Deploying Convolutional Networks on Untrusted Platforms Using 2D Holographic Reduced Representations

Mohammad Mahmudul Alam [1] **Edward Raff** [1 2 3] **Tim Oates** [1] **James Holt** [2]

Abstract

Due to the computational cost of running inference for a neural network, the need to deploy the inferential steps on a third party's compute environment or hardware is common. If the third party is not fully trusted, it is desirable to obfuscate the nature of the inputs and outputs, so that the third party can not easily determine what specific task is being performed. Provably secure protocols for leveraging an untrusted party exist but are too computational demanding to run in practice. We instead explore a different strategy of fast, heuristic security that we call *Connectionist Symbolic Pseudo Secrets*. By leveraging Holographic Reduced Representations (HRR), we create a neural network with a pseudo-encryption style defense that empirically shows robustness to attack, even under threat models that unrealistically favor the adversary.

1. Introduction

As convolutional neural networks (CNN) have become more popular, so to have the concerns around their deployment. Many tricks like low-precision floats, pruning of weights, and classic software engineering and performance tuning have been employed to reduce these computation costs. Still, it is often necessary to deploy a model on third-party compute hardware or cloud environments for a variety of reasons (e.g., lower latency to customers, lack of computing resources, and elasticity of computing demand). In these situations, there are cases where the owner of the model does not fully trust the third party and desires to obfuscate information about the model running on this untrusted platform.

The current solutions to this situation naturally come from the encryption community, and the tools of Secure Multiparty Computation (SMC) (Kerschbaum, 2006; Du & Atallah, 2001) and Homomorphic Encryption (HE) (Gilad-Bachrach et al., 2016) provide methods for running programs on untrusted hardware that guarantee the privacy of the results. These are valuable tools, but computationally demanding and limiting. They often require restrictions on even basic CNN functionality like avoiding softmax activation and sigmoid/tanh non-linearity, limits on the size of the computation itself, and can dwarf the compute time saved by offloading to the third party. Especially when providers charge by compute-hours, this makes SMC and HE tools impractical when computing and latency constraints are a factor, or when neural networks are very large.

Current approaches to untrusted inference are all slower than simply running the computation locally, making them impractical. For this reason, our work scarifies provable security for empirical security, by developing an approach to insert "secrets" into a network's input that can be later extracted, yet obfuscate the input/output to the untrusted party in an encryption-like manner. We emphatically stress this is not strong encryption, but empirically we observe a realistic adversary's attacks are at random-guessing performance, and an unrealistically powerful adversary fairs little better. We term our approach CONNECTIONIST SYMBOLIC PSEUDO SECRETS (CSPS)[1], and compared to the fastest alternative (Mishra et al., 2020). CSPS is $5000\times$ faster and transfers $18,000\times$ less data, making it practically deployable.

To summarize, we leverage inspirations from encryption and neuro-symbolic methods to symbolically represent a one-time pad strategy from the encryption literature within a neural network. The rest of our paper is organized as follows. First, we will review work related to our own in section 2. Our approach uses a Vector Symbolic Architecture (VSA) known as the Holographic Reduced Representations (HRR) from more classical symbolic AI work that may not be

[1]Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD, USA [2]Laboratory for Physical Sciences, College Park, MD, USA [3]Booz Allen Hamilton, McLean, VA, USA. Correspondence to: Edward Raff <Raff_Edward@bah.com>, Tim Oates <oates@cs.umbc.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

[1]Our code can be found at https://github.com/NeuromorphicComputationResearchProgram/Connectionist-Symbolic-Pseudo-Secrets

familiar to all readers, so we will review them briefly in section 3. This will allow us to discuss our method CSPS and how we develop a mechanism for inserting a secret "one-time pad" into a network input and extracting it from the output in section 4. This produces a 5000× speedup and $18,000\times$ reduction in data transfer compared to the fastest alternatives, providing the first speedup for untrusted computation, as shown in section 5. In addition, we show an overly powerful adversary is empirically only slightly better than random guessing, providing practical security for many applications, and extensive ablation studies over six alternative design choices that validate our approach. Finally, we conclude in section 6 with a discussion of the limitations of our approach. Most notably that *we are not implementing true strong encryption*, and so must temper expectations where privacy is a critical requirement.

2. Related Work

The desire to hide the details of a program's inputs and outputs from third-party performing the computation has been studied for many decades by the security and cryptography communities. These methods have been naturally adapted to deep learning tasks, providing provable privacy guarantees. Unfortunately, the high costs of these methods prevent them from being useful when there is any compute or runtime constraint, often requiring multiple order-of-magnitude slowdowns. We review the primary approaches.

The first approach that has been used is (Fully) Homomorphic Encryption (FHE), which allows recasting any program into a new version that takes encrypted inputs, and produces encrypted outputs, providing strong privacy. However, this conversion process can result in extreme computational cost, often requiring arbitrary-precision integer arithmetic[2]. To make FHE "practical", restrictions on the size, depth, and activation functions have been necessary to minimize these compute overheads. For example, (Gilad-Bachrach et al., 2016) required squared activations ($\sigma(x) = x^2$) to perform MNIST in an hour per datum. Current FHE methods, through a mix of network and FHE optimizations, can scale to CIFAR 10 (Brutzkus et al., 2019), but result in networks slower and less accurate than our CSPS. We are not aware of any works that have scaled past CIFAR-10 for FHE-based inference (Chou et al., 2018; QaisarAhmadAlBadawi et al., 2020; van Elsloo et al., 2019; Nandakumar, 2019; Esperanca et al., 2017).

The second broad class of approaches is protocol-based, requiring multiple rounds of communication where data is sent back-and-forth between the host that is requesting computation, and the third party server performing the bulk of computation. Methods like Secure Multi-party Computation (SMC) (Kerschbaum, 2006; Du & Atallah, 2001) and other "protocols" are developed on top of "Oblivious Transfer" (OT), a primitive by which a *sender* and *receiver* exchange messages (Rabin, 1981). Many OT protocols[3] have been customized for deep learning applications (Riazi et al., 2018; Rouhani et al., 2018; Chandran et al., 2019; Riazi et al., 2019; Liu et al., 2017; Mohassel & Zhang, 2017), but suffer similar limitations to FHE. They require minutes of computation per data point prediction, require multiple rounds of computation (a problem for deployment with any limited bandwidth or high latency network), and must send large "messages" on the order of hundreds of megabytes per prediction. We note that our approach requires only one round of communication, the messages are the same size as the original data points and can perform predictions in milliseconds.

Hybrid approaches combining OT and FHE have been developed (Juvekar et al., 2018; Mishra et al., 2020) and are faster than only OT or FHE, but they have not yet overcome the compute, multiple rounds of communication, and scaling limitations that prevent practical use.

The most similar approach to our own work is InstaHide (Huang et al., 2020), which randomly combines training instances with a second population of images. These mixed images (and labels) are sent to a third party for *training*, in an attempt to hide the true training task from the third party. Carlini et al. (2021) showed how to break InstaHide and proved learning bounds indicating the impossibility of the approach. The key failure of InstaHide being a dual problem that: 1) the random additions are highly structured natural images, creating an attack avenue and 2) the goal is third party training, which requires InstaHide to provide the mixed image, leaving only 4 parameters a "secret" per image. Our focus on HRRs allows unstructured secrets making attack harder, and the focus on inference of a trained model allows us to hide a large secret from the third party. We perform extensive customized attacks against CSPS to show we do not suffer the same failing, and provide learning bounds in the linear case that show we do not suffer the same conditions identified by (Carlini et al., 2021).

We note that to the best of our knowledge, our work is the only approach seeking an approximate solution to the problem. This means our method should not be used when privacy is of extreme importance to be "mission critical". Still, we do obtain empirically good privacy in our results, and we show that our method is the only approach practically deployable when runtime or latency is a requirement.

[2]Also called "bignum" or "big-integer".

[3]We note that there are many different classes of protocols involved in these works, and our related work is oversimplifying them to be *just* "OT", but a full description of the different nuances would not aid the reader in understanding our approach, and all prior work share the same fundamental limitations.

In particular, for all prior work cited *the time it takes to run the FHE or OT protocols are orders of magnitude greater than the time to compute the result locally*. Our work is the first that we are aware of to present a method that enters the positive direction on the runtime trade-off.

3. Technical Background

Holographic Reduced Representations (HRR) is a method of representing compositional structure using circular convolution in distributed representations (Plate, 1995). Vectors can be composed together using circular convolution which is referred to as a *binding* operation. Using the original notation for binding $\circledast$ of Plate's (1995) paper, the binding operation is expressed in eq. 1, where $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^d$ are arbitrary vector values, $\mathcal{F}(\cdot)$ and $\mathcal{F}^{-1}(\cdot)$ are the Fast Fourier Transform and its inverse, respectively.

$$\mathcal{B} = \mathbf{x}_i \circledast \mathbf{y}_i = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}_i) \odot \mathcal{F}(\mathbf{y}_i)) \tag{1}$$

$\mathcal{B} \in \mathbb{R}^d$ is the bound term comprised of $\mathbf{x}_i$ and $\mathbf{y}_i$. Two things makes HRR intriguing and valuable: the use of circular convolution, which is commutative, and its ability to retrieve bound components. The retrieval of bound components is referred to as *unbinding*. A vector can be retrieved by defining an inverse function $\dagger : \mathbb{R}^d \rightarrow \mathbb{R}^d$ and identity function $\mathcal{F}(\mathbf{y}_i^\dagger) \cdot \mathcal{F}(\mathbf{y}_i) = \vec{1}$ which gives $\mathbf{y}_i^\dagger = \mathcal{F}^{-1}\left(\frac{1}{\mathcal{F}(\mathbf{y}_i)}\right)$. Using the inverse of vector $\mathbf{y}_i$, another component of the bound term can be approximately retrieved by $\mathbf{x}_i \approx \mathcal{B} \circledast \mathbf{y}_i^\dagger$

These properties are interesting because they hold in expectation even if $\mathcal{B}$ is defined with multiple terms, i.e., $\mathcal{B} = \sum_{i=1}^{k} \mathbf{x}_i \circledast \mathbf{y}_i$, or when composed with hierarchical structure. This allows composing complex symbolic relationships by assigning meaning to arbitrary vectors, staying in a fixed d-dimensional space. As the number of terms bound or added together increases, the noise of the reconstruction $\mathbf{x}_i'$ will also increase. To make these properties work we will use initialization conditions proposed by (Ganesan et al., 2021), where $\mathbf{x}_i, \mathbf{y}_i \sim \pi(\mathcal{N}(0, 1/d))$, where $\pi(\cdot)$ is a projection onto the ball of complex unit magnitude $\pi(\mathbf{y}_i) = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{y}_i)}{|\mathcal{F}(\mathbf{y}_i)|}\right)$, and $\mathcal{N}(\mu, \sigma^2)$ is the Normal distribution.

