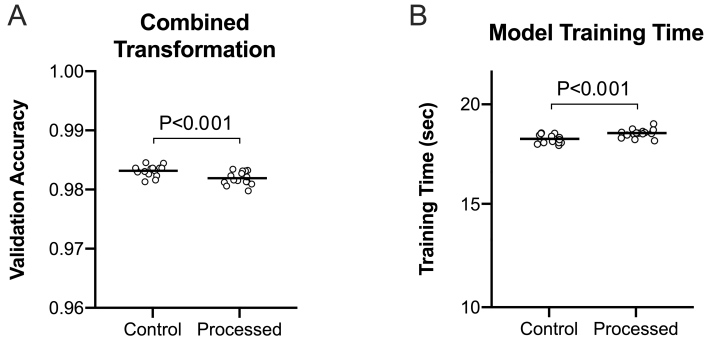

Figure 4: Multiple transformations.
A. Model training on data encrypted with the combined transformation of random padding (2,28), random perturbation (min = -5% to max = +5%), fixed index shuffling, and orthogonal transformation. Mean validation accuracy was 0.9819 ± 0.0011 compared with 0.9831 ± 0.0009 for control (P<0.001)
B. Model training time. Training time for models trained on data encrypted by combined transformation was 18.567 ± 0.166 sec compared to 18.271 ± 0.162 sec for models trained on control MNIST (P<0.001).

5.4 Floating point precision of the encrypted result

Reducing the floating point precision of the encrypted result after combined transformation of random padding (2,28), random perturbation (min = -5% to max = +5%), fixed index shuffling, and orthogonal transformation did not reduce model accuracy. Validation accuracy was (0.9819 ± 0.0010) for the original float64 data (training tensor 384.52 MB in memory), (0.9819 ± 0.0010) for float32 data (192.26 MB in memory), and (0.9821 ± 0.0011) for float16 data (96.13 MB in memory) (P=0.68).

5.5 Extent of padding and perturbation

The impact of varying the extent of added padding dimensions on validation accuracy was tested in the context of padding alone and in combination with multiple transformations (padding, perturbation (min = -5% to max = +5%), fixed index shuffling, and orthogonal transformation). Figure 5 shows the results of increasing pad size from 0 rows (control) to 10 rows (10,28) of padding. For both padding alone and padding in the context of multiple transformations, increasing size of the appended pad produced a small decrement in validation accuracy across the range tested (Figure 5A).

Similarly, when the minimum to maximum random perturbation range is adjusted, increasing the extent of perturbation produced a small decrease in validation accuracy across the range tested for both perturbation applied alone and in combination with multiple transformations (Figure 5B).

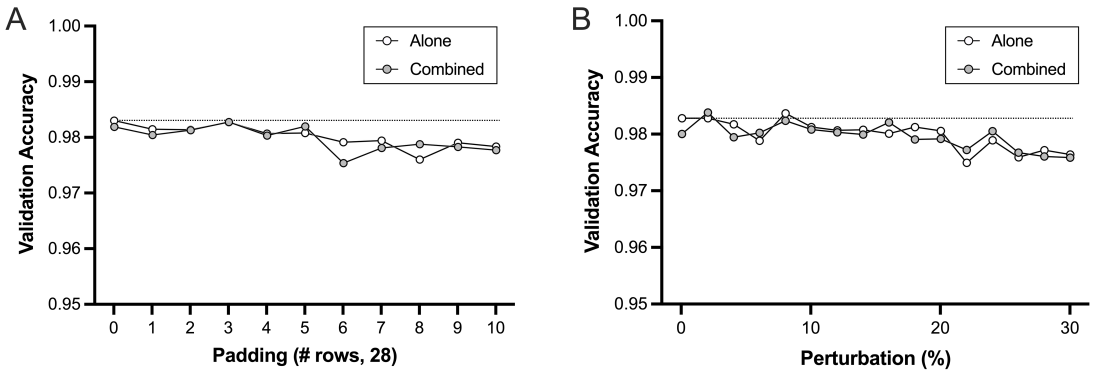

Figure 5: Variation of padding and perturbation.
A. Variation of number of padding rows from control (0 rows) to 10 rows (10,28). For both single transformation by random padding (Alone, open circles) and padding in the context of multiple transformations (padding, perturbation, fixed index shuffle, and orthogonal transformation; Combined, closed circles), a very small decrement in validation accuracy is seen across the range of padding tested.
B. Variation in the extent of perturbation. Random perturbation is varied by the extent of the possible minimum and maximum % range of perturbation (for example, 5% perturbation refers to min = -5% to max = +5%) In both single transformation by random perturbation (Alone, open circles) and in the context of multiple transformations (Combined, closed circles), increasing random perturbation shows a very small decrement across the range tested.

5.6 Testing on other data sources

To explore whether similar results may be obtained with datasets known to be more challenging than MNIST, additional experiments tested ANN training on encrypted and control versions of Fashion-MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky], EMNIST Digits [Cohen et al., 2017], and EMNIST Letters [Cohen et al., 2017]. Encryption used on Fashion-MNIST, CIFAR-10, EMNIST Digits, and EMNIST Letters was 2-row random padding, random perturbation (-5% to +5%), fixed shuffling, and random orthogonal matrix transformation (the same transformation applied to MNIST, above). Training on the Fashion-MNIST database yielded the following validation accuracies: control 0.8855 ± 0.0031, encrypted 0.8823 ± 0.0033 (P=0.009). Training on CIFAR-10 yielded: control 0.5305 ± 0.0035, encrypted 0.5078 ± 0.0045 (P<0.001). The typical approach to

obtain better accuracies with datasets like CIFAR-10 involves the use of CNN structures. Modified CNN structures, including pre-encryption spatial feature extraction and oracle-mediated spatial layer processing, can be integrated with the Tensorlock encryption method [Flint, 2026], but are beyond the scope of this initial report. (White papers on Tensorlock adaptation to CNNs, LLM, GANs, and other network structures can be found at tensorlock.ai/documents.) Training on EMNIST Digits yielded: control 0.9913 ± 0.0004, encrypted 0.9911 ± 0.0004 (P=0.16), and on EMNIST Letters yielded: control 0.9241 ± 0.0012, encrypted 0.9213 ± 0.0016 (P<0.001). The absolute difference in validation accuracy between encrypted and control training for each dataset is shown in Table 1 (left column). Similar results are found with 'standard padding and perturbation' (uniform random padding and perturbation, as illustrated in Figure 2A) and 'advanced padding and perturbation' (as illustrated in Figure 2D). The right column in Table 1 shows the accuracy difference from control with datasets encrypted with source-derived PCA-score bootstrap padding and spectrally optimized nonlinear perturbation with a utility-preserving penalty (see Section 6.6.4 for details).

Table 1: Absolute accuracy difference across datasets

	Absolute accuracy difference	
Dataset	**Standard padding / perturbation**	**Advanced padding / perturbation**
MNIST	0.0012	0.0010
Fashion-MNIST	0.0032	0.0017
CIFAR-10	0.0227	0.0257
EMNIST Digits	0.0002	0.0005
EMNIST Letters	0.0028	0.0011

At the per-example level, models trained on encrypted and control datasets showed high rates of agreement. For example, with MNIST encrypted with advanced padding and perturbation as shown in Table 1, correctness agreement was 0.9873, Jaccard similarity of correct sets was 0.9872, and the Q statistic was 0.9923.

6 Security analysis

As described above, our threat model consists of a potential attacker in possession of an encrypted training dataset (Zd) with labels (Yd). The attacker can also send unencrypted attacker-generated examples (x'_a, x'_b, $\cdots$) *not* present in the original dataset (Xd) that can be used to obtain proposed inference labels (y'_a, y'_b, $\cdots$) from an inference oracle (Figure 1).

The Tensorlock privacy scheme is specifically designed to protect against an attacker using these elements to reverse our encryption function $f(Xd) \rightarrow Zd$ by discovering the inverse function $f^{-1}(Zd) \rightarrow Xd$ while intentionally preserving geometric relationships needed for neural-network training. To achieve this, the Tensorlock transformation changes the coordinate representation of the input data. Padding introduces additional dimensions, fixed index shuffling changes the locations of the latent coordinates, and random orthogonal transformation mixes these coordinates across the encrypted vector space.

While the geometric relationships among padded vectors are preserved by orthogonal transformation, the secret Haar-random orthogonal matrix changes the basis in which the data are represented, thereby obscuring the direct correspondence between individual encrypted coordinates and the original plaintext coordinates.

