WHITE PAPER

Adapting Tensorlock to Convolutional Neural Networks

White paper · October 2026

Summary

Tensorlock encrypts training, test, and inference data so that a fully connected artificial neural network (ANN) can train on the ciphertext directly. The encryption removes the coordinate locality that convolutional neural networks (CNNs) require. This paper describes two ways to combine Tensorlock with CNN-style spatial processing.

Pre-encryption spatial feature extraction applies fixed convolution and pooling to the plaintext, then encrypts the resulting features. Training needs no interaction with the data owner. The convolution kernels cannot be learned during encrypted training.

Oracle-mediated spatial layer processing keeps the convolution kernels trainable. A key-holding service, the spatial oracle, performs each locality-dependent operation during training. This design requires repeated interaction, and it extends the threat model analyzed in the initial Tensorlock manuscript.

This paper is conceptual. It reports no experiments, and it makes no security claim beyond those in the initial Tensorlock manuscript.

1. Background

1.1 The Tensorlock transformation

Tensorlock applies four steps to each example: padding, perturbation, fixed index shuffling, and random orthogonal matrix transformation. Let $h \in \mathbb{R}^d$ be a flattened plaintext example. Padding and perturbation produce a vector of higher dimension $D > d$:

$$u = \mathrm{Pert}(\mathrm{Pad}_{S,r}(h)) \in \mathbb{R}^D.$$

Here, S is the hidden set of coordinates that hold the plaintext signal, and r is a padding vector drawn separately for each example. A fixed permutation matrix P and a Haar-random orthogonal matrix Q then give the ciphertext:

$$z = uA, \qquad A = PQ, \qquad A^\top A = I.$$

This paper writes the aggregate transformation as $z = T(h)$. The key comprises S, A, and the settings that generate padding and perturbation.

A key holder can reverse the orthogonal and padding steps:

$$\hat{h} = \left(zA^\top\right)_S = h + \eta.$$

The residual η is the perturbation noise. The key holder recovers h only up to this noise.

1.2 Why CNNs require adaptation

CNNs are a standard route to higher accuracy on harder image datasets. The two main CNN-specific operations of convolution and pooling combine values at neighboring coordinates. In Tensorlock, index shuffling moves each coordinate to a secret position, and orthogonal transformation mixes every coordinate across all D ciphertext dimensions. A network that receives z cannot identify which ciphertext values came from adjacent pixels.

1.3 Parties and threat model

This paper uses the threat model of the initial Tensorlock manuscript. Party A owns the plaintext dataset and the key. Party B receives the encrypted dataset and labels, and trains the network. Party C is an attacker who holds the encrypted dataset and labels, and who may be part of Party B.

That manuscript also defines an **inference oracle** controlled by Party A. It accepts a plaintext example, encrypts it privately, runs the trained network, and returns only the predicted label. It never discloses ciphertext for a submitted plaintext.

Section 3 introduces a second component, a **spatial oracle**. The two oracles differ in what they accept and return, and this paper keeps their names distinct.

2. Pre-encryption spatial feature extraction

2.1 Method

Party A performs every locality-dependent operation on plaintext, before encryption. Consider a conventional feature-extraction pipeline, where C_k denotes convolution and P_k denotes pooling:

$$x \to C_1 \to P_1 \to C_2 \to P_2.$$

Party A computes one or more intermediate representations:

$$h^{(0)} = x, \quad h^{(1)} = C_1\left(h^{(0)}\right), \quad h^{(2)} = P_1\left(h^{(1)}\right), \quad h^{(3)} = C_2\left(h^{(2)}\right), \quad h^{(4)} = P_2\left(h^{(3)}\right).$$

Party A flattens each selected representation and encrypts it with its own transformation:

$$z^{(k)} = T_k\left(h^{(k)}\right).$$

A feature map of shape $14 \times 14 \times 32$, for example, flattens to a vector of 6,272 dimensions before padding. Each T_k can use an independent key.

Convolution and pooling precede the transformation, so training requires no access to the key.