4. Connectionist Symbolic Pseudo Secrets

Our approach to make Connectionist Symbolic Pseudo Secrets requires two steps. First, we introduce a simple modification of the HRR from 1-D to 2-D, exploiting a property of its construction so that we can embed the secret into the inputs in such a manner that they are likely to be preserved by the network. Then we design a training approach to use the symbolic behavior of HRR to bind the input, and then unbind the output, such that the majority of work can be done by a remote 3rd party.

4.1. 2D HRR As Pseudo One-Time Pad

Our first insight comes from the fact that in Equation 1, the result $\mathcal{B} = \boldsymbol{x} \circledast \boldsymbol{s}$ is a simple linear operation that at infinite precision is invertible, giving $\boldsymbol{s} = \boldsymbol{x}^\dagger \mathcal{B}$. Thus if we have $\boldsymbol{x}$ represent the image (network input) we wish to obscure, and we have a random secret $\boldsymbol{s}$ to apply, then the resulting $\mathcal{B}$ object will appear random in nature. And for any bound output $\mathcal{B}$, there are infinite possible image/secret pairs that will produce the exact same output[4]. This allows for a "one-time pad" kind of approach to obscuring the input to the network from the untrusted party. If we can preserve the secret $\boldsymbol{s} \in \mathcal{B}$ as $\mathcal{B}$ is processed by a neural network, we can attempt to extract it using the unbinding operation at the end. Phrased mathematically, if $f(\cdot)$ is a normal CNN, we desire a secure function $\tilde{f}(\cdot)$ such that $\tilde{f}(\boldsymbol{x} \circledast \boldsymbol{s}) \circledast \boldsymbol{s}^\dagger \approx f(\boldsymbol{x})$, yet $\tilde{f}(\boldsymbol{x} \circledast \boldsymbol{s})$ appears random. A critical part of this is to maintain the information within $\boldsymbol{s}$.

We can achieve this by recognizing that the HRR operation is equivalent to a 1D convolution over a sequence, but the 1D convolution is not an important property. By simply switching to 2D Fourier Transforms, we instead perform 2D convolutions to bind our secret, which aligns the resulting $\boldsymbol{x} \circledast \boldsymbol{s}$ with the 2D CNN that will process it. By construction, this retains all the symbolic properties of HRR, as experimentally shown in Figure 2, but allows the inputs to behave in a manner consistent with a CNN.

This is critical as it means the binding operation $\circledast$ is equivalent to another convolutional layer of the network, where the secret $\boldsymbol{s}$ is a user-chosen weight matrix rather than a learned one. This also means subsequent layers of convolution and pooling may learn to retain the structure of $\boldsymbol{s}$ to a sufficient degree that it can be extracted later, and effectively obfuscates the nature of the input as shown in Figure 3.

4.2. Network Design

We now specify our novel approach to a network architecture that leverages 2D HRR to hide the output, while offloading $\approx 75\%$ of computation onto a remote third party. The proposed method has three networks, one larger backbone network $f_W(\cdot)$ that performs the "work" of feature extraction, and two identical smaller networks. As for the main network $f_W(\cdot)$, U-net CNN architecture (Ronneberger et al., 2015) has been employed, and at deployment would be run by the untrusted party. There are multiple benefits

[4]The output is not uniformly random, a key difference from a true one-time pad.

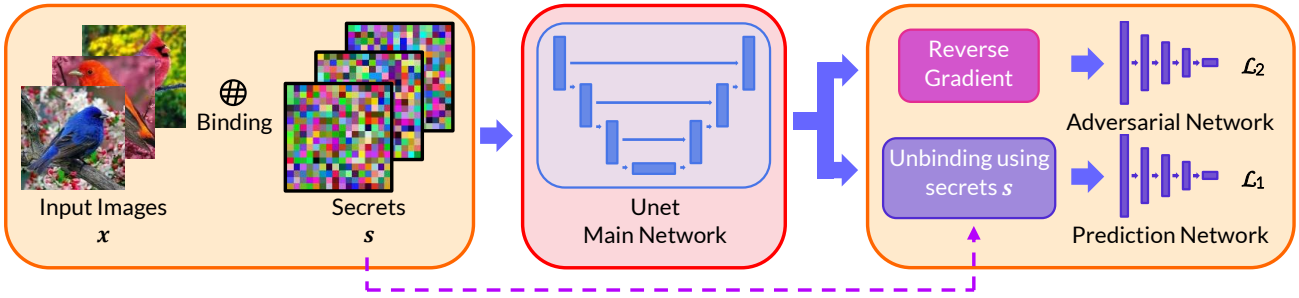


Figure 1: Block diagram of encryption process of the CNN using improved 2D HRR with three stages. Both of the orange regions are on the user-end. The secrets to unbind the images and outputs of the main network are only shared in these regions (dashed line). The red region indicates the untrusted third party who will run the main network after it has been trained.

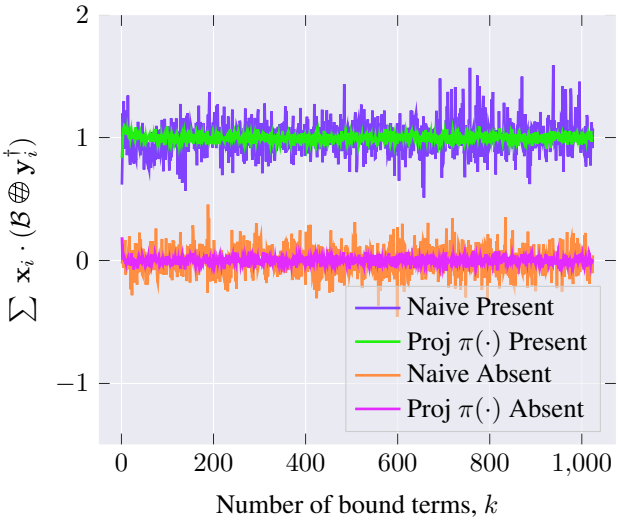
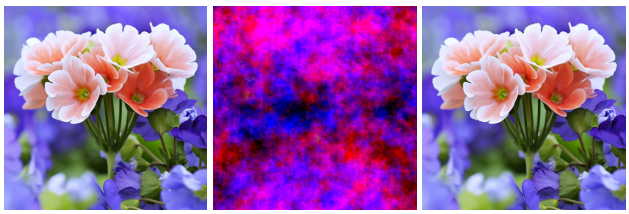


Figure 2: Binding and unbinding terms using improved 2D HRR where for the terms that are present, $\mathbf{x}_i \circledast \mathbf{y}_i \in \mathcal{B}$, the output along the y-axis is close to 1 and for the absent terms, $\mathbf{x}_i \circledast \mathbf{y}_i \notin \mathcal{B}$, output is close to 0. Retrieval without using projection to the input is referred to as *Naive present* and *absent* shown in violet and orange color. Present and absent terms output with projection is shown in green and pink, respectively.

of using U-net architecture. It is a deep CNN with identical input and output shapes so that our secret vector $\boldsymbol{s}$ can be used on the input to and output to $f_W(\cdot)$. This requirement is because the secret $\boldsymbol{s}$ needs to be matched with the shape of the output of the network. The client computes $\hat{\boldsymbol{x}} = \boldsymbol{x} \circledast \boldsymbol{s}$, sending $\hat{\boldsymbol{x}}$ to the third party while keeping the randomly chosen $\boldsymbol{s} \sim \pi\left(\mathcal{N}(0, 1/\sqrt{W \times H \times D})\right)$ a secret. The provider sends back the result $\boldsymbol{r} = f_W(\hat{\boldsymbol{x}})$.

Afterward, two identical classification networks are de-



(a) Original Image (b) Bound Image (c) Retrieved Image

Figure 3: A sampled image $\boldsymbol{x}$ in (a) bound with a secret $\boldsymbol{s}$ in (b) using improved 2D HRR. The original image is retrieved using $\boldsymbol{s}^\dagger(\boldsymbol{x} \circledast \boldsymbol{s}) \approx \boldsymbol{x}$ is shown in (c).

signed, as shown in the third stage of Figure 1. One is the **Prediction Network** $f_P(\cdot)$ which is classifying after unbinding the main network outputs, giving the prediction $\hat{y}_P = f_P(\boldsymbol{r} \circledast \boldsymbol{s}^\dagger)$. The other network is the **Adversarial Network** $f_A(\cdot)$ which attempts to perform the prediction task without access to the secret $\boldsymbol{s}$, computing it's own prediction $\hat{y}_A = f_A(\boldsymbol{r})$. Both networks are used during training, but we reverse the gradient sign (i.e., multiply by -1) from $f_A(\cdot)$ back to the backbone network $f_W(\cdot)$. Ganin et al. (2016) introduced this approach as a form of domain adaption, but we instead use it to enforce that the secret $\boldsymbol{s}$ be necessary to extract meaning from the base network f_W. This gradient reversal will ensure that $f_A(\cdot)$ is attempting to minimize the same predictive loss, but the change in gradient sign means $f_W(\cdot)$ receives a learning signal sending it's optimization in the opposite direction - discouraging it from retaining any information that would be useful to $f_A(\cdot)$, but still receiving the correct gradient from $f_P(\cdot)$.

Combined, the binding $\hat{\boldsymbol{x}} = \boldsymbol{x} \circledast \boldsymbol{s}$ ensures that the input to $f_W(\cdot)$ is random in appearance and is not discernible on its own, and the gradient reversal on $f_A(\cdot)$ ensures the output $\boldsymbol{r} = f_W(\hat{\boldsymbol{x}})$ is also not informative on its own. A new secret $\boldsymbol{s}$ is sampled for every prediction so that the third

party cannot collect multiple samples to try and discover any "single key". This design heuristically provides the components of a secure protocol, but uses only standard operations built into all modern deep learning frameworks, giving it minimal overhead compared to prior approaches outlined in section 2. The overall training procedure is given in Algorithm 1.