The following sections consider explicitly what information is preserved by the transformation, what remains hidden from an attacker, and under what conditions the original plaintext representation might be recovered.

6.1 Coordinate mixing under the transformation

Random orthogonal matrix transformation rotates our padded vectors in D-dimensional space, where D is the total number of padded dimensions, derived from the original d plaintext dimensions and m padding dimensions ($D = d + m$). The process of padding, index shuffling, and orthogonal transformation distributes plaintext information across the encrypted coordinate system, mixing the components of plaintext vectors across the components of transformed vectors.

For a random orthogonal matrix Q, each encrypted coordinate is therefore formed from many small contributions distributed across the D pre-transform coordinates. This mixing tends to produce approximately Gaussian marginal coordinate distributions, as illustrated empirically in Figure 6.

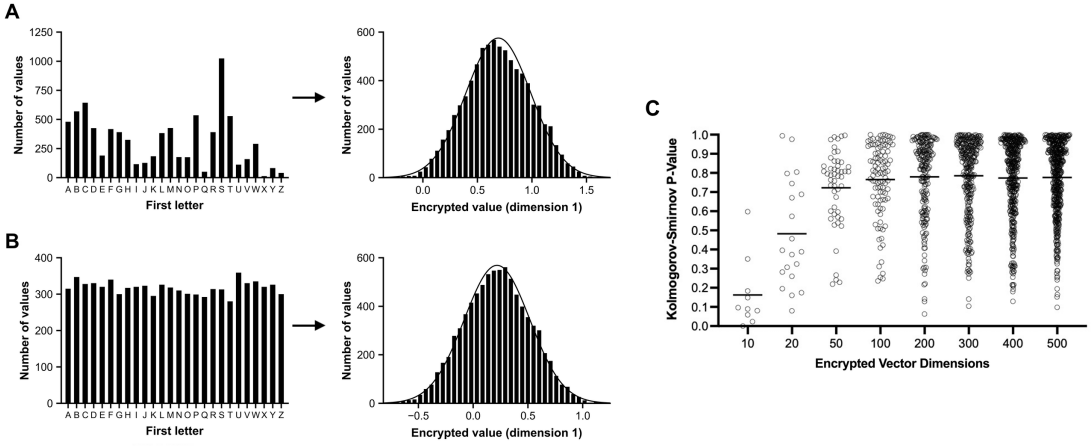

Figure 6: Transformation mixes coordinate structure across dimensions.
A. The distribution of initial letters of 5-letter plaintext English words, Tensorlock-transformed to an approximately Gaussian ciphertext distribution.
B. A random uniform distribution of plaintext letters is similarly Tensorlock-transformed into an approximately Gaussian ciphertext distribution.
C. The distribution of encrypted vectors is increasingly Gaussian at higher vector dimensions, with increasing Gaussian goodness of fit by the Kolmogorov-Smirnov P-value (higher P-values represent better fit).

Increasing the number of dimensions by padding further contributes to this mixing effect. The original plaintext signal occupies only d dimensions, while the remaining m dimensions contain random padding. After the random orthogonal transformation, the signal-bearing and padding coordinates are jointly mixed across all $D = d + m$ encrypted dimensions. Thus, increasing m distributes the information from a smaller number of plaintext dimensions into a larger encrypted space, while also increasing the contribution of random padding to each encrypted coordinate.

This effect can be illustrated by examining the distributions of values within individual vector dimensions. In plaintext representations, individual coordinates may have strongly structured distributions reflecting properties of the underlying data, as illustrated with vectorized words in Figure 6A. After aggregate transformation, each encrypted coordinate is composed from contributions

distributed across many padded and shuffled coordinates. As a result, the marginal distributions of encrypted coordinates become substantially less reflective of the structure present in individual plaintext coordinates and become increasingly well approximated by Gaussian distributions at higher padded vector dimensions (Figure 6).

6.2 Information leakage

The information leakage in the Tensorlock encryption process can be defined by focusing on the specific combination of padding and orthogonal transformation.

An original plaintext vector is $x_i \in \mathbb{R}^d$ with corresponding label $y_i \in Y$. We first embed x_i into a higher-dimensional space. Let $D > d$ be the encrypted dimension. The padded vector is:

$$u_i = \mathrm{Pad}_{S,r_i}(x_i) \in \mathbb{R}^D \tag{1}$$

where $S \subseteq \{1, \ldots, D\}$ is the hidden set of coordinates containing the original signal, and r_i is a padding vector. (This includes values in added dimensions and/or interlaced padding.)

A fixed random shuffle is then applied (a permutation matrix $P \in \mathbb{R}^{D\times D}$), followed by a random orthogonal matrix $Q \in O(D)$, where $Q^\top Q = QQ^\top = I$. The encrypted vector is $z_i = u_i PQ$. Since the product of a permutation matrix and an orthogonal matrix is orthogonal, the aggregate orthogonal matrix is $A = PQ$, where $A^\top A = I$.

The padded vector encryption is thus:

$$z_i = u_i A \tag{2}$$

So the encrypted dataset is $Z = UA$, where

$$U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \in \mathbb{R}^{n\times D}, \qquad Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \in \mathbb{R}^{n\times D}. \tag{3}$$

The attacker receives (Z, Y), but not X, U, S, P, Q, A.

Because A is orthogonal,

$$ZZ^\top = UAA^\top U^\top = UU^\top \tag{4}$$

Therefore, the attacker can compute

$$G_Z = ZZ^\top \tag{5}$$

which equals the *Gram matrix* of the padded vectors:

$$G_U = UU^\top \tag{6}$$

This information leakage is intentional: it is what allows neural networks to train directly on the encrypted data—The Gram matrix serves as a summary of the structural relationships like lengths and angles between vectors in a given set.

The Tensorlock method preserves:

$$\langle z_i, z_j \rangle = \langle u_i, u_j \rangle \tag{7}$$

and

$$\|z_i - z_j\|_2 = \|u_i - u_j\|_2. \tag{8}$$

So, in the absence of auxiliary information, (Z) directly determines the Gram geometry but not its original coordinate orientation.

6.3 Indistinguishability of the orthogonal transformation

Because a Haar-random orthogonal matrix is uniformly distributed over all rotations and reflections of the vector space, multiplying the padded data by such a matrix randomizes the coordinate basis without changing the dataset's geometry.

The Gram matrix, which records pairwise inner products between samples, is preserved in the transformation of padded vectors to encrypted vectors. Therefore, any two padded datasets with the same Gram matrix have equivalent orthogonal geometry: they differ only by a rotation or reflection of the padded D-dimensional space. More formally, if two padded datasets U and V satisfy

$$UU^\top = VV^\top, \tag{9}$$

then there exists an orthogonal matrix $R \in O(D)$ such that

$$V = UR. \tag{10}$$

If A is sampled from the Haar distribution on $O(D)$, then Haar invariance gives

$$RA \overset{d}{=} A, \tag{11}$$

where $\overset{d}{=}$ denotes equality in distribution. Consequently,

$$VA = URA \overset{d}{=} UA. \tag{12}$$

Thus, conditioned on the preserved Gram matrix, the ciphertext distribution contains no further information identifying the orientation of the padded dataset.

Specifically, the attacker observes

$$Z = UA. \tag{13}$$

Because A is unknown and orthogonal, every candidate dataset of the form $U' = ZR$ for any orthogonal matrix $R \in O(D)$ is equally compatible with the observed encrypted data. Indeed, Equation (13) implies

$$U = ZA^\top. \tag{14}$$

But since the attacker does not know A, there are infinitely many possible decryptions:

$$U_R = ZR, \qquad R \in O(D). \tag{15}$$

All of these satisfy

$$U_R U_R^\top = ZRR^\top Z^\top = ZZ^\top. \tag{16}$$

The attacker can therefore determine the Gram matrix $ZZ^\top$, and hence the orthogonal-invariant geometry of the padded dataset, but not its original coordinate orientation. Equivalently, among padded datasets with the same Gram matrix, the ciphertext does not distinguish which coordinate

representation generated the observed encrypted data. The attacker therefore cannot identify the true coordinate system of U, much less the original unpadded coordinates x_i, without additional information that breaks this orthogonal symmetry.

6.4 Effect of padding

Padding further strengthens the above argument because the attacker is not merely trying to invert a transformation of the form $z_i = x_i Q$.

Instead, the attacker observes

$$z_i = \mathrm{Pad}_{S,r_i}(x_i)A, \tag{17}$$

where $S \subseteq \{1, \ldots, D\}$ denotes the hidden set of pre-orthogonal coordinates containing plaintext signal and r_i denotes the padding values associated with example i.