2.2 Training data and network architecture

Party B receives the label y and one or more encrypted representations for each example:

$$\left(z^{(0)}, z^{(1)}, \ldots, z^{(4)}, y\right).$$

The network that Party B trains is fully connected. With a single representation, it takes the form

$$z^{(k)} \to \mathrm{Dense} \to \mathrm{Dense} \to \mathrm{Softmax}.$$

With several representations, each feeds a separate fully connected branch B_k. The branch outputs are concatenated and passed to shared dense layers:

$$\left[B_1\left(z^{(1)}\right), B_2\left(z^{(2)}\right), B_3\left(z^{(3)}\right)\right] \to \mathrm{Dense\ layers} \to \hat{y}.$$

The result is an encrypted classifier over fixed spatial features. It is not an end-to-end CNN, because the spatial feature extractor lies outside the trainable network.

2.3 Constraint: the kernels are fixed

In a conventional CNN, backpropagation updates the convolution weights at every optimizer step. Here, Party A computes the convolutions once, before training, so the weights cannot change during encrypted training.

Three arrangements satisfy this constraint. Party A can use predefined filters, such as wavelet scattering transforms. Party A can use a CNN that was trained beforehand and then frozen. Party A can use any other deterministic spatial transformation.

The frozen-extractor arrangement is the most direct implementation:

$$x \xrightarrow{\text{frozen CNN}} h \xrightarrow{T} z \xrightarrow{\text{trainable dense layers}} \hat{y}.$$

Party B trains only the dense layers above the frozen representation.

2.4 Pooling and multiscale representations

Max-pooling and average-pooling have no trainable parameters, so Party A can apply them before encryption without restriction. A pipeline with a rectified linear unit (ReLU) activation takes the form

$$x \to \mathrm{Conv}_{\mathrm{fixed}} \to \mathrm{ReLU} \to \mathrm{Pool} \to T \to \text{encrypted ANN}.$$

Party A can also encrypt the image at several pooled resolutions, such as x, $P_2(x)$, $P_4(x)$, and $P_8(x)$. The network then receives information at several receptive-field scales.

2.5 Inference

Inference repeats the training pipeline exactly. The inference oracle applies the same feature extractor and the same transformation to each submitted example:

$$x' \to \text{feature extractor} \to h' \to T \to z' \to \text{trained network} \to y'.$$

Party B receives no plaintext and no unencrypted feature map at any stage.

2.6 Cost

The orthogonal matrix A has D^2 entries, and convolution enlarges the representation. An MNIST image with two padding rows has 840 dimensions, and its matrix occupies

about 6 MB at 64-bit precision. A $14 \times 14 \times 32$ feature map needs about 315 MB before padding. A $28 \times 28 \times 32$ feature map needs about 5 GB.

Each additional encrypted representation also adds to the transmitted dataset and to the input layer of Party B's network. Deep, globally pooled features avoid most of this cost. A frozen backbone that emits a vector of 512 to 2,048 dimensions needs a matrix of 2 MB to 34 MB.

2.7 Security considerations

This strategy stays within the threat model of the initial Tensorlock manuscript. The attacker still holds only ciphertext and labels, and still receives only labels from the inference oracle. That manuscript's analysis applies to each encrypted representation, with $h^{(k)}$ in the role of the plaintext. Four differences require attention.

Multiple views. Each ciphertext set discloses the Gram matrix of its padded vectors. Several representations of one dataset disclose several Gram matrices, aligned by example. Independent keys prevent direct alignment of coordinates across representations. The combined leakage has not been analyzed.

Covariance knowledge. The initial Tensorlock manuscript finds that meaningful recovery requires exact or near-exact knowledge of plaintext covariance. A public frozen backbone weakens this protection. An attacker can pass similar public images through the same backbone and estimate the feature covariance. Party A can keep the extractor private and treat its weights as key material.

Feature inversion. An attacker who recovers a feature map has not recovered the image. Feature maps from early layers can nevertheless be inverted to approximate images. Deep, pooled features disclose less about the input.

Dimension and value range. Pooled representations have few dimensions, which shrinks the hypothesis space of $2^D - 2$ signal-coordinate subsets. Additional padding restores it. Feature maps are also sparse and unbounded, unlike pixel data. Padding and perturbation settings must be defined for each representation.