Algorithm 1 CONNECTIONIST SYMBOLIC PSEUDO SECRETS Training using a dataset with images of size $W \times H \times D$ using a loss function $\ell(\cdot, \cdot)$.

for $\boldsymbol{x}_i, y_i \in$ dataset **do** ▷*Training loop*
  $\boldsymbol{s} \sim \pi\left(\mathcal{N}(0, 1/\sqrt{W \times H \times D})\right)$ ▷*New secret*
  $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{x}_i \circledast \boldsymbol{s}$ ▷*Obfuscated input*
  $\boldsymbol{r} \leftarrow f_W(\hat{\boldsymbol{x}})$ ▷*Run by 3rd party after training*
  $\hat{y}_P \leftarrow f_P\left(\boldsymbol{r} \circledast \boldsymbol{s}^\dagger\right)$ ▷*Used locally after training*
  $\hat{y}_A \leftarrow \text{REVERSEGRAD}(f_A(\boldsymbol{r}))$ ▷*Discarded after training*
  $\mathcal{L} \leftarrow \ell(y, \hat{y}_P) + \ell(y, \hat{y}_A)$ ▷*Incur training loss*
  Back-propagate on the loss $\mathcal{L}$
  Run optimization step

The main network $f_W(\cdot)$ has four U-Net rounds in every experiment and doubles from 64 filters after each round, reversing for the decode. The $f_A(\cdot)$ and $f_P(\cdot)$ are always identical, with 3 rounds of aggressive convolution followed by pooling to minimize compute costs and shrink the representation, followed by two fully connected hidden layers. Mini-ImageNet receives a fourth round of pooling due to its larger resolution. All network details and code can be found in Appendix A. We further perform extensive ablation studies in subsection D.1 looking at different binding operations (HRR without projection, 1D HRR, 1D HRR with Hilbert Curves, and the vector-derived transformation binding (VTB)) and network designs (Residual style) that show our design of U-Net with 2D HRRs is critical to obtaining high predictive accuracy.

While our results are heuristic for the deep neural networks, we provide theoretical evidence for our approach by analyzing the linear case. Given an adversary who has all n bounded inputs $\hat{\mathbf{x}}_i$ with the true labels y_i, the problem is likely not linearly learnable due to an $\mathcal{O}(n)$ Rademacher complexity, as we show in Theorem 4.1.

Theorem 4.1. *Learning $\boldsymbol{w}^\top \boldsymbol{x}_i \circledast \boldsymbol{s}_i$ without the secrets $\boldsymbol{s}_i$ has a non-trivial Rademacher complexity of $\mathcal{O}(n)$,*

Proof. The CSPS Rademacher model gives $\frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{w}\in\mathbb{R}^d | \|\mathbf{w}\|_2 \leq 1} \sum_{i=1}^n \sigma_i \mathbf{w}^\top \boldsymbol{x}_i\right]$. The binding operation with a vector $\boldsymbol{s}_i$ is equivalent to the matrix-vector product with a corresponding circulant matrix S_i^C. Each $\mathbf{w}^\top S_i^C$ can be written as an independent random rotation leading to n independent $\hat{\mathbf{w}}_i$ terms, allowing the supremum to move into the summation to give $\frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sum_{i=1}^n \sup_{\tilde{\mathbf{w}}_i \in \mathbb{R}^d | \|\mathbf{w}_i\|_2 \leq 1} \sigma_i \tilde{\mathbf{w}}_i^\top \boldsymbol{x}_i\right]$. Applying the result for n independent linear models (Shalev-Shwartz & Ben-David, 2021) trained on one point gives the final complexity $\sum_{i=1}^n \|\boldsymbol{x}_i\|_2 \leq n \max_i \|\boldsymbol{x}_i\|_2$. □

5. Experiments & Results

We do not argue that CSPS is any true form of encryption, only that it is empirically effective at hiding the nature of inputs and outputs sent to an untrusted party. We will demonstrate this through a series of experiments to show that: 1) Compared to a network with the same design but without the HRR binding/unbinding of the secret $\boldsymbol{s}$, that our approach has some loss of accuracy but is more accurate than prior approaches. 2) The loss of accuracy can be largely mitigated by averaging the results of ≤ 10 queries. 3) Our approach is robust to adversaries using unsupervised learning that try to infer class information. 4) Our approach is still robust to unrealistically strong adversaries that know the training data and classes, obtaining $1.5 - 4.7\times$ random guessing accuracy. 5) Our approach is up to $290\times$ faster than existing provable methods. We also perform an extensive ablation study of alternative designs that show our approach performs considerably better than alternatives. Before our results, we briefly review the datasets and training details.

As the proposed method is doing image classification, various well-known image classification datasets are used for the experiments. These datasets are diverse in shape, color, channels, contents, and the number of classes. In total 5 image classification datasets are utilized, namely, MNIST, SVHN, CIFAR-10, CIFAR-100, and Mini-ImageNet. Dataset details, along with training time, and data augmentation can be found in Appendix B.

5.1. Accuracy Results

Our results focus on Top-1 classification accuracy for all datasets, and Top-5 accuracy for datasets with 100 classes. We start by demonstrating the accuracy of our approach in Table 1, where "Base" indicates a network with the same total architecture (including U-Net backbone), but without any of the binding/unbinding of secrets or gradient reversal of the adversarial network. This shows that 1) our method can scale to Mini-ImageNet, a result not previously possible (Mishra et al., 2020), and 2) that there is some cost that our secret binding incurs on the accuracy of the result.

The results in Table 1 are for a single attempt at the prediction process, and noise is introduced by the randomly selected secret $\boldsymbol{s}$. We can average out this noise by sending k inputs $\boldsymbol{x} \circledast \boldsymbol{s}_1, \boldsymbol{x} \circledast \boldsymbol{s}_2, \ldots, \boldsymbol{x} \circledast \boldsymbol{s}_k$ to be classified, and averaging the resulting k predictions. This provides the

Table 1: Accuracies of the Base model, and model with secret binding and unbinding using improved 2D HRR.

Dataset	Model	Top-1	Top-5
MNIST 28×28	Base	98.80	–
	CSPS	98.51	–
SVHN 32×32	Base	93.76	–
	CSPS	88.44	–
CIFAR-10 32×32	Base	83.57	–
	CSPS	78.21	–
CIFAR-100 32×32	Base	62.59	86.99
	CSPS	48.84	75.82
Mini-ImageNet 84×84	Base	55.73	80.55
	CSPS	40.99	66.99

result given in Figure 4, showing that $k \leq 10$ is sufficient to almost completely eliminate the accuracy drop. Additional discussion of this result is in Appendix C. We note the lower accuracy numbers compared to more modern networks on these problems comes from the difficulty of learning with random $\boldsymbol{s}$ vectors bound to the input. For example, our CIFAR-10 training accuracy is 86.17%, which is close to the test accuracy. For this reason overfitting does not appear to be a culprit in the results. The difficulty of the trained network to work with random $\boldsymbol{s}$ vectors also provides intuition as to its success as a defense: the attacker with less access should have more difficulty handling the HRR vectors, and thus inhibits their success.

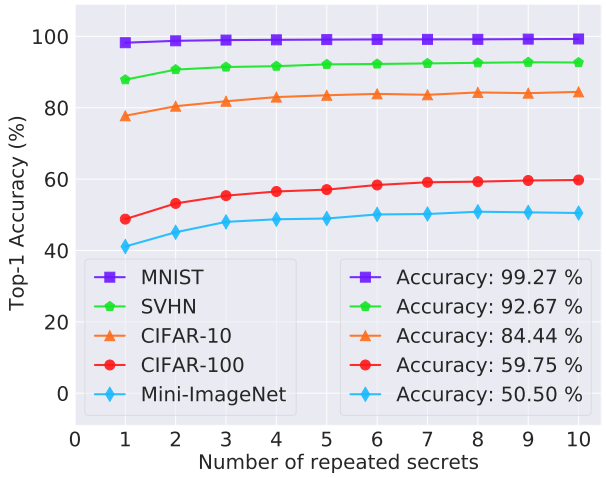


Figure 4: Accuracy (y-axis) of CSPS after averaging k predictions (x-axis), which almost fully restores the accuracy lost due to the secret binding/unbinding.

Table 2: Time to perform prediction on each dataset in its entirety, where the Homomorphic Encryption alternative is a single CNN layer and unrealistically small to minimize its runtime at the cost of all predictive accuracy.

Dataset	Our CSPS	HE Est.
MNIST	4.56 Seconds	2 Hours 46 Minutes
SVHN	12.44 Seconds	55 Hours 32 Minutes
CIFAR-10	7.58 Seconds	21 Hours 20 Minutes
CIFAR-100	9.07 Seconds	43 Hours 53 Minutes
Mini-ImageNet	28.37 Seconds	Timeout

Table 3: Amount of computation performed by the local user and the remote third party

Dataset	Remote %	Local %
MNIST	74.24	25.76
SVHN	65.06	34.94
CIFAR-10	66.08	33.92
CIFAR-100	66.78	33.22
Mini-ImageNet	74.42	25.58

5.2. Run-time Results

To show the speed advantages of CSPS, we perform a comparison with HE that is *unrealistically favorable to HE*. HE has significant design constraints for a neural network, and thus we could not replicate our "Base" architecture with HE libraries like (Benaissa et al., 2021). Instead, we compare against a HE network with a single convolutional layer, followed by aggressive pooling and a fully connected layer, making the model extremely small, unable to learn with any predictive accuracy, and unusable with performance that is close to random guessing (except on MNIST). The results are in Table 2, where Mini-ImageNet failed to make a single prediction in under 24 hours. These settings are overly idealized for HE, and is still $290\times$ slower than our CSPS.

Because CSPS uses standard deep learning code, there is no extraneous compute overhead for arbitrary precision math or multiple rounds of network communication. Thus we can look at the amount of compute saved by offloading to a remote party in Table 3. We see that at least 65% of compute can be offloaded, netting a $2.9-3.5\times$ reduction in cost. This cost savings can be important for low-power, battery, or compute constrained devices. The fastest prior work by (Mishra et al., 2020) requires 60 MB of extra communication *per prediction* on CIFAR-10 (that is $18,000\times$ larger than the image being worked on, the entire corpus is only 200 MB), and is reported to be $5019\times$ slower than our ap-

proach. To the best of our knowledge, CSPS is thus the only method that can realize a real-world resource reduction.