Recovering the original plaintext therefore requires identifying which coordinates in the pre-orthogonal representation correspond to plaintext and which correspond to padding. That is, the attacker must infer the hidden set

$$S \subseteq \{1, \ldots, D\}. \tag{18}$$

Without knowledge of S, an attacker cannot distinguish plaintext coordinates from padding coordinates and therefore cannot uniquely recover x_i.

Importantly, S is defined in the pre-orthogonal coordinate system. As shown in the preceding section, this coordinate system is itself not identifiable from the ciphertext: conditioned on the preserved Gram matrix, the ciphertext distribution contains no further information identifying the orientation of the padded dataset. The attacker therefore cannot search for S directly in the observed ciphertext coordinates.

The attacker consequently faces two coupled inverse problems: identifying a coordinate representation consistent with the observed orthogonal-invariant geometry and identifying which coordinates in that hidden representation correspond to plaintext rather than padding.

If the plaintext dimension d is known, the number of possible signal-coordinate subsets is

$$\binom{D}{d} = \frac{D!}{d!(D-d)!}. \tag{19}$$

This quantity describes the size of the latent signal-coordinate hypothesis space before incorporating auxiliary statistical information about the plaintext distribution or covariance structure. It does not imply that all candidate subsets are equally plausible or that an optimized attack strategy must enumerate every subset exhaustively.

The orthogonal transformation hides the coordinate basis, while padding hides which coordinates were meaningful before transformation. Together, these transformations create a non-identifiable inverse problem from the attacker's perspective:

$$x_i \longrightarrow u_i \longrightarrow z_i, \tag{20}$$

where observing z_i does not determine a unique latent plaintext interpretation without additional information that reduces the orthogonal or padding ambiguities.

The binomial count in Equation (19) assumes, for simplicity, that the attacker has *a priori* knowledge of both D and d. In the threat model considered here, however, the attacker observes the encrypted dimension D but does not know the original plaintext dimension d. The attacker knows only that $D > d$, under the assumption that some padding has been performed.

If D is known but d is unknown, the latent hypothesis space includes all possible nonempty proper signal-coordinate subsets. For *each* candidate plaintext dimension $\hat{d} = 1, \ldots, D-1$, there are

$$\binom{D}{\hat{d}} \tag{21}$$

possible choices of signal-coordinate set. Therefore, the total number of possible nonempty proper subsets is

$$\sum_{\hat{d}=1}^{D-1} \binom{D}{\hat{d}} = 2^D - 2. \tag{22}$$

As discussed below in Section 6.6, additional statistical information may reduce the effective hypothesis space by making some candidate representations more plausible than others. The covariance-based attacks considered below explicitly test how much such auxiliary information can reduce this ambiguity. Those experiments therefore address whether knowledge of plaintext structure is sufficient to identify the hidden coordinate representation and recover meaningful example-specific plaintext information.

6.5 Access to an inference oracle

In our threat model (Figure 1), an attacker is able to submit plaintext vectors to an inference oracle, which returns only inference label results (y'_j.).

So the attacker receives no direct algebraic constraints of the form

$$z'_j = \mathrm{Pad}(x'_j)A. \tag{23}$$

The oracle leaks information about the learned decision function

$$f : \mathbb{R}^d \to Y, \tag{24}$$

but not about the encryption key. Thus, for key recovery, the oracle gives only scalar or categorical outputs:

$$x'_j \mapsto y'_j, \tag{25}$$

rather than vector equations involving A. These data are insufficient to solve for the hidden orthogonal matrix.

6.6 Attacker knowledge of plaintext covariance structure

6.6.1 Covariance structure

In our threat model, the attacker does not have access to the plaintext vectors that were used to generate encrypted vectors. However, an attacker could have general information about the subject matter of a given project from which they have obtained encrypted vectors. For example, they might know that the encrypted vectors are derived from a certain type of financial data, English-language text, or a particular type of medical imaging study.

While general knowledge of the plaintext subject matter cannot be modeled mathematically or experimentally, more exact forms of reference knowledge can establish bounds on attacker capabilities.

If we assume that the attacker knows the *exact dataset* from which the data were obtained (without known plaintext to ciphertext row correspondence), this would mean that the attacker possesses the exact plaintext covariance structure. Knowing the covariance of the plaintext X does not mean that the attacker knows the covariance of the padded data U. Rather, it gives the attacker a known statistical component embedded inside the covariance of U, and their attack strategy can attempt to locate and align that component after orthogonal transformation.

After subtracting the mean value in each vector dimension from every example, the *centered plaintext* dataset is

$$X_c = X - \mathbf{1}\mu_X^\top, \tag{26}$$

where $X \in \mathbb{R}^{n\times d}$, $\mu_X \in \mathbb{R}^d$ is the vector of column means, and $\mathbf{1} \in \mathbb{R}^n$ is the all-ones vector. Then $\mathbf{1}\mu_X^\top$ is an $n \times d$ matrix that repeats the mean vector in every row.

The plaintext covariance matrix, which summarizes how the plaintext dimensions vary together across examples, is then

$$\Sigma_X = \frac{1}{n-1} X_c^\top X_c. \tag{27}$$

Its eigenvalue decomposition is

$$\Sigma_X = V_X \Lambda_X V_X^\top, \tag{28}$$

where the columns of V_X are orthonormal covariance eigenvectors representing the principal directions of variation in the plaintext data, and the diagonal entries of Λ_X are the corresponding nonnegative eigenvalues, representing the variance along those directions.

Before orthogonal transformation, the plaintext vectors are embedded into the larger D-dimensional padded vectors U. For illustration, if the plaintext and padding coordinates are written as contiguous blocks,

$$U = [X\ R], \tag{29}$$

then after centering, the covariance matrix of the padded data can be written as

$$\Sigma_U = \begin{bmatrix} \Sigma_X & \Sigma_{XR} \\ \Sigma_{RX} & \Sigma_R \end{bmatrix}. \tag{30}$$

If the padding values are generated independently of the plaintext values, both cross-covariance terms are expected to be small:

$$\Sigma_{XR} \approx 0, \qquad \Sigma_{RX} \approx 0. \tag{31}$$

The padded covariance matrix can therefore be approximated as

$$\Sigma_U \approx \begin{bmatrix} \Sigma_X & 0 \\ 0 & \Sigma_R \end{bmatrix}. \tag{32}$$

Thus, although the attacker does not know the complete padded covariance matrix Σ_U, the known plaintext covariance matrix Σ_X is present as a component of it. Interlacing the plaintext and padding coordinates changes the locations of these components but does not remove this underlying covariance relationship. Interlacing can be represented as a coordinate permutation before the final orthogonal transformation.

The padded vectors are then transformed by the unknown orthogonal matrix A:

$$Z = UA, A^\top A = I. \tag{33}$$

Because the same orthogonal transformation is applied to every example, after centering the encrypted data the covariance matrix of Z is

$$\Sigma_Z = A^\top \Sigma_U A. \tag{34}$$

Matrices related in this way are *orthogonally similar*. Orthogonal similarity changes the coordinate directions in which the covariance structure is expressed, but preserves the covariance eigenvalues exactly. Consequently,

$$\text{eig}(\Sigma_Z) = \text{eig}(\Sigma_U). \tag{35}$$

The attacker can therefore calculate the eigenspectrum of the encrypted data and search it for eigenvalue structure corresponding to the known plaintext covariance matrix Σ_X. If plaintext-associated eigenvalues can be identified within the eigenspectrum of Σ_U, their corresponding encrypted-space eigenvectors provide information about how those plaintext covariance directions were rotated by the unknown orthogonal transformation A.

Let a padded-data covariance eigenvector v_j associated with the plaintext component satisfy

$$\Sigma_U v_j = \lambda_j v_j. \tag{36}$$

Combining Equations 34 and 36, and using $AA^\top = I$, gives

$$\Sigma_Z \left(A^\top v_j\right) = \lambda_j \left(A^\top v_j\right). \tag{37}$$

Thus, $A^\top v_j$ is an eigenvector of the encrypted covariance matrix Σ_Z with the same eigenvalue λ_j.