3. Oracle-mediated spatial layer processing

3.1 Method

This strategy keeps the convolution kernels trainable. Party B operates on ciphertext and delegates each locality-dependent operation to a **spatial oracle**. The spatial oracle is a service that holds the Tensorlock key. Party A can operate it, or it can run in key-holding client software inside Party A's environment.

The spatial oracle accepts ciphertext and returns ciphertext. In this respect it differs from the inference oracle, which accepts plaintext and returns only a label. This difference governs the security analysis in Section 3.7.

The underlying operation is simple for a key holder. The key holder applies convolution and pooling to transformed data by using the inverse shuffle indices or the transpose of the orthogonal matrix. A separate key serves each array size that pooling creates.

3.2 Forward pass

Let layer l compute $h_{l+1} = F_l(h_l; W_l)$, where F_l is a convolution, a pooling operation, or a short sequence of both. The spatial oracle receives z_l and performs three steps:

$$\hat{h}_l = \left(z_l A_l^{\top}\right)_{S_l}, \qquad h_{l+1} = F_l\left(\hat{h}_l; W_l\right), \qquad z_{l+1} = T_{l+1}(h_{l+1}).$$

For a convolution with activation σ, the middle step is $h_{l+1} = \sigma\left(W_l * \hat{h}_l + b_l\right)$. For max-pooling over regions R_{ij}, it is

$$h_{l+1}[i, j, c] = \max_{(p,q) \in R_{ij}} \hat{h}_l[p, q, c].$$

The transformation T_{l+1} uses a new key sized to the output tensor. Party B sees only z_l and z_{l+1}.

The oracle operates on $\hat{h}_l$, which includes the perturbation noise from the preceding encryption. The oracle can draw new padding and perturbation values at each call, or it can store them for each example. The choice affects training noise and leakage, and it has not been analyzed.

3.3 Backward pass

Trainable kernels require gradients of the loss L, so the spatial oracle takes part in backpropagation. Party B sends the row vector $g_{l+1} = \nabla_{z_{l+1}} L$. The oracle first carries this gradient back through T_{l+1}:

$$\nabla_{h_{l+1}} L = \left(g_{l+1} A_{l+1}^{\top}\right)_{S_{l+1}} .$$

The orthogonal step contributes the transpose, and the padding coordinates are discarded. For bounded additive perturbation, the Jacobian is the identity, except at clipped coordinates, where the gradient is zero.

The oracle then differentiates F_l to obtain $\nabla_{W_l} L$ and $\nabla_{\hat{h}_l} L$. For max-pooling, it uses the stored locations of the selected maxima. Last, it carries the gradient back through the decryption step:

$$\nabla_{z_l} L = E_{S_l}\left(\nabla_{\hat{h}_l} L\right) A_l,$$

where E_{S_l} places the gradient in the signal coordinates and zeros in the padding coordinates. The oracle returns $\nabla_{z_l} L$ to Party B. The oracle must retain, or regenerate, the forward-pass values that differentiation requires.

3.4 Ownership of the convolution weights

Two arrangements are possible.

Oracle-owned weights. The kernels stay inside Party A's system. The oracle computes the forward operation, computes the weight gradients, and applies the updates locally. Party B sees each spatial block as a differentiable module with hidden parameters. This arrangement gives the stronger separation.

Party B-owned weights. Party B sends the current kernels with each call. The oracle returns the encrypted activations, the weight gradients, and the input gradients. Party B keeps control of the whole model. This arrangement increases communication and exposes more information, as Section 3.7 describes.

3.5 Spatial blocks

A call to the oracle after every elementary operation would dominate training time. Several locality-dependent operations can be grouped into one spatial block:

$$\boxed{\text{decrypt} \to \text{Conv} \to \text{ReLU} \to \text{Conv} \to \text{ReLU} \to \text{Pool} \to \text{encrypt}}$$

A complete network then takes the form

$$z_0 \to \boxed{\text{spatial block 1}} \to z_1 \to \boxed{\text{spatial block 2}} \to z_2 \to \text{encrypted dense layers} \to \hat{y}.$$

When Party B performs no computation between consecutive blocks, the blocks merge into one. With oracle-owned weights, the design then reduces to a simpler form. Party A runs a trainable convolutional extractor on plaintext that it already holds, and it sends only the encrypted output:

$$x \xrightarrow{\text{Party A: trainable CNN}} h \xrightarrow{T} z \xrightarrow{\text{Party B: dense layers}} \hat{y}.$$

Party B returns $\nabla_z L$ at each step, and Party A updates the extractor. In this form, the oracle never decrypts ciphertext that Party B supplies. The arrangement corresponds to split learning, with Tensorlock protecting the activations that cross the boundary.