5.3. Realistic Adversary

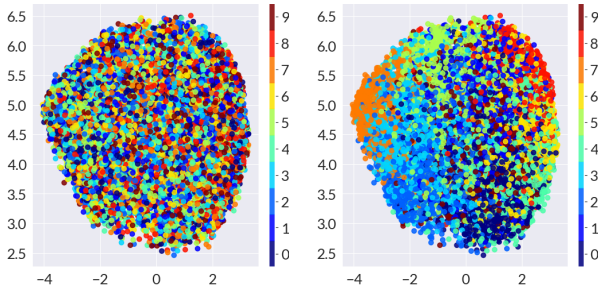
Following Biggio et al. (2014) we specify a realistic adversary that seeks to infer the nature of our model's outputs and class distribution. Because the output shape $\boldsymbol{r} \in \mathbb{R}^{W \times H \times D}$ has no relationship with the number of classes, they must attempt to use some unsupervised clustering to identify patterns within the predictions. We have applied several diverse clustering algorithms such as Kmeans (Arthur & Vassilvitskii, 2006; Raff, 2021), Spectral (Ng et al., 2002), Gaussian Mixture Model (GMM), Birch (Zhang et al., 1996), and HDBSCAN (Malzer & Baum, 2020) cluster to the outputs $\boldsymbol{r} = f_W(\cdot)$ that the adversary has access to. If they are able to perform clustering with greater than random chance, they may be able to extract information about how our model works. We pessimistically assume the adversary knows the exact number of clusters k that they should be looking for. Thus we use the Adjusted Rand-Index (ARI) to score how well the clusters perform with respect to the true class labels (Vinh et al., 2010). A near 0% ARI indicates there is no information to be extracted , and thus our clustering has performed well. We also consider the case where the adversary clusters on the bounded inputs they receive, $\hat{x}$, for completeness. We note this is a poor attack avenue in realistic settings because multiple classes may exist per given set of input images, and that information is only leaked by the network $f_W(\cdot)$ and not the inputs.

The results are in Table 4, where the adversary performs best on MNIST with $\leq 1.5\%$ ARI scores. On all other datasets, the score is $\leq 0.2\%$, indicating there is almost no label information to be extracted from the clusters using existing methods. This shows our approach is effective in hiding the nature of the output and predictions from the untrusted party. Note that on HDBSCAN zero scores are obtained because of degenerate results. HDBSCAN has the concept of "outliers" that do not belong to any cluster and assigns almost the entire test set to the "outlier" class, resulting in worst-case scores. We note $\hat{x}$ is larger than $\boldsymbol{r}$, resulting in Spectral clustering timing out after several days of running. Overall the results clearly demonstrate that minimal amount of label information is leaked by the model.

Figure 5 shows visually how CSPS is able to achieve this result. The result vectors $\boldsymbol{r}$ that the untrusted party has access to are plotted using UMAP (McInnes et al., 2018). Because $f_A(\cdot)$ is trained with gradient reversal, $f_W(\cdot)$ has learned a representation $\boldsymbol{r}$ that requires the secret $\boldsymbol{s}$ to extract the meaning from its representation. Thus the result is an embedding space that points from the same class are randomly dispersed and intermixed. When it is clustered the clusters do not correspond to the true class distributions, as shown on the right. Additional visualizations showing how the cluster labels do not correlate with class labels in Appendix D.

Table 4: Clustering results of the adversary attempting to discover class information directly from the main network output $\boldsymbol{r}$ (top) and the bound image inputs $\hat{\boldsymbol{x}}$ (bottom). All numbers are percentages, and the Adjusted Rand-Index is $\leq 1.5\%$ for all cases. Since ARI accounts for random-chance clustering, the unrealistic adversary is not able to meaningful discern class information.

	MNIST	SVHN	CIFAR 10	CIFAR 100	Mini ImgNet
K-Means	1.28	0.06	0.21	0.03	0.08
Spectral	0.01	0.01	0.00	0.00	0.02
GMM	1.28	0.06	0.17	0.04	0.09
Birch	1.51	0.03	0.13	0.05	0.07
HDBSCAN	0.00	0.00	0.00	0.00	0.00
K-Means	-0.02	-0.01	0.18	0.54	0.42
GMM	0.01	0.00	0.09	0.61	0.44
Birch	0.20	0.00	0.14	0.45	0.35
HDBSCAN	0.00	-0.24	1.23	0.01	0.02



(a) True Class Labels (b) Birch Cluster Labels

Figure 5: UMAP embeddings of output r of $f_W(\cdot)$ (left) and the "best" clustering (right). Without the secret $\boldsymbol{s}$ the class labels appear random (left), and clustering detects spurious density patterns with no correlation to the true labels.

5.4. Overly Strong Adversary

An unrealistically powerful adversary would have access to the entire training set, the class labels, know the procedure of binding/unbinding secrets, and be able to train their own model that predicts the class label from the intermediate result $\boldsymbol{r}$ (i.e., knows everything but the secrets $\mathbf{s}_i$). This is in fact what the adversarial network $f_A(\cdot)$ performs, and represents the worst possible case scenario for the adversary's strength: that they can train their own model to ignore the secret $\boldsymbol{s}$ and extract the true labels. We thus perform this test by training a new model on the ground-truth pairs between $\boldsymbol{r}_i$ and the class label $\boldsymbol{y}_i$, with the results shown in Table 5.

Note an adversary of this strength already knows what the predictive task and type of data is, which is what CSPS is designed to hide. Success of CSPS at this strength shows efficaciousness from task level to individual level protection beyond our intended goal, but also provides even greater evidence of task level protection.

Table 5: Accuracies of the **Adversarial Network** (lower is better) where the secret to unbind the output of the U-Net is unknown.

Dataset	Top-1 Accuracy (%)	Top-5 Accuracy (%)
MNIST	19.72	–
SVHN	21.13	–
CIFAR-10	12.91	–
CIFAR-100	2.66	10.33
Mini-ImageNet	4.68	15.01

In this worst case situation, the adversary's predictions are little better than random-guessing performance. For MNIST, SVHN, and CIFAR-10 that would be 10%, with SVHN having the best results at just $2.1\times$ better than random-guessing. For CIFAR-100 and Mini-ImageNet random-guessing is 1%, and we see high ratios at 2.6 and $4.7\times$, but the total accuracy is still far below the 60% and 51% obtainable by knowing the secret $\boldsymbol{s}$. This shows that our approach is highly effective at obscuring the information from the untrusted party, forcing them near random-guessing performance even in unrealistically powerful settings. This also reinforces that our approach is not real encryption, and should not be used when provable security is a requirement. Our results do indicate strong empirical security though, and the only method that is practical from a runtime perspective. While multiple classifications of the same image with different secrets improves accuracy for the user, the same can not be said for the adversary. Figure 6 shows that attack success improves by a minor amount only on MNIST for a network trained to take in k pairs of an image bound with different secrets.

Table 6: Accuracy predicting the inputs comparing defender CSPS and unrealistic adversary attacking the inputs

Dataset	Our CSPS	Unrealistic Adversary
MNIST	98.51 %	81.23 %
SVHN	88.44 %	39.61 %
CIFAR-10	78.21 %	43.40 %
CIFAR-100	48.84 %	16.58 %
Mini-ImageNet	40.99 %	16.00 %

In the most extreme scenario, the adversary would have access to the bound images along with the class labels and it may train a network to learn class labels directly from the bound images. Results of this experiment are reported in Table 6. Even though this unrealistic adversary seemingly performed better with the access of correct class labels (note linear models can get 92% MNIST accuracy), still without the correct secret to unbind the image, it falls behind our CSPS method, providing a strong level of individual protection.

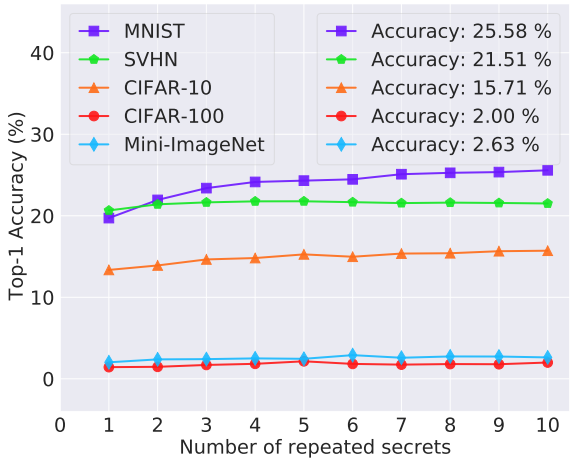


Figure 6: Accuracy (y-axis) of **Unrealistic Adversary** after averaging k predictions (x-axis) of the secret binding/unbinding.

5.5. Model Inversion Adversary

As our final attack, we use the Frechet Inception Distance (FID)(Heusel et al., 2017) to design an inversion attack. Given the bound input $\hat{\boldsymbol{x}} = \boldsymbol{x} \circledast \boldsymbol{s}$, the adversary has their own copy of the training data to compute the FID score of $\hat{\boldsymbol{x}} \circledast \hat{\boldsymbol{s}}^{\dagger}$. Then the adversary can optimize their copy of $\hat{\boldsymbol{s}}$ to try and find the secret that will result in a realistic looking image. We remind the reader that without CSPS, the adversary intrinsically receives the true input $\boldsymbol{x}$ whenever a prediction is made, and so no comparison to a baseline is possible. Our goal is purely to see if an inversion strategy yields cracks in the effectiveness of CSPS.

Examples of the attack are presented in Figure 7, showing that inverting the original images is highly challenging. This attack involves the adversary having the true training data, knowing the procedure to extract the input, and using gradient descent to attempt to find a secret that maximizes the apparent realism of the result via FID scores.

A second inversion strategy assumes the adversary has examples of secrets $\boldsymbol{s}$ and encoded images $\boldsymbol{r}$, and attempts to learn an auto-encoder that minimizes $\|\boldsymbol{r} - \boldsymbol{s}\|_2^2$, to directly predict the secret from a bound image. We find that this strategy also fails to provide meaningful results, as demonstrated in Figure 8.

Figure 7: Model inversion attack by the adversary using projected gradient descent to unbind the bound images given sample of the original images. Images are shown in pairs where the original image is shown in left and generated image is shown in right. Results for MNIST (a), SVHN (b), CIFAR-10 (c), CIFAR-100 (d), and Mini-ImageNet (e) all confirm the adversary can not extract the nature of the bound inputs even when optimizing for visual realism to extract the secret.

Figure 8: Inversion attack using a trained auto-encoder to directly predict the secret $\boldsymbol{s}$. Results for MNIST (a), SVHN (b), CIFAR-10 (c), CIFAR-100 (d), and Mini-ImageNet (e) show that the adversary can not meaningfully predict the secret from new outputs $\boldsymbol{r}$.

6. Conclusion and Limitations

We have shown the first approach that allows deploying the majority of computational effort onto an untrusted third party, that does not impose compute or communication overheads larger than performing the computation locally. This makes it the only current option for deployment scenarios that are runtime or power-constrained, such as on mobile devices. While the method is not provably secure, limiting use to highly sensitive data, it shows strong empirical robustness to unrealistically powerful adversaries. Compared to state-of-the-art alternatives our CSPS is $\approx 5000\times$ faster and sends $\approx 18,000\times$ less data per query.