In the ideal case in which the relevant eigenvalues are distinct and well separated, the plaintext covariance eigenvector v_j and its corresponding encrypted covariance eigenvector w_j therefore satisfy

$$w_j = \pm A^\top v_j. \tag{38}$$

The factor ± 1 occurs because an eigenvector is mathematically determined only up to sign. If v_j is an eigenvector with eigenvalue λ_j,

$$\Sigma_U v_j = \lambda_j v_j, \tag{39}$$

then $-v_j$ is an equally valid eigenvector because

$$\Sigma_U(-v_j) = \lambda_j(-v_j). \tag{40}$$

These independent sign ambiguities can be represented collectively by a diagonal sign matrix

$$S = \text{diag}(s_1, \ldots, s_d), \qquad s_j \in \{-1, +1\}. \tag{41}$$

An attack leveraging the above structure attempts to estimate the inverse alignment of the plaintext component by:

- attempting to identify encrypted covariance eigenvalues corresponding to the known plaintext covariance spectrum

- pairing their associated encrypted and plaintext eigenvectors
- and resolving the remaining eigenvector sign choices.

This procedure is predicted to be exact or near-exact only under highly favorable conditions. Padding eigenvalues may overlap plaintext eigenvalues, finite-sample covariance estimates may differ, and repeated or nearly repeated eigenvalues do not uniquely determine individual eigenvectors.

The properties described above provide the foundation for experimentally testing whether the statistical information preserved by the orthogonal transformation is sufficient for meaningful recovery of individual plaintext examples.

6.6.2 Covariance structure – experimental attack modeling

To determine how useful knowledge of plaintext covariance structure is in practice, we modeled attacks that attempted to identify encrypted covariance eigenvalues corresponding to the known plaintext covariance spectrum, pair encrypted and plaintext eigenvectors, and resolve eigenvector sign choices. We start with exact (row-permuted) knowledge of plaintext covariance structure and then slightly decrease the attacker's knowledge to establish bounds on real-world scenarios in which the attacker has some information about plaintext subject but does not possess exact plaintext covariance structure information.

The experimental setup provides the attacker with a reference dataset with dimension d and a Tensorlock-encrypted dataset (interlaced padding to increase dimensions by 50%, $\pm 5\%$ perturbation with clipping, index shuffle, orthogonal transformation) with dimension D. After the eigenvalue/eigenvector/signs recovery attempt is complete, the attacker-recovered dataset is compared pairwise to the plaintext ground-truth dataset that was used to generate the encrypted dataset. The base condition provides the attacker with a reference dataset that is row-permuted (example-shuffled) relative to the ground-truth dataset, so they do not possess exact before-after transformation pairs but they do possess exact knowledge of plaintext covariance structure. Further experiments test the effect of slightly reducing the attacker's plaintext covariance knowledge by slightly changing the relationship between the reference dataset and the ground-truth dataset.

The results of this attack method are reported as mean absolute error (MAE): for coordinate float values in the range (0,1), recovered vector to ground-truth vector pairs are compared per-coordinate and the mean taken per example and then averaged across the entire dataset. The *paired MAE* reflects the average per-coordinate error between recovered and ground-truth example pairs and the *same-class MAE* reflects the per-coordinate error between recovered examples and the average of randomly selected ground-truth examples in the same label category. For reference, pairs of randomly generated vectors with float (0,1) values have a mean paired MAE of approximately $0.\overline{33}$.

6.6.3 Covariance structure – 'known d' attack

In the presence of exact (row-permuted) reference dataset, recovery success varied based on the complexity of the dataset, with better recovery of MNIST than for Fashion-MNIST, as shown in Table 2. When we use MNIST as a reference for an EMNIST-Digits ground-truth dataset (a 60K/10K split of the overall EMNIST-Digits, transposed to correspond to MNIST's configuration) that represents the same subject matter but with a slightly different covariance structure, the recovery is very poor (paired MAE substantially increases from 0.082 with MNIST ground truth to 0.241 with EMNIST-Digits ground truth). The worsening in paired MAE with this subtle difference between reference and ground-truth makes the paired MAE similar to same-class MAE, indicating

Table 2: Recovery with original dataset reference

Reference	**Ground-Truth**	**Mean Paired MAE**	**Mean Same-class MAE**
Fashion-MNIST	Fashion-MNIST	0.109	0.223
MNIST	MNIST	0.082	0.179
MNIST	EMNIST-Digits	0.241	0.236

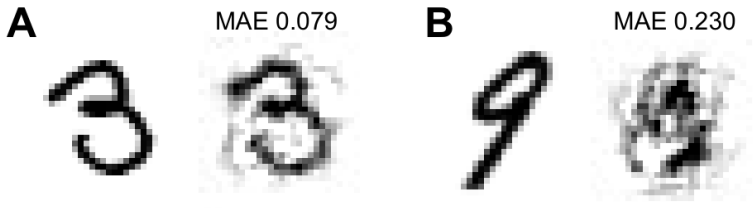

Figure 7: Recovery examples with original dataset references.
A. Ground truth and recovery with MNIST reference and row-permuted MNIST ground truth.
B. Ground truth and recovery with MNIST reference and EMNIST-Digits ground truth.

no recovery of example-specific detail beyond category detail. Examples of recovery attempts are shown in Figure 7.

To more finely adjust the difference between the reference and ground-truth datasets, we next altered the ground-truth subtly from the original dataset used as reference by applying image filtering to the ground-truth. Figure 8 shows the result of subtle image sharpening (and even more subtle gradations between the original and sharpened images), with increase in mean paired MAE from 0.110 under exact conditions through to 0.208 when ground-truth has full sharpening applied as illustrated (application of PIL's ImageFilter.SHARPEN). Intermediate levels of sharpening (created by averaging) demonstrated intermediate worsening of paired MAE (Figure 9).

The mathematical underpinning of the covariance structure attack depends on the cross-covariance terms in the covariance matrix approaching zero, which in turn is only the case if padding values and plaintext values have substantially independent covariance structure (see above). When the attacker is given a reference dataset, they know the number of plaintext dimensions, the number of ciphertext dimensions, and thus the number of padding dimensions. If uniform-sampled random padding and perturbation are used in this context, the recovery process is typically able to correctly identify the correct padding coordinates. For example, in the 'known d' + exact / near-exact reference recovery experiments shown in Table 2 and Figures 7 & 8, the recovery process correctly identified all padding coordinates.

This correct identification of padding coordinates when padding values are drawn from a uniform random distribution is predicted from the covariance structure created by this approach. If each padding coordinate is independently sampled from a uniform distribution on $[0, 1]$, then after centering it has mean 0 and variance $1/12$. Because the padding coordinates are mutually independent, their off-diagonal covariances are expected to be close to zero. A covariance matrix proportional to the identity matrix has the same eigenvalue in every coordinate direction. Thus, the m padding directions are expected to contribute approximately m covariance eigenvalues clustered near $1/12$. Because the final orthogonal transformation preserves covariance eigenvalues, these similar padding eigenvalues remain present in the eigenspectrum of the encrypted data and can appear as a plateau near $1/12$. In a finite dataset the plateau will not be perfectly flat, and

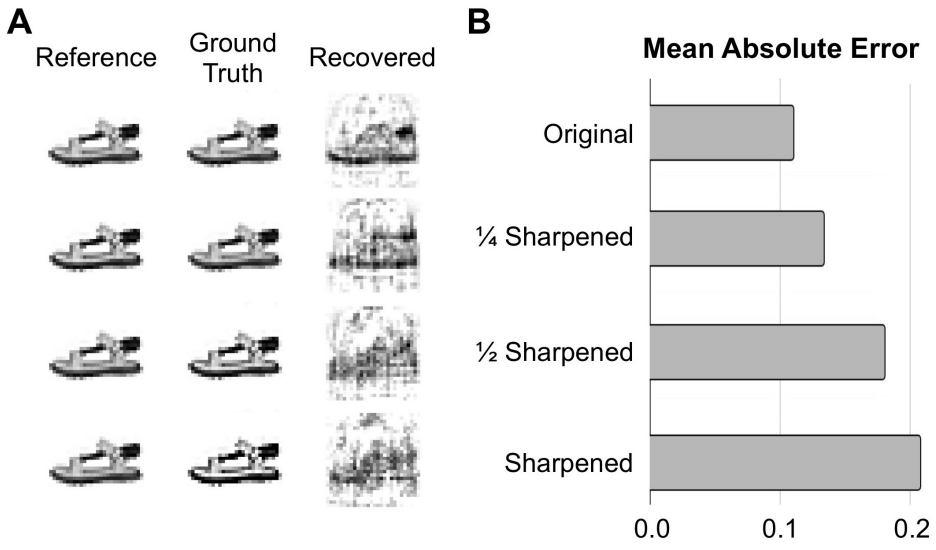

Figure 8: Impact of subtle ground-truth filtering on recovery.
A. Reference, ground-truth, and recovery attempt for examples under conditions of exact match between reference and ground truth (top row) and varying degrees of image sharpening applied to ground truth (lower rows).
B. Mean paired MAE under the conditions displayed in (A): original ground truth and increasing degrees of subtle sharpening as shown.

plaintext covariance eigenvalues may overlap the same region. The plateau therefore provides only an approximate indication of the number of padding dimensions rather than an exact identification of the padding subspace.