3.6 Inference

In the threat model of the initial Tensorlock manuscript, Party A's inference oracle runs the trained network. Party A therefore holds the convolution weights and the dense layers at inference time. The oracle computes the full forward pass locally and returns only the label. Inference adds no interaction and discloses no ciphertext.

3.7 Security considerations

This strategy extends the threat model of the initial Tensorlock manuscript, and that manuscript's analysis does not cover it. Its argument assumes that the attacker holds a static ciphertext set and receives only labels from the inference oracle. Under that assumption, the attacker obtains no vector equations that involve the key. The spatial oracle changes this assumption in four ways when Party B is adversarial.

Chosen ciphertext queries. In the general protocol, Party B can submit any vector as z_l and observe the vector response z_{l+1}. Each response is a function of the keys. When a block is linear, such as convolution without activation or average-pooling, repeated queries can characterize that function.

Chosen kernels. With Party B-owned weights, Party B also selects F_l. An identity kernel returns a re-encryption of the decrypted input. Kernels that shift the image by one pixel

reveal how spatial translation acts in ciphertext coordinates. Both responses constrain the key.

Weight gradients. With Party B-owned weights, the oracle returns $\nabla_{W_l} L$ in plaintext kernel coordinates. This gradient is a function of plaintext activations.

Chosen gradients and evolving features. In every arrangement, Party B selects the gradients that it returns. With oracle-owned weights, those gradients steer the extractor. Attacks of this kind are known in split learning. The encrypted features also change at every step, so Party B observes a sequence of related Gram matrices.

Several measures reduce this exposure. Oracle-owned weights remove the chosen-kernel and weight-gradient channels. The merged form in Section 3.5 removes chosen ciphertext queries, because the oracle starts from plaintext that it holds. The oracle can limit query rates and record each call in a ledger of transformation events. A trusted execution environment (TEE), which isolates code and data from the host system, can host the oracle.

These measures are not established as sufficient. A deployment of this strategy with an adversarial Party B requires a dedicated security analysis.

4. Comparison

Property	Pre-encryption spatial feature extraction	Oracle-mediated spatial layer processing
Location of spatial operations	Party A, before encryption	Spatial oracle, during training
Learned convolution kernels	No; fixed or pretrained and frozen	Yes
End-to-end CNN training	No	Yes
Interaction during training	None	At every training step
Key use during training	None	Continuous
Added training latency	None	One or more round trips per step
Remote GPU (graphics processing unit) use	Uninterrupted	Interrupted by oracle calls
Network trained by Party B	Dense layers over fixed features	Dense layers, with or without kernels
Relation to the initial threat model	Within it, with the caveats in Section 2.7	Extends it; see Section 3.7
Implementation complexity	Lower	Higher
Most direct use	Frozen extractor with encrypted dense layers	Trainable extractor with encrypted dense layers

The first strategy gives up learned spatial filters and removes interaction. The second strategy keeps learned spatial filters and requires an interactive key holder.

5. Limitations

This white paper describes design concepts and does not report experimental results.

The security considerations in Sections 2.7 and 3.7 identify exposures but do not bound them. Attacks on other data-encoding methods have shown that such transformations can contain structure that initial analysis misses. The initial Tensorlock manuscript does not claim information-theoretic privacy or general ciphertext indistinguishability, and that limitation applies to both strategies.

6. Conclusion

Tensorlock can support CNN-style processing in two ways. Pre-encryption spatial feature extraction moves fixed convolution and pooling to the data owner. It preserves non-interactive training and the initial threat model, and it gives up learned kernels. Oracle-mediated spatial layer processing keeps learned kernels through an interactive key holder, and it requires a security analysis of its own.