Our approach is validated by attacking it at a threat model stronger than what we intend it to be used for. We stress this intended treat model of CSPS as a limitation because we can not prove it's effectiveness in the more general context. Our hope is that the numerous attacks at a higher threat model will mean that any future successful attack against CSPS will thus require some scientific advancement, but also means we must acknowledge the risk of unidentified attacks. It should thus be used with high caution for any application where privacy is a hard requirement. The scalability of CSPS is also limited, we found it unable to reach full ImageNet scale in its current form - a limitation shared by other approaches, but still a constraint toward many applications.

Acknowledgments

We thank Frank Ferraro for his helpful review of this paper, and the anonymous reviewers for their additional experiments that have further strengthened the paper's empirical conclusions.

References

Arthur, D. and Vassilvitskii, S. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

Bachlechner, T., Majumder, B. P., Mao, H. H., Cottrell, G. W., and McAuley, J. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.

Benaissa, A., Retiat, B., Cebere, B., and Belfedhal, A. E. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

Biggio, B., Fumera, G., and Roli, F. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, 2014. ISSN 10414347. doi: 10.1109/TKDE.2013.57.

Brutzkus, A., Gilad-Bachrach, R., and Elisha, O. Low Latency Privacy Preserving Inference. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 812–821. PMLR, 2019. URL https://proceedings.mlr.press/v97/brutzkus19a.html.

Carlini, N., Deng, S., Garg, S., Jha, S., Mahloujifar, S., Mahmoody, M., Thakurta, A., and Tramer, F. Is private learning possible with instance encoding? *Proceedings - IEEE Symposium on Security and Privacy*, pp. 410–427, 2021. ISSN 10816011. doi: 10.1109/SP40001.2021.00099.

Chandran, N., Gupta, D., Rastogi, A., Sharma, R., and Tripathi, S. EzPC: Programmable and efficient secure two-party computation for machine learning. *Proceedings*

- *4th IEEE European Symposium on Security and Privacy, EURO S and P 2019*, pp. 496–511, 2019. doi: 10.1109/EuroSP.2019.00043.

Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., and Fei-Fei, L. Faster CryptoNets: Leveraging Sparsity for Real-World Encrypted Inference. *arXiv*, 2018. URL http://arxiv.org/abs/1811.09953.

Du, W. and Atallah, M. J. Secure Multi-party Computation Problems and Their Applications: A Review and Open Problems. In *Proceedings of the 2001 Workshop on New Security Paradigms*, NSPW '01, pp. 13–22, New York, NY, USA, 2001. ACM. ISBN 1-58113-457-6. doi: 10.1145/508171.508174. URL http://doi.acm.org/10.1145/508171.508174.

Esperanca, P., Aslett, L., and Holmes, C. Encrypted accelerated least squares regression. In Singh, A. and Zhu, J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 334–343, Fort Lauderdale, FL, USA, 2017. PMLR. URL http://proceedings.mlr.press/v54/esperanca17a.html.

Ganesan, A., Gao, H., Gandhi, S., Raff, E., Oates, T., Holt, J., and McLean, M. Learning with holographic reduced representations. *Advances in Neural Information Processing Systems*, 2021.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial Training of Neural Networks. *J. Mach. Learn. Res.*, 17(1):2030–2096, Jan 2016. ISSN 1532-4435.

Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 201–210, New York, New York, USA, 2016. PMLR. URL https://proceedings.mlr.press/v48/gilad-bachrach16.html.

Gosmann, J. and Eliasmith, C. Vector-derived transformation binding: an improved binding operation for deep symbol-like processing in neural networks. *Neural computation*, 31(5):849–869, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6629–6640, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Huang, Y., Song, Z., Li, K., and Arora, S. InstaHide: Instance-hiding Schemes for Private Distributed Learning. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4507–4518. PMLR, 2020. URL https://proceedings.mlr.press/v119/huang20i.html.

Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651–1669, Baltimore, MD, aug 2018. {USENIX} Association. ISBN 978-1-939133-04-5. URL https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar.

Kerschbaum, F. Practical Private Regular Expression Matching. In Fischer-Hübner, S., Rannenberg, K., Yngström, L., and Lindskog, S. (eds.), *Security and Privacy in Dynamic Environments: Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22–24 May 2006, Karlstad, Sweden*, pp. 461–470. Springer US, Boston, MA, 2006. ISBN 978-0-387-33406-6. doi: 10.1007/0-387-33406-8_43. URL https://doi.org/10.1007/0-387-33406-8{_}43.

Liu, J., Juuti, M., Lu, Y., and Asokan, N. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pp. 619–631, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134056. URL https://doi.org/10.1145/3133956.3134056.

Malzer, C. and Baum, M. A hybrid approach to hierarchical density-based cluster selection. In *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 223–228. IEEE, 2020.

McInnes, L., Healy, J., and Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., and Popa, R. A. Delphi: A Cryptographic Inference Service

for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. {USENIX} Association, aug 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

Mohassel, P. and Zhang, Y. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38. IEEE, may 2017. ISBN 978-1-5090-5533-3. doi: 10.1109/SP.2017.12. URL http://ieeexplore.ieee.org/document/7958569/.

Nandakumar, K. Towards Deep Neural Network Training on Encrypted Data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

Ng, A. Y., Jordan, M. I., and Weiss, Y. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pp. 849–856, 2002.

Plate, T. A. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641, 1995.

QaisarAhmadAlBadawi, A., Chao, J., Lin, J., Mun, C. F., Jie, S. J., Tan, B. H. M., Nan, X., Khin, A. M. M., and Chandrasekhar, V. Towards the AlexNet Moment for Homomorphic Encryption: HCNN, the First Homomorphic CNN on Encrypted Data with GPUs. *IEEE Transactions on Emerging Topics in Computing*, 2020. ISSN 21686750. doi: 10.1109/TETC.2020.3014636.

Rabin, M. O. How To Exchange Secrets with Oblivious Transfer. *Technical Report TR-81, Aiken Computation Lab, Harvard University*, pp. 1–5, 1981. URL http://dm.ing.unibs.it/giuzzi/corsi/Support/papers-cryptography/187.pdf.

Raff, E. Exact Acceleration of K-Means++ and K-Means‖. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2928–2935, 2021. doi: 10.24963/ijcai.2021/403. URL https://arxiv.org/abs/2105.02936.

Riazi, M. S., Weinert, C., Tkachenko, O., Songhori, E. M., Schneider, T., and Koushanfar, F. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pp. 707–721, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355766. doi: 10.1145/3196494.3196522. URL https://doi.org/10.1145/3196494.3196522.

Riazi, M. S., Samragh, M., Chen, H., Laine, K., Lauter, K., and Koushanfar, F. XONN: XNOR-Based Oblivious Deep Neural Network Inference. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, pp. 1501–1518, USA, 2019. USENIX Association. ISBN 9781939133069.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Rouhani, B. D., Riazi, M. S., and Koushanfar, F. Deepsecure: Scalable Provably-Secure Deep Learning. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357005. doi: 10.1145/3195970.3196023. URL https://doi.org/10.1145/3195970.3196023.

Shalev-Shwartz, S. and Ben-David, S. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2021.

van Elsloo, T., Patrini, G., and Ivey-Law, H. SEALion: a Framework for Neural Network Inference on Encrypted Data. *arXiv*, 2019. URL http://arxiv.org/abs/1904.12840.

Vinh, N. X., Epps, J., and Bailey, J. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *J. Mach. Learn. Res.*, 11:2837–2854, dec 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1953024.

Zhang, T., Ramakrishnan, R., and Livny, M. Birch: An efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.

A. Network Model Details

To train the proposed model, first both training and augmented data are bound with a randomly generated secret sampled from a normal distribution with complex unit magnitude projection using improved 2D HRR. The bound image is then transferred to the main network, i.e., U-net. Next, the output of the U-net is received to the user-end and exploited by both the prediction and the adversarial network. In the prediction network, the output of the U-net is unbound using the secret, and class labels are predicted. On the other hand, the same output of the U-net is fed to the adversarial network by reversing the gradient which also predicts the class labels. As no secret unbinding is performed in the adversarial network, it will make the classification harder if someone tries to classify labels using the output of the U-net without doing secret unbinding.

The input and output dimension of each of the network solely depends on the dataset. The size of the prediction and adversarial network also depends on the dataset and the number of parameters increases as the dimension of the images increases. For instance, CIFAR-10 dataset has RGB images of dimension $(32 \times 32 \times 3)$ in 10 different classes. Therefore, the input and output dimension of the U-net is $(32 \times 32 \times 3)$. As for the prediction and adversarial network, three blocks of Convolutional and Maxpooling layer are used back-to-back with the number of filters 32, 64, 128 in each block. The output dimension of the final block is $(4 \times 4 \times 128)$ which is flattened to a vector of length 2048. Next, three FC layers are used back-to-back, and the dimension of the vector is reduced from 2048 to 1024, 512, and finally, 10 which is the number of classes in the CIFAR-10 dataset.

The network is optimized using Adam optimizer with a learning rate of 10^{-3} which is subsequently reduced by 10 fold after 200 epochs for better convergence. The batch size is chosen is to be 64 and each of the networks is trained until convergence up to 400 epochs.