We therefore tested how padding coordinate identification and example recovery are impacted when padding and perturbation approaches are used that intentionally obscure the covariance of padding and plaintext coordinates.

6.6.4 Obfuscation of padding and plaintext covariance

To measure the impact of methods to obscure the distinction between padding and plaintext covariance structure, we employed *data-informed structured padding* and *structured nonlinear perturbation*. Examples of data-informed structured padding tested included reference-derived structured padding (a principal component analysis (PCA)-score bootstrap approach), signal-dependent structured padding (using cross-covariance), and mixed structured padding (hybrid padding in which the PCA bootstrap and cross-covariance are chosen at random for each padding value). Examples of structured non-linear perturbation tested included a fixed random nonlinear cross-coordinate perturbation, a spectrally optimized nonlinear perturbation, and spectrally optimized nonlinear perturbation with a utility-preserving penalty. Each of these techniques applies a fixed, input-dependent nonlinear function that couples coordinates while limiting the maximum value change to a specified range.

These padding and perturbation techniques can be applied in many different combinations, together with varying parameters like perturbation bounds and clipping vs. reflection. For the sake of simplicity, we will present here the results of one approach as an example:

- Source-derived PCA-score bootstrap padding: Padding vectors are generated by projecting

source examples onto selected principal components and independently resampling the resulting PCA-score vectors with replacement, optionally followed by a fixed rotation of the padding subspace.

- Spectrally optimized nonlinear perturbation with a utility-preserving penalty: A nonlinear function is optimized on the training data to rotate the covariance eigenbasis and reduce distinctive spectral gaps while maintaining a specified perturbation bound. This approach is supplemented with a penalty that limits changes to class centroids in a low-dimensional projection.
- 5% perturbation range with clipping to (0,1) float range.

When the above padding and perturbation strategies are used, there is a substantial drop in accuracy of example recovery despite exact reference - ground truth match and known d conditions, and the number of correctly identified padding coordinates drops from the baseline uniform padding and uniform perturbation condition (Table 3, Figure 9).

Table 3: Recovery performance with uniform vs. PCA-bootstrap padding

Ref & Gnd Truth	**Padding**	**Perturbation**	**Paired MAE**	**Same-class MAE**	**Correct Padding Coord.**
MNIST	Uniform	Uniform	0.082	0.179	392/392
MNIST	PCA Bootstrap	Struct. Nonlinear	0.140	0.190	195/392
Fashion-MNIST	Uniform	Uniform	0.109	0.223	392/392
Fashion-MNIST	PCA Bootstrap	Struct. Nonlinear	0.171	0.240	355/392

Summarizing the above findings, some degree of example recovery success is possible when an attacker possesses an exact (row-permuted) reference dataset and knows the value of d, but very small differences between reference and ground truth or use of advanced padding and perturbation techniques result in poor recovery results, even with knowledge of d.

6.6.5 'Unknown d' attack modeling

The scenario in which d is unknown presents a much greater challenge to an attacker. A methodical search strategy must be chosen across all $\hat{d}$ candidates where $\hat{d} < D$, and if uniform random padding is assumed, a screen for a covariance plateau should be used as an estimator for a potential range of $\hat{d}$ candidates.

We have deployed several strategies for attempted recovery in this context, including a direct spectral heuristic and a covariance spectral plateau-aware, joint generative estimator. To provide a reference that discloses domain covariance information but not d, we used downsampled reference tensors (and also tested upsampled ground truth tensors).

Even when uniform padding is used and the $\hat{d}$ candidate search is tuned to this (thus providing the attacker with knowledge they would not have), the attempt to estimate the number of padding dimensions almost always fails to identify the correct $\hat{d} = d$. When $\hat{d}$ identified is close to or equal to d, the chosen plaintext coordinates were $< 70\%$ correct. Even with uniform padding, our attempted methods failed to reliably identify coordinates to remove from the estimated plaintext that actually corresponded to padding coordinates—the other $> 30\%$ were plaintext coordinates that were confused with padding.

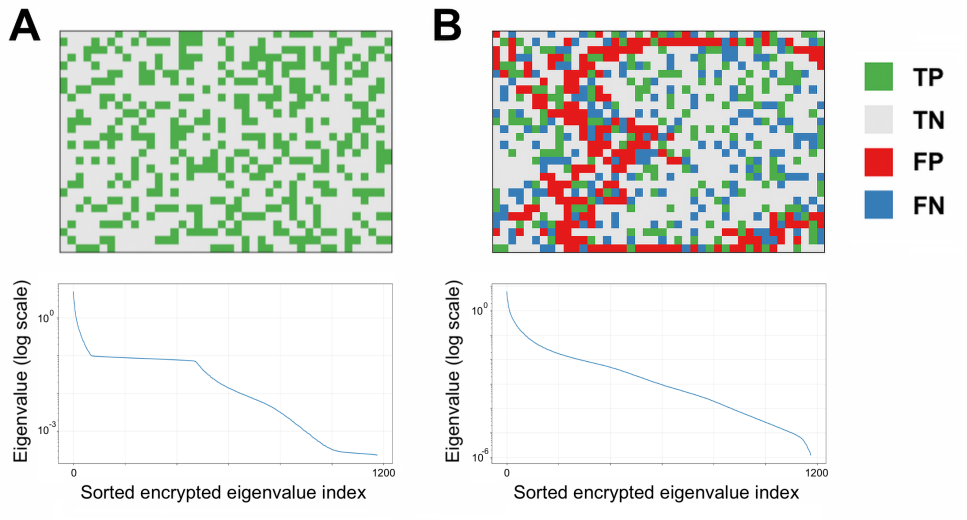

Figure 9: Padding coordinate identification with known d and exact (row-permuted) reference.
A. Correct identification of all padding coordinates with exact MNIST reference and ground-truth pair (row permuted) and random uniform padding and perturbation. Green pixels in the padding coordinate map (upper panel) are true positive (TP) padding coordinates and gray pixels are true negative (TN) plaintext coordinates. The lower panel shows a plateau in the sorted eigenvalue index of the encrypted vectors caused by use of uniform padding.
B. Substantial failure to correctly identify padding coordinates with the same exact MNIST reference to ground truth pair when advanced padding and perturbation are used (data shown from the same PCA-score bootstrap padding and structured nonlinear perturbation experiment shown in Table 3). Red pixels are false positives (FP) and blue pixels are false negatives (FN). The lower panel shows that advanced padding and perturbation eliminate the plateau in the sorted eigenvalue index.

Advanced padding and perturbation techniques, as described above, start off with poor recovery and poor padding coordinate identification even in the presence of known d and an exact row-permuted reference (Table 3, Figure 9). As expected, we were unable to obtain any meaningful example recovery, padding coordinate identification, or even correct determination of $\hat{d} = d$ when we performed 'unknown d' recovery experiments with advanced padding and perturbation techniques such as PCA-score bootstrap padding and structured nonlinear perturbation.

6.7 Direct exhaustive inversion without plaintext knowledge

The preceding analysis shows that Tensorlock creates a large inverse problem involving an unknown orthogonal transformation, hidden plaintext-coordinate structure, and perturbation. The covariance-based experiments in Section 6.6 demonstrate that exact or near-exact plaintext covariance information can sometimes reduce this search space, but this advantage quickly diminishes as covariance knowledge is even slightly reduced. In the case where an attacker lacks *a priori* plaintext structural knowledge, they face the task of direct exhaustive inversion.

For an example with $d = 784$ plaintext dimensions and $D = 1176$ encrypted dimensions, direct exhaustive inversion requires searching an enormous space. A 1176×1176 orthogonal matrix has $D(D-1)/2 = 690{,}900$ continuous degrees of freedom. In computer memory, this continuous space is represented at finite floating-point precision, yielding an astronomically large number of numerically distinguishable orthogonal transformations. Discretizing each degree of freedom at approximately double-precision resolution corresponds to a nominal search space on the order of $2^{36 \text{ million}}$ possibilities. While this is not a precise cryptographic keyspace count, it illustrates the scale of the transformation space and why straightforward brute-force reversal is computationally infeasible at realistic encrypted vector dimensions.