Below we show code snippets of each network's definition for a complete specification of its design, layers, activation, and neurons. Complete code is provided in the supplemental material.

```
def forward(self, x, key):
    x_main = self.F_main(x)  #r = f_W(x ⊛ s)
    y_attack = self.F_attact(x_main)  #ŷ_A = ReverseGrad(f_A(x))
    x_unbind = unbinding_2d(x_main, key)  #tmp = r ⊛ s†
    y_pred = self.F_pred(x_unbind)  #ŷ_P = f_P(tmp) = f_P(r ⊛ s†)
    return y_pred, y_attack, x_main
```

Figure 9: Forward function signature used by all of our networks.

```
class NetworkMiniImageNet(nn.Module):
    def __init__(self, activation=nn.LeakyReLU(0.1)):
        super().__init__()

        self.F_main = nn.Sequential(
            unet.UNet2D(3, 3)
        )

        self.F_attact = nn.Sequential(
            RevGrad(),
            nn.Conv2d(3, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(128, 256, (2, 2), padding=(1, 1)), nn.BatchNorm2d(256), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(256, 256, (3, 3), padding=(1, 1)), nn.BatchNorm2d(256), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(6400, 2048), activation, nn.BatchNorm1d(2048),
            nn.Dropout(p=0.5),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 100), nn.Softmax(dim=-1)
        )

        self.F_pred = nn.Sequential(
            nn.Conv2d(3, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(128, 256, (2, 2), padding=(1, 1)), nn.BatchNorm2d(256), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(256, 256, (3, 3), padding=(1, 1)), nn.BatchNorm2d(256), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(6400, 2048), activation, nn.BatchNorm1d(2048),
            nn.Dropout(p=0.5),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 100), nn.Softmax(dim=-1)
        )
```

Figure 10: Network for Mini-ImageNet results.

```
class NetworkCIFAR100(nn.Module):
    def __init__(self, activation=nn.LeakyReLU(0.1)):
        super().__init__()

        self.F_main = nn.Sequential(
            unet.UNet2D(3, 3)
        )

        self.F_attact = nn.Sequential(
            RevGrad(),
            nn.Conv2d(3, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(128, 256, (3, 3), padding=(1, 1)), nn.BatchNorm2d(256), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(4096, 2048), activation, nn.BatchNorm1d(2048),
            nn.Dropout(p=0.5),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 100), nn.Softmax(dim=-1)
        )

        self.F_pred = nn.Sequential(
            nn.Conv2d(3, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(128, 256, (3, 3), padding=(1, 1)), nn.BatchNorm2d(256), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(4096, 2048), activation, nn.BatchNorm1d(2048),
            nn.Dropout(p=0.5),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 100), nn.Softmax(dim=-1)
        )
```

Figure 11: Network for CIFAR-100 results.

```
class NetworkCIFAR10(nn.Module):
    def __init__(self, activation=nn.LeakyReLU(0.1)):
        super().__init__()

        self.F_main = nn.Sequential(
            unet.UNet2D(3, 3)
        )

        self.F_attact = nn.Sequential(
            RevGrad(),
            nn.Conv2d(3, 32, (3, 3), padding=(1, 1)), nn.BatchNorm2d(32), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(32, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 512), activation, nn.BatchNorm1d(512),
            nn.Dropout(p=0.5),
            nn.Linear(512, 10), nn.Softmax(dim=-1)
        )

        self.F_pred = nn.Sequential(
            nn.Conv2d(3, 32, (3, 3), padding=(1, 1)), nn.BatchNorm2d(32), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(32, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 512), activation, nn.BatchNorm1d(512),
            nn.Dropout(p=0.5),
            nn.Linear(512, 10), nn.Softmax(dim=-1)
        )
```

Figure 12: Network for CIFAR-10 results.

```
class NetworkSVHN(nn.Module):
    def __init__(self, activation=nn.LeakyReLU(0.1)):
        super().__init__()

        self.F_main = nn.Sequential(
            unet.UNet2D(3, 3)
        )

        self.F_attact = nn.Sequential(
            RevGrad(),
            nn.Conv2d(3, 32, (3, 3), padding=(1, 1)), nn.BatchNorm2d(32), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(32, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 512), activation, nn.BatchNorm1d(512),
            nn.Dropout(p=0.5),
            nn.Linear(512, 10), nn.Softmax(dim=-1)
        )

        self.F_pred = nn.Sequential(
            nn.Conv2d(3, 32, (3, 3), padding=(1, 1)), nn.BatchNorm2d(32), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(32, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(2048, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 512), activation, nn.BatchNorm1d(512),
            nn.Dropout(p=0.5),
            nn.Linear(512, 10), nn.Softmax(dim=-1)
        )
```

Figure 13: Network for SVHN results.

```
class NetworkMNIST(nn.Module):
    def __init__(self, activation=nn.LeakyReLU(0.1)):
        super().__init__()

        self.F_main = nn.Sequential(
            nn.Conv2d(1, 3, (1, 1)),
            unet.UNet2D(3, 3),
            nn.Conv2d(3, 1, (1, 1)),
        )

        self.F_attact = nn.Sequential(
            RevGrad(),
            nn.Conv2d(1, 32, (3, 3), padding=(1, 1)), nn.BatchNorm2d(32), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(32, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(1152, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 512), activation, nn.BatchNorm1d(512),
            nn.Dropout(p=0.5),
            nn.Linear(512, 10), nn.Softmax(dim=-1)
        )

        self.F_pred = nn.Sequential(
            nn.Conv2d(1, 32, (3, 3), padding=(1, 1)), nn.BatchNorm2d(32), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(32, 64, (3, 3), padding=(1, 1)), nn.BatchNorm2d(64), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Conv2d(64, 128, (3, 3), padding=(1, 1)), nn.BatchNorm2d(128), activation,
            nn.MaxPool2d((2, 2)), nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(1152, 1024), activation, nn.BatchNorm1d(1024),
            nn.Dropout(p=0.5),
            nn.Linear(1024, 512), activation, nn.BatchNorm1d(512),
            nn.Dropout(p=0.5),
            nn.Linear(512, 10), nn.Softmax(dim=-1)
        )
```

Figure 14: Network for MNIST results.

B. Dataset and Training Details

Additionally, during training, data is augmented by applying random horizontal flip (50%), random brightness $(0.7, 1.3)$, random contrast $(0.7, 1.3)$, random crop $(0.7, 1.3)$, random rotation $(-60, 60)$, random translation $(0.0, 0.25)$, random scaling $(0.9, 1.1)$, and Gaussian blur with standard deviation of $(0.5, 1.5)$. Therefore, the network will learn from a large dataset and be more likely to be regularized.

Table 7: Datasets Description

Dataset	Dimension	Classes	Training	Testing
MNIST	$28 \times 28 \times 1$	10	50000	10000
SVHN	$32 \times 32 \times 3$	10	73257	26032
CIFAR-10	$32 \times 32 \times 3$	10	50000	10000
CIFAR-100	$32 \times 32 \times 3$	100	50000	10000
Mini-ImageNet	$84 \times 84 \times 3$	100	50000	10000

C. Further Exposition on Averaged Predictions

Our results from Figure 4 show improved accuracy and have a subtle benefit. Under existing protocols discussed in section 2, performing k predictions would be k times as expensive. We would expect this to be $< k$ times as expensive in a real-world deployment, as it allows amortizing the communication overhead/latency once for k predictions. It is also well known that batched computations are more compute efficiently, so sending k predictions to perform instead of 1 would be less than k times as much compute on the party running the backbone network $f_W(\cdot)$.

The reader may rightfully wonder if this decreases the security of our method, as some amount of information is revealed if it is known that the k items all represent the same input with different secrets. We do not expect this to be the case to any meaningful degree, given our results in section 5 showing clustering being ineffective at the extraction of information and even still being robust to omnipotent adversaries that are aware of the training approach, dataset, and class labels.

Further, if this was a concern there are strategies that could be employed to further complicate life for the untrusted party. This includes sending additional random/fake prediction tasks, interleaving the k replicates with other prediction tasks and other possible strategies that hide this information. This gets into a game-theoretic exercise beyond the scope of our work and is of limited importance given that we explicitly do not aim for provable security.

D. Clustering Visual Results

Below are plots of the true class distribution, followed by the cluster's identified classes, for all datasets. This is most legible on MNIST (Figure 15), SVHN (Figure 16), and CIFAR-10 (Figure 17) due to the lower number of classes. They show clearly that the true class distribution appears random in the output space when the secret $\boldsymbol{s}$ is not available, and the clusters identified are latching onto a false manifold produced by the model. The results for CIFAR-100 (Figure 18) and Mini-ImageNet (Figure 19) are qualitatively the same, but hard to reader due to the issues in plotting 100 classes makes many different classes "look" similar due to the same coloring.

In Figure 20 we show all datasets UMAP plots of the true class labels after unbinding the secret $\boldsymbol{s}$, which shows how dramatically the output space is changed. This is clearest for MNIST, but we find an unusually consistent grouping for all other datasets. Each group has a mix of classes in it that can be separated with reasonable accuracy (as evidenced by our results), but the totality of the behavior is not yet fully understood. The most important result from this though is to demonstrate the significance of obfuscating impact that $\boldsymbol{s}$ has on the manifold of the data.

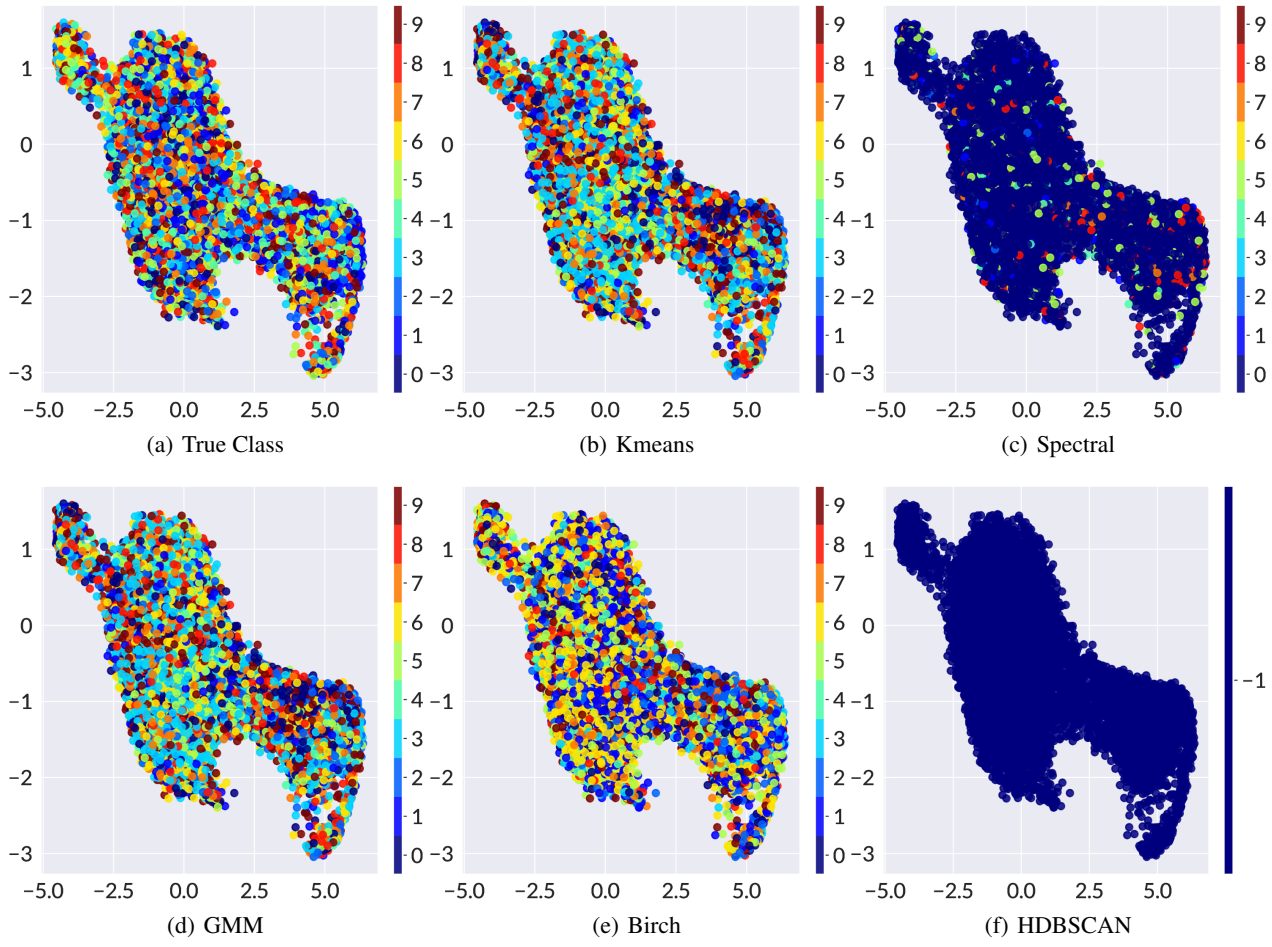
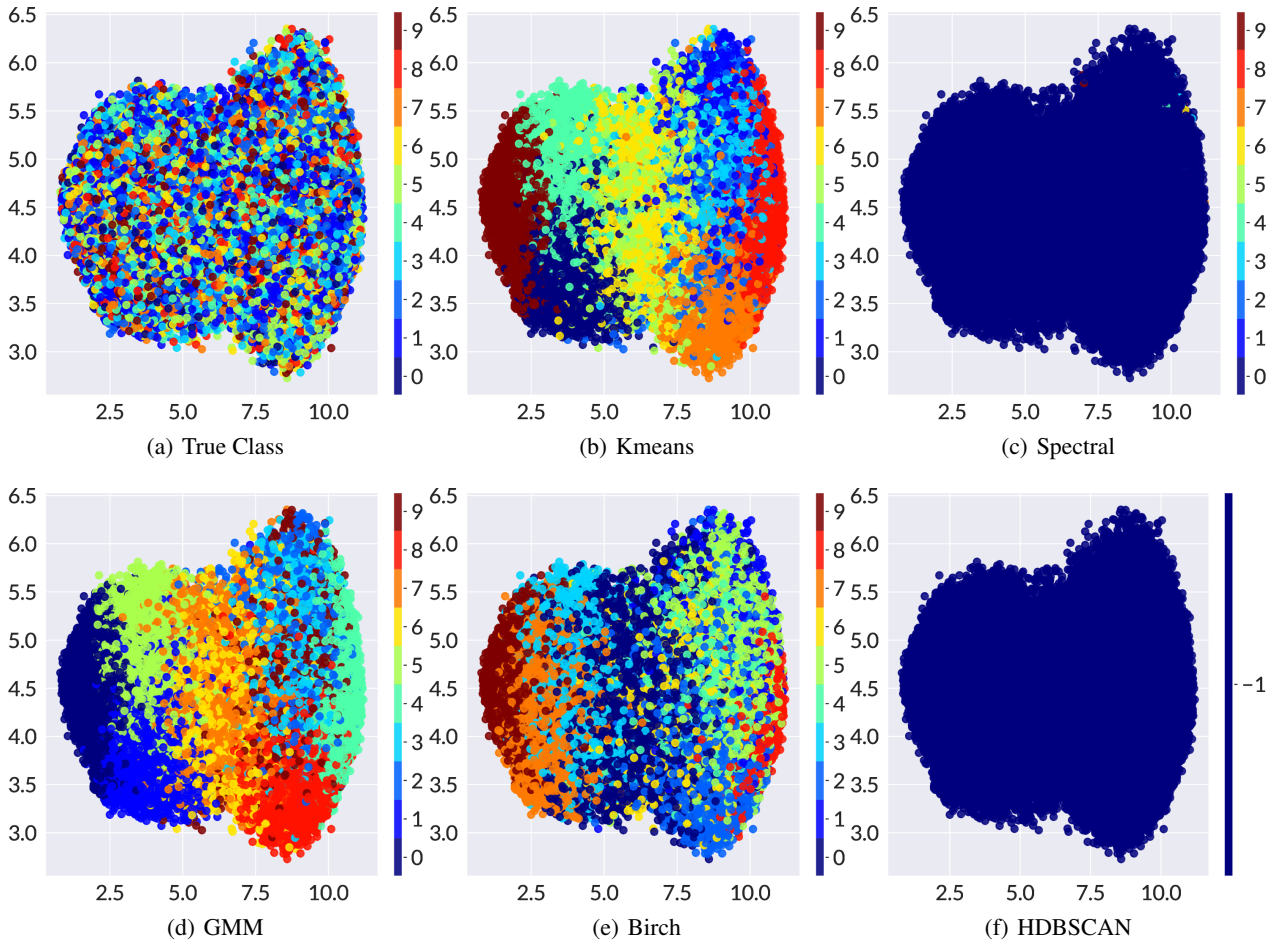


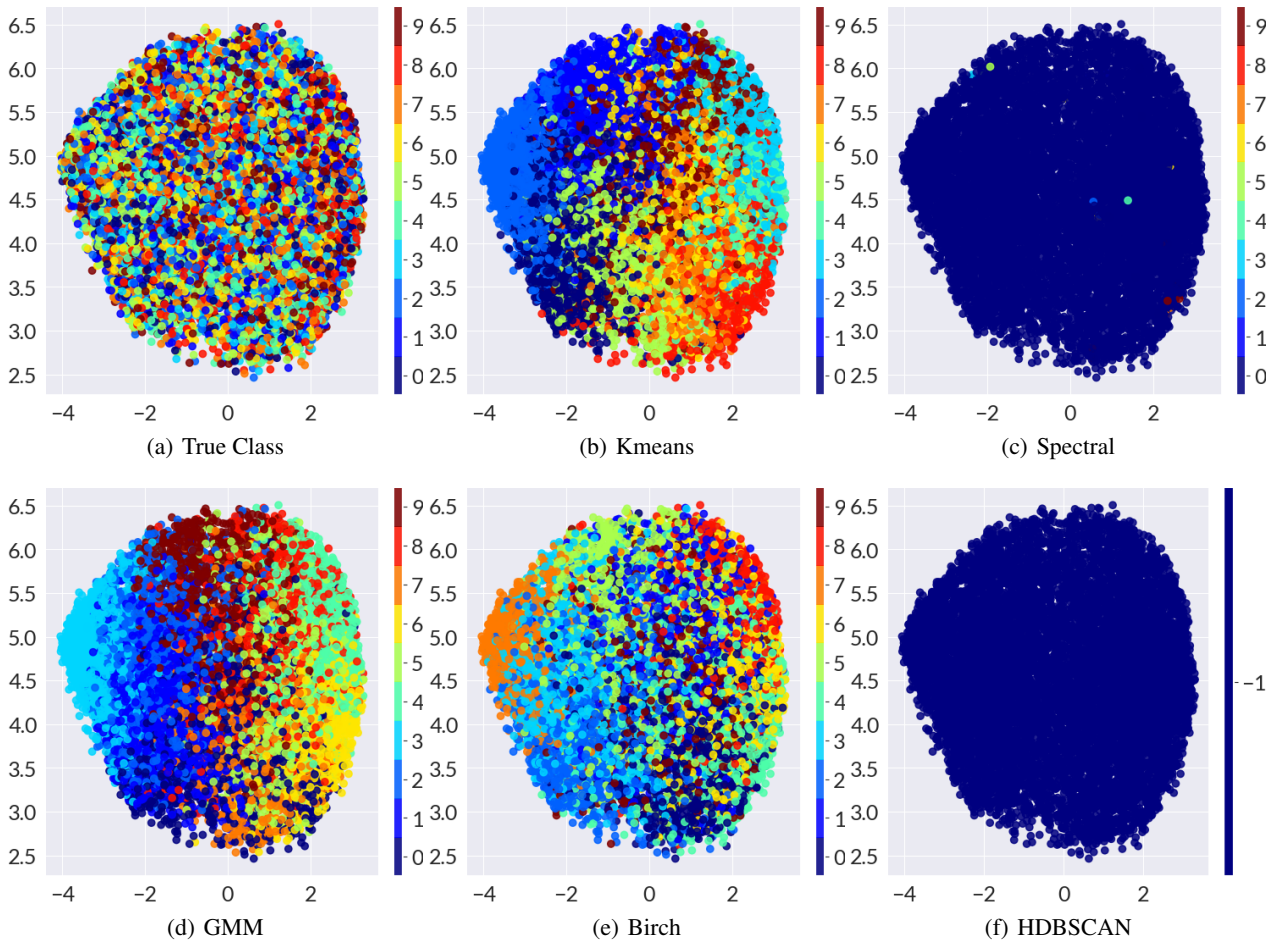
Figure 15: MNIST2 UMAP 2D Embedding representation of the clustering of the output of the U-Net.



(a) True Class (b) Kmeans (c) Spectral

(d) GMM (e) Birch (f) HDBSCAN

Figure 16: MNIST2 UMAP 2D Embedding representation of the clustering of the output of the U-Net.



(a) True Class (b) Kmeans (c) Spectral

(d) GMM (e) Birch (f) HDBSCAN

Figure 17: CIFAR-10 UMAP 2D Embedding representation of the clustering of the output of the U-Net.