Since the plaintext dimension is also unknown, the attacker must also consider approximately 10^{354} possible nonempty signal-coordinate subsets for our example with $D = 1176$ encrypted dimensions. This adds a further combinatorial search problem because the attacker must determine not only the hidden orthogonal transformation, but also which latent coordinates correspond to the original plaintext dimensions.

Finally, an attacker would additionally need to account for or estimate the effects of the unknown perturbation before recovering the underlying plaintext representation.

Further uncertainty arises because the attacker does not know which transformation steps were applied, the order in which they were applied, or whether particular steps were applied iteratively. The many possible forms and parameterizations of padding and perturbation further increase the space of candidate encryption processes that an attacker must consider.

6.8 Challenge dataset

An example dataset encrypted with the above methods (beginning with a synthetic unknown dataset) is available at tensorlock.ai/challenge_dataset.

To fit the threat model, the plaintext dataset is a synthetic dataset with content, shape, dimension d, and datatype unknown to the potential attacker. The components of the Tensorlock method applied, their order, and the specific parameters used are unknown to the potential attacker.

The encrypted dataset is stored as a PyTorch .pt file and consists of a 58,862 train / 9,786 test split with (49, 48) examples with float32 values and labels in 10 categories (0-9).

7 Discussion

7.1 Summary

The network training properties and security parameters of the Tensorlock encryption method are described here in detail. Training on datasets transformed by sequential padding, perturbation, index shuffling, and orthogonal transformation results in validation accuracies that are only slightly lower than those seen with unmodified plaintext datasets, with only a very small increase in training time. In the absence of exact or near-exact knowledge of plaintext covariance structure, an attacker attempting to invert the transformation requires computationally infeasible search across the resulting transformation and coordinate hypothesis spaces.

7.2 Security aspects of the method

The security of our encryption methods in the context of the threat model (Figure 1) is assessed above in Section 6 - Security Analysis.

7.2.1 Security parameters

Practical security schemes have variable *security parameters* that can be used to change the strength of the encryption scheme, such as the key size in AES [National Institute of Standards and Technology, 2001]. Small AES key sizes might allow the scheme to be easily broken with modern computing resources, while larger key sizes such as the current AES standard of 256 bit keys provide security by making the problem computationally intractable. Varying security parameters in such schemes typically involve tradeoffs; for example, larger key sizes yield greater security but lead to slower encryption computation.

In the present method, there are several security parameters that can be adjusted to vary the level of security for a given project within the defined threat model:

- Extent of padding applied per example (number of dimensions m appended to plaintext vector dimensions d to yield the padded $(d + m)$ dimension padded examples.
- Location of padding relative to plaintext vector dimensions (*e.g.,* interlaced, extrinsic, contiguous, non-contiguous).
- The type of padding values applied (*e.g.,* uniform, Gaussian, sampled, or data-informed structured padding).
- Choice of perturbation function (*e.g.,* change in values by random amounts or by another mathematical function or functions).
- Perturbation range (upper / lower bounds of the random perturbation or other perturbation functions).
- Handling of edge values when perturbation is applied (*e.g.,* clipping, reflection).
- Iterative application of multiple transformations (the ability to perform multiple rounds of padding, perturbation, index shuffling, and orthogonal transformation with the option to vary any of the above security parameters in each round, change the order of transformation, or exclude particular transformations in a given round).

For larger input vectors, larger random orthogonal matrices are required. For extremely large random orthogonal matrices, sampling the Haar distribution becomes computationally intensive. In such a circumstance, computation time can be greatly reduced by splitting input vectors into smaller

sub-vectors, generating random orthogonal matrices corresponding to the number of dimensions of the sub-vectors, dotting the sub-vectors by each orthogonal matrix, and concatenating the resulting output vectors. This process is the same as matrix multiplication by a block diagonal matrix and is thus an aggregate orthogonal transformation. If this approach is used, any of the orthogonal matrices applied could be a square random binary permutation matrix corresponding to a fixed index shuffle. The result of this would be a hybrid shuffle - random orthogonal matrix transformation, representing yet another possible security parameter that could be manipulated.

Independent security testing may be performed on a publicly accessible encrypted challenge dataset (see Section 6.8 Challenge dataset, above).

7.3 Alternative approaches

Various alternative approaches to the problem of securing data for model training have been proposed, with recent work focused on the areas of federated learning, the use of data treated with fully homomorphic encryption, and several proposals for instance encoding, including the InstaHide method [Huang et al., 2021], the NeuraCrypt method [Yala et al., 2021], the DarKnight scheme [Hashemi et al., 2020], and application of single-step orthogonal transformation based on an image key [Chen, 2019].

7.3.1 Federated learning

Federated learning is a well-established approach for performing distributed model updates using data on local devices, wherein training data do not have to be transmitted from the local device to a central server. In the federated learning framework, a model and weights are transmitted to the local device, weights are updated by additional local training, and then the updated models and weights returned to the central server [McMahan et al., 2017, Kairouz et al., 2021, Li et al., 2020, Rodríguez-Barroso et al., 2020]. While this approach has the advantage of not needing to transmit data from a local device to a central server, and has found wide application in updating models using rich data on more computing-constrained systems such as mobile devices, it has certain disadvantages when it comes to the problem of model training of private data. Software provided by the central server must be run to train or update models on local systems that maintain the private data, and this software has direct access to the private data. This means that the party maintaining the private data ('Party A') must trust the party providing the software ('Party B'). Party A must trust Party B to run software behind Party A's firewall and additionally trust this software to have direct access to Party A's private data. Party A additionally must trust Party B not to transmit any aspect of Party A's private data back to Party B's servers along with the updated models and weights that have to be transmitted from behind Party A's firewall to Party B's server.

Beyond security issues, federated learning poses a commercial risk: aggregating updates across diverse data sources dilutes and obscures the specific value of any single organization's private data contribution.

7.3.2 Fully homomorphic encryption

Fully homomorphic encryption (FHE) represents a family of approaches that allow mathematical manipulation of encrypted data such that when a manipulated ciphertext is decrypted, the result is identical to an analogous mathematical manipulation performed on the original data prior to encryption [Lauter, 2021, Munjal and Bhatia, 2023, Alloghani et al., 2019]. There have been initial explorations of using FHE in the context of secure machine learning [Bost et al., 2014, Graepel et al., 2013, Lauter, 2021, Nandakumar et al., 2019, Papernot et al., 2016], but FHE requires

profound alterations to machine learning algorithms that yield compute times that are many orders of magnitude higher than required for work on unencrypted data using conventional methods [Nandakumar et al., 2019]. For example, training on MNIST in an initial exploration of using FHE with a neural network required downsampling of the MNIST dataset to 8x8 pixel examples and use of a cyclotomic ring encryption considered to not provide sufficient security. Despite these dramatic simplifications, compute times for the neural network ranged from 40 minutes to 1.5 days [Nandakumar et al., 2019]. When a more secure FHE algorithm was tested, the time to process a single neuron in the first layer was 15 minutes [Nandakumar et al., 2019]. In another experiment to perform unsupervised learning with K-means-clustering on an FHE-encrypted dataset, single-thread runtimes were estimated to be on the timescale of months [Jäschke and Armknecht, 2019].

7.3.3 The InstaHide method

One form of instance encoding intended to provide secure machine learning is the InstaHide method [Huang et al., 2021]. InstaHide first applies the approach of 'mixup' [Zhang et al., 2018], performing a linear combination of several images from the training set and a large public database to generate multiple examples derived from combinations of the original examples. Next, the InstaHide method extends image pixel values to a negative-to-positive range, then applies a random pattern of sign inversions to each image [Huang et al., 2021]. Subsequent work by others [Carlini et al., 2021a, Carlini, 2020] has shown this method to be vulnerable to an attack that takes advantage of the specific steps known to occur in the InstaHide method, in particular, taking the absolute value of all pixel values (to remove the sign-flipping in InstaHide), cluster encoding the dataset by training a neural network "to detect when two encodings were generated from the same image," and solving an under-determined system of equations by gradient descent to recover an approximation of the original images [Carlini et al., 2021a].

7.3.4 The NeuraCrypt method

Another proposed form of instance encoding for secure machine learning is the NeuraCrypt method [Yala et al., 2021]. NeuraCrypt performs random encoding of an original dataset by sampling a random neural network with convolutional layers, batch normalization, and ReLU, using this randomly chosen network to encode the data, together with random positional embedding and permutation of image patches, to "encode positional information into the feature space while hiding spatial structure" [Yala et al., 2021]. The architecture of the networks used in NeuraCrypt is "closely inspired by the design of patch-embedding modules" of Vision Transformer networks [Dosovitskiy et al., 2021].

In a subsequent paper showing that it is possible to fully break the privacy of the NeuraCrypt method, the described attack leverages the particular structure of NeuraCrypt, namely the use of patch-embedding and information leakage from the position encoding used [Carlini et al., 2021b]. Their practical attacks on InstaHide and NeuraCrypt demonstrate that particular encoding architectures can preserve exploitable structure sufficient for reconstruction or recovery of the encoding transformation.

Carlini et al. have also argued more broadly that useful instance-encoding schemes face fundamental limits in achieving strong indistinguishability-based privacy [Carlini et al., 2021a]. Their results show that, under the privacy definitions they consider, retention of sufficient information for nontrivial learning necessarily permits some forms of distinguishing attack.

A key assumption underlying this argument concerns the adversary's access to the encoding function itself. In the formal attack models of Carlini et al., the adversary is granted oracle access

to the encoder, together with the associated learner and distribution samplers [Carlini et al., 2021a].

Our threat model does not grant this access. Party C possesses the encrypted dataset (Z_d) and the plaintext labels (Y_d), and may submit self-generated plaintext inference examples (x') to Party A's inference oracle, but the oracle returns only the predicted label (y'). The encrypted example (z') is never disclosed and is deleted immediately after inference is performed. Party C therefore never obtains a matched plaintext-ciphertext pair and cannot use such pairs to estimate the encoding function.

The security of our method depends on the inference-oracle access restriction described in Section 2: an encrypted representation is never released for attacker-supplied plaintext. The Tensorlock threat model therefore differs from the access assumptions used by Carlini et al., in which the attacker is granted oracle access to the encoding function.

7.3.5 The DarKnight and AsymML methods

Another proposed approach to training sensitive data using untrusted cloud hardware is the DarKnight approach, which passes traditionally encrypted examples to the cloud, where the examples are decrypted within a trusted execution environment (TEE) and basic ML operations are begun within this TEE [Hashemi et al., 2020]. In order to perform intensive linear and nonlinear operations on untrusted cloud GPUs, the DarKnight scheme blinds the data passed outside the TEE to the untrusted hardware as a *virtual batch*, where a number of input examples are linearly combined to form a number of encoded examples [Hashemi et al., 2020], similar to the linear combination *mixup* approach used as part of the InstaHide method discussed above [Huang et al., 2021].

The DarKnight authors exclude TEE side-channel attacks from their scope and do not provide model confidentiality on the untrusted GPU [Hashemi et al., 2021]. The reorganization of training into separate TEE and GPU environments also comes with a significant compute cost. Another model training over TEE and GPU technique, AsymML, improves on the compute times but still has a 15-fold execution time cost compared to plaintext GPU runs [Niu et al., 2022].

7.3.6 Single-step orthogonal transformation

Orthogonal transformation has been proposed as a single-step transformation for machine learning applications, specifically by performing QR factorization on a digital image file of appropriate size and using the resulting orthogonal matrix to transform input vectors [Chen, 2019].

The main weakness of single-step orthogonal transformation is vulnerability to direct algebraic recovery when a sufficient number of plaintext–ciphertext pairs are available. If plaintext examples are arranged as rows of a square matrix X, ciphertext examples as rows of a corresponding matrix Z, and the transformation is $Z = XT$, then, when X is invertible, the unknown transformation can be recovered directly as $T = X^{-1}Z$.

The simple inversion attack described above does not apply directly to the Tensorlock transformation. Padding changes the dimensionality of the representation, while example-dependent padding and perturbation mean that the relationship between an original plaintext vector and its encrypted representation is not described by a single fixed linear transformation acting on the original plaintext coordinates alone. The attacker must additionally infer the hidden padded representation and account for perturbation before the orthogonal transformation can be reversed.

7.3.7 Differential privacy

A broad area of computer science and mathematics known as differential privacy concerns privacy-preserving data analysis (the attempt to achieve a balance between data utility for analysis in the aggregate, while maintaining privacy at the individual level). Differential privacy is a modern approach to privacy-preserving data analysis that quantifies the tradeoff between data utility and privacy when training on original, sensitive data [Dwork and Roth, 2014]. The goal privacy guarantee in differential privacy is that an attacker does not learn significantly more about an individual (or example) if that individual (or example) is present in a training dataset than when they are not present in the training dataset [Dwork et al., 2006]. Differential privacy approaches can be used with neural network training [Abadi et al., 2016], and privacy budgets can be individualized on a per-user basis [Boenisch et al., 2023]. The fundamental distinction from our approach is that differential privacy ensures a quantifiable degree of privacy by way of aggregation and analytic noise without being centered on encryption.

7.3.8 Other approaches

Other variations on the theme of homomorphic encryption with additional constraints have been tested with neural networks with improved performance results, such as the approach of leveled homomorphic encryption [Bos et al., 2013], together with tailoring of network structures to the specific encryption approaches used in order to make the required computations more tractable for neural network training [Gilad-Bachrach et al., 2016].

Another proposed approach adapts multi-party computation techniques [Goldreich, 2004], in which encrypted data is passed back and forth between parties, updating calculations layer by layer in stepwise fashion, with the sending party required at each step to decrypt the data, apply a nonlinear transformation, encrypt the result and return this to the other party [Barni et al., 2006, Orlandi et al., 2007]. While recent adaptations of this technique have improved the computational efficiency, the approach still requires bidirectional transmission of very large amounts of data, substantially longer training times when compared with plaintext training, and technical sophistication on both ends of the data exchange to implement the method [Keller and Sun, 2021].

7.4 Advantages of the present methods

Compared to alternative approaches, the Tensorlock method has certain advantages. Tensorlock enables:

- Use of largely unchanged machine learning systems on the encrypted data.
- Customization of the encryption process, with the option to vary many different security parameters.
- High-accuracy model training with very minimal impact on training time.
- Application of the same encryption approach to training, test, and inference data.

Tensorlock's requirement to encrypt inference data the same way that training and test data were encrypted protects the economic value of the encrypted training data used in a collaboration. A collaboration may be established such that the party that contributes encrypted data controls the secret of the encryption process, which must be securely accessed in order to perform inference. While massive datasets and compute power are both needed to create modern machine learning

solutions, to date, the economic value of the data required for training has been substantially undervalued [Beauchesne]. Despite increases in data volumes available for model training, demand for training data is already outstripping supply [Villalobos et al., 2024, Roded and Slattery]. By securing both privacy and the economic value of input data for model training and inference, the Tensorlock technique will help improve the future availability of needed data.

7.5 Limitations

Our study has limitations. The security analysis presented here is limited to the specific threat model defined in Figure 1 and to the classes of attacks examined in this work. Tensorlock is not claimed to provide information-theoretic privacy or general ciphertext indistinguishability. The transformation intentionally preserves geometric relationships required for machine-learning utility, including the Gram geometry of the padded dataset, and this preserved structure represents information available to an attacker.

The covariance-based recovery attacks examined here test whether this preserved information is sufficient to recover the original plaintext representation under progressively stronger assumptions about attacker knowledge, but the specific approaches tested are not exhaustive. Nevertheless, the analysis presented here goes beyond showing that a particular set of attacks performed poorly. Specifically, we show that while the encrypted dataset preserves the orthogonal-invariant geometry of the padded data, the original coordinate basis and the latent padding coordinates remain unknown. Within the stated threat model, the preserved geometric information is insufficient to uniquely recover the plaintext representation without additional information that breaks the orthogonal and padding ambiguities. The covariance-based attacks examined here demonstrate one way in which such ambiguity can be reduced: meaningful recovery becomes possible when an attacker possesses highly specific (exact or near-exact) knowledge of the plaintext covariance structure, while recovery rapidly degrades as that knowledge becomes less precise.

Our results do not constitute a proof that no more effective attack exists. As demonstrated by prior attacks on InstaHide [Carlini et al., 2021a] and NeuraCrypt [Carlini et al., 2021b], privacy-preserving transformations may contain exploitable structure that is not apparent from initial analysis, and future attacks may exploit properties not considered here. The conclusions regarding Tensorlock security should therefore be understood as conditional on the stated threat model and on the absence of additional auxiliary information sufficient to resolve the hidden coordinate representation.

7.6 Conclusion

Tensorlock, a technique for encrypting training, test, and inference data, enables direct network training on encrypted datasets. Training on datasets transformed by sequential padding, perturbation, fixed index shuffling, and random orthogonal transformation yields only slightly reduced validation accuracies compared to training on plaintext controls, with only very small differences in training times.

Modeled statistical attacks described here achieved meaningful recovery only with exact or near-exact knowledge of plaintext covariance structure. In the absence of such knowledge, inversion of $f^{-1}(Zd) \rightarrow Xd$ requires navigating astronomically large transformation and coordinate hypothesis spaces.

The methods described here can be deployed across a broad range of machine learning applications that require data security, and also serve to protect the business value and underlying intellectual property of training data.

References

Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, October 2016. doi: 10.1145/2976749.2978318.

Mohamed Alloghani, Mohammed M. Alani, Dhiya Al-Jumeily, Thar Baker, Jamila Mustafina, Abir Hussain, and Ahmed J. Aljaaf. A systematic review on the status and progress of homomorphic encryption technologies. *Journal of Information Security and Applications*, 48:102362, October 2019. ISSN 2214-2126. doi: 10.1016/j.jisa.2019.102362.

Eric Aubinais, Elisabeth Gassiat, and Pablo Piantanida. Fundamental Limits of Membership Inference Attacks on Machine Learning Models. *Journal of Machine Learning Research*, 26(263): 1–54, 2025. ISSN 1533-7928.

M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th Workshop on Multimedia and Security*, MM&Sec '06, pages 146–151, New York, NY, USA, September 2006. Association for Computing Machinery. ISBN 978-1-59593-493-2. doi: 10.1145/1161366.1161393.

Mathieu Beauchesne. The Crucial Importance of Data Valuation in Artificial Intelligence. https://www.gft.com/ca/en/blog/data-valuation-in-ai.

Franziska Boenisch, Christopher Mühl, Adam Dziedzic, Roy Rinberg, and Nicolas Papernot. Have it your way: Individualized Privacy Assignment for DP-SGD. https://arxiv.org/abs/2303.17046v2, March 2023.

Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Martijn Stam, editor, *Cryptography and Coding*, Lecture Notes in Computer Science, pages 45–64, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-45239-0. doi: 10.1007/978-3-642-45239-0_4.

Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data (Cryptology Archive, Paper 2014/331, https://ia.cr/2014/331). Technical Report 331, 2014.

Nicholas Carlini. InstaHide Disappointingly Wins Bell Labs Prize, 2nd Place. https://nicholas.carlini.com/writing/2020/, 2020.

Nicholas Carlini, Samuel Deng, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Shuang Song, Abhradeep Thakurta, and Florian Tramer. Is Private Learning Possible with Instance Encoding? (arXiv:2011.05315), April 2021a.

Nicholas Carlini, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Florian Tramer. NeuraCrypt is not private (arXiv:2108.07256), August 2021b.

Monchu Chen. Methods and processes of encrypted deep learning services (US Patent US16132078), March 2019.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: An extension of MNIST to handwritten letters (arXiv:1702.05373), March 2017.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arXiv:2010.11929), June 2021.

Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, August 2014. ISSN 1551-305X, 1551-3068. doi: 10.1561/0400000042.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 265–284, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-32732-5. doi: 10.1007/11681878_14.

Alexander Flint. Method and system for securely storing data for use with artificial neural networks (US Patent 12,731,025), September 2026.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210. PMLR, June 2016.

Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, Cambridge, UK ; New York, 1st edition, May 2004. ISBN 978-0-521-83084-3.

Thore Graepel, Kristin Lauter, and Michael Naehrig. ML Confidential: Machine Learning on Encrypted Data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 1–21, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-37682-5. doi: 10.1007/978-3-642-37682-5_1.

Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. DarKnight: A Data Privacy Scheme for Training and Inference of Deep Neural Networks (arXiv:2006.01300), October 2020.

Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. DarKnight: An Accelerated Framework for Privacy and Integrity Preserving Deep Learning Using Trusted Hardware. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, pages 212–224, New York, NY, USA, October 2021. Association for Computing Machinery. ISBN 978-1-4503-8557-2. doi: 10.1145/3466752.3480112.

Peter Henderson, Xuechen Li, Dan Jurafsky, Tatsunori Hashimoto, Mark A. Lemley, and Percy Liang. Foundation Models and Fair Use. *Journal of Machine Learning Research*, 24(400):1–79, 2023. ISSN 1533-7928.

Jack Hidary. AI training data is running low – but we have a solution. https://www.weforum.org/stories/artificial-intelligence/data-ai-training-synthetic/, December 2025.

Yangsibo Huang, Zhao Song, Kai Li, and Sanjeev Arora. InstaHide: Instance-hiding Schemes for Private Distributed Learning (arXiv:2010.02772), February 2021.

Angela Jäschke and Frederik Armknecht. Unsupervised Machine Learning on Encrypted Data. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*,

pages 453–478, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10970-7. doi: 10.1007/978-3-030-10970-7_21.

Nicola Jones. The AI revolution is running out of data. What can researchers do? *Nature*, 636 (8042):290–292, December 2024. ISSN 1476-4687. doi: 10.1038/d41586-024-03990-2.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning (arXiv:1912.04977), March 2021.

Marcel Keller and Ke Sun. Secure Quantized Training for Deep Learning (arXiv:2107.00501), July 2021.

Alex Krizhevsky. CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.

Kristin E. Lauter. Private AI: Machine Learning on Encrypted Data (https://eprint.iacr.org/2021/324). Technical Report 324, 2021.

Yann LeCun, Corinna Cortes, and Chris Burges. The MNIST database of handwritten digits. https://yann.lecun.com/exdb/mnist/.

Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, November 2020. ISSN 0360-8352. doi: 10.1016/j.cie.2020.106854.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data (arXiv:1602.05629), February 2017.

Kundan Munjal and Rekha Bhatia. A systematic review of homomorphic encryption and its contributions in healthcare industry. *Complex & Intelligent Systems*, 9:3759–3786, 2023. ISSN 2198-6053. doi: 10.1007/s40747-022-00756-z.

Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards Deep Neural Network Training on Encrypted Data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 40–48, June 2019. doi: 10.1109/CVPRW.2019.00011.

National Institute of Standards and Technology. Advanced Encryption Standard (AES). (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 197. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2001.

Yue Niu, Ramy E. Ali, and Salman Avestimehr. 3LegRace: Privacy-Preserving DNN Training over TEEs and GPUs. *Proceedings on Privacy Enhancing Technologies*, 4:183–203, 2022. ISSN 2299-0984.

OpenAI. How ChatGPT and our foundation models are developed. https://help.openai.com/en/articles/7842364-how-chatgpt-and-our-foundation-models-are-developed.

Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious Neural Network Computing via Homomorphic Encryption. *EURASIP Journal on Information Security*, 2007(1):1–11, December 2007. ISSN 1687-417X. doi: 10.1155/2007/37343.

Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the Science of Security and Privacy in Machine Learning (arXiv:1611.03814). November 2016.

Mark Phillips. International data-sharing norms: From the OECD to the General Data Protection Regulation (GDPR). *Human Genetics*, 137(8):575–582, August 2018. ISSN 1432-1203. doi: 10.1007/s00439-018-1919-7.

Tal Roded and Peter Slattery. What drives progress in AI? Trends in Data. https://futuretech.mit.edu/news/what-drives-progress-in-ai-trends-in-data.

Nuria Rodríguez-Barroso, Goran Stipcich, Daniel Jiménez-López, José Antonio Ruiz-Millán, Eugenio Martínez-Cámara, Gerardo González-Seco, M. Victoria Luzón, Miguel Angel Veganzones, and Francisco Herrera. Federated Learning and Differential Privacy: Software tools analysis, the Sherpa.ai FL framework and methodological guidelines for preserving data privacy. *Information Fusion*, 64:270–292, December 2020. ISSN 1566-2535. doi: 10.1016/j.inffus.2020.07.009.

Kim Theodos and Scott Sittig. Health Information Privacy Laws in the Digital Age: HIPAA Doesn't Apply. *Perspectives in Health Information Management*, 18(Winter):1l, December 2020. ISSN 1559-4122.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Position: Will we run out of data? Limits of LLM scaling based on human-generated data. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 49523–49544. PMLR, July 2024.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms (arXiv:1708.07747), September 2017.

Adam Yala, Homa Esfahanizadeh, Rafael G. L. D' Oliveira, Ken R. Duffy, Manya Ghobadi, Tommi S. Jaakkola, Vinod Vaikuntanathan, Regina Barzilay, and Muriel Medard. NeuraCrypt: Hiding Private Health Data via Random Neural Networks for Public Training (arXiv:2106.02484), June 2021.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization (arXiv:1710.09412), April 2018.