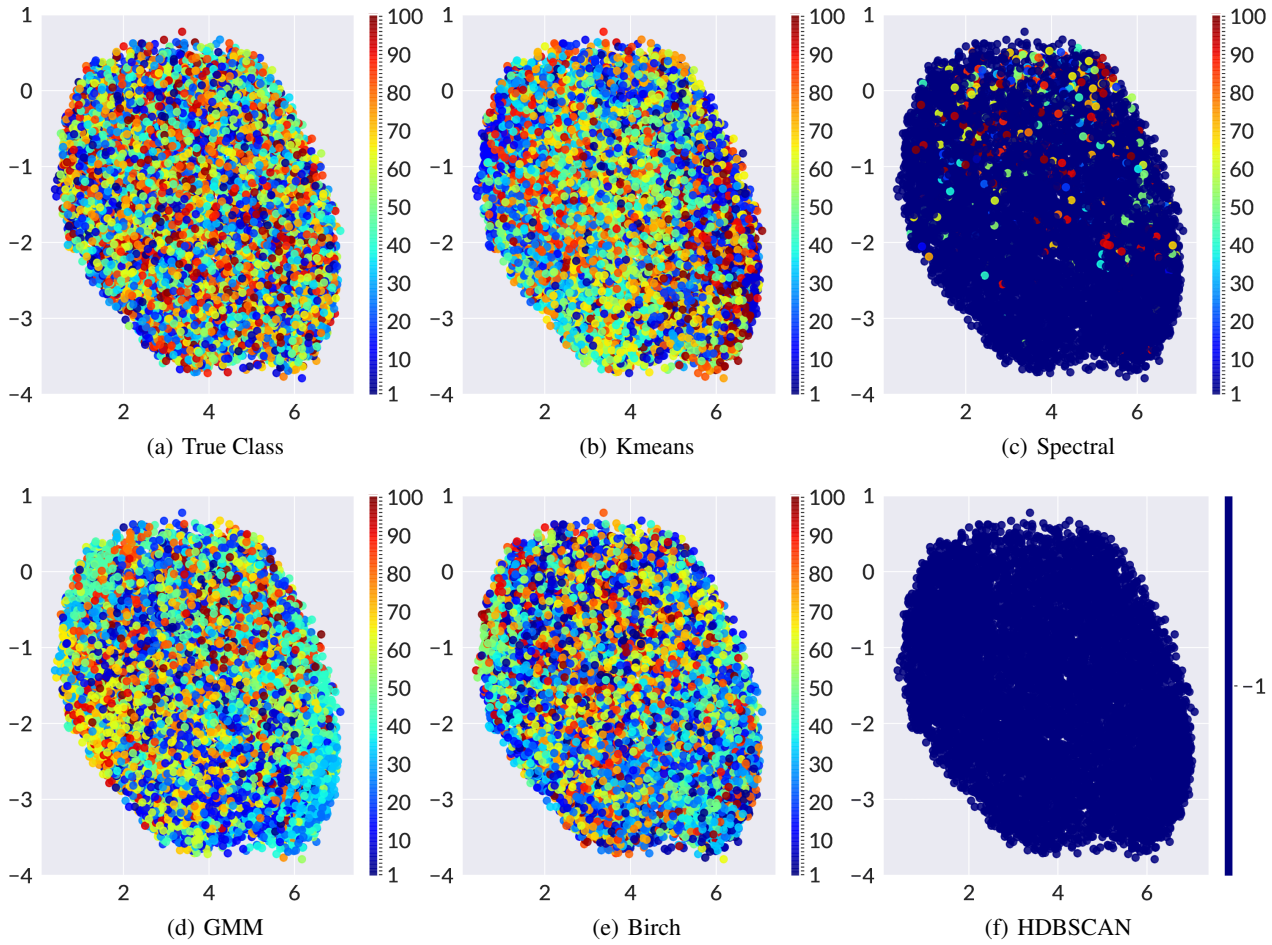


Figure 18: CIFAR-100 UMAP 2D Embedding representation of the clustering of the output of the U-Net.

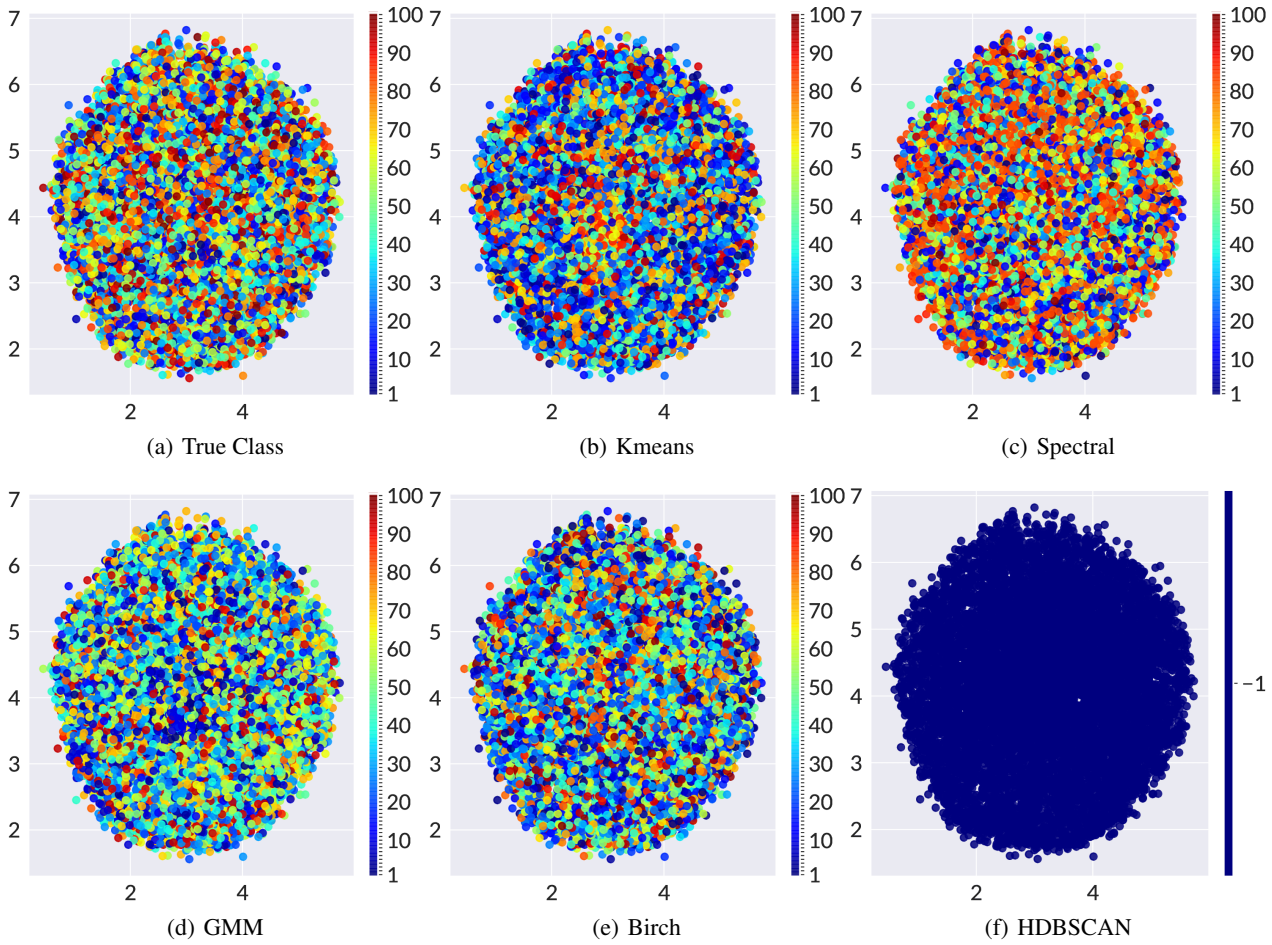
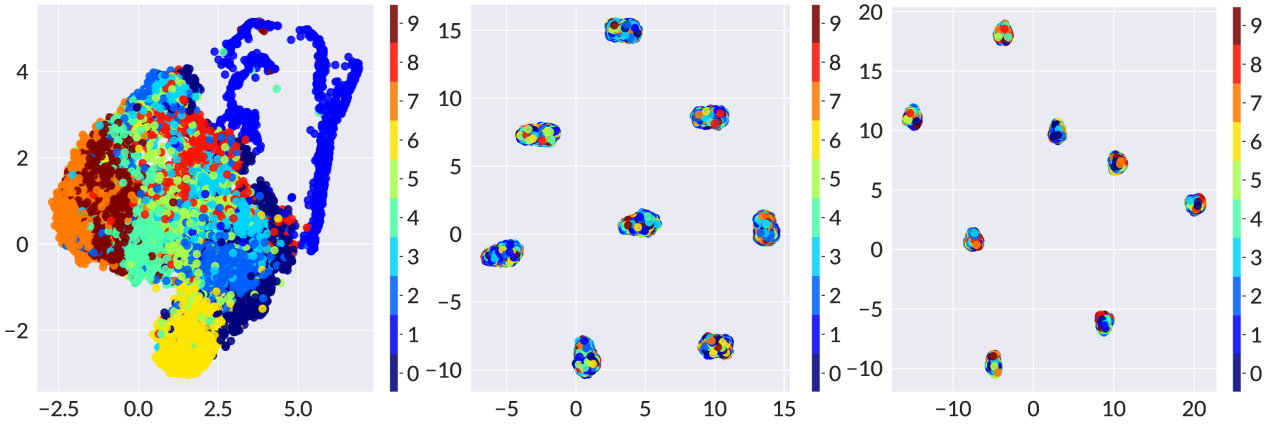
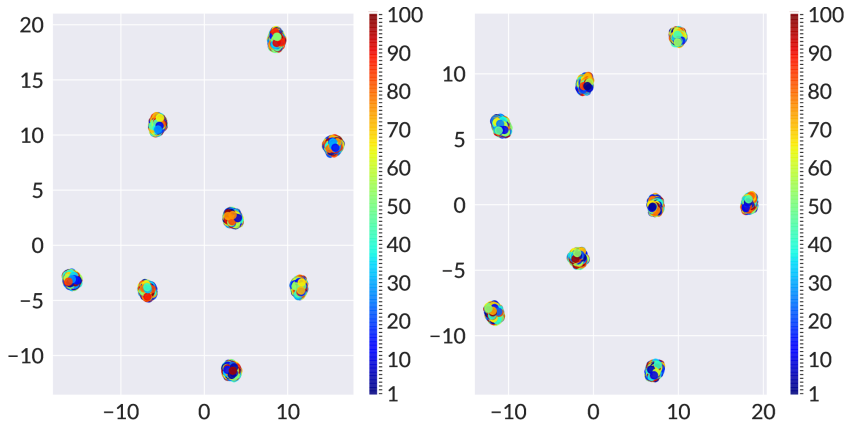


Figure 19: Mini-ImageNet UMAP 2D Embedding representation of the clustering of the output of the U-Net.



(a) MNIST (b) SVHN (c) CIFAR-10



(d) CIFAR-100 (e) Mini-ImageNet

Figure 20: UMAP embeddings of output $r \circledast s^{\dagger}$, showing what the space looks like once the secret is applied to extract the output. We can see on MNIST that the classes are clustered well but grouped together. This appears to be a large part of what happens to the sub-cluster that occurs on other datasets, where the secret is consistently placing the populations into one of eight groups. The consistency of the eight populations is not yet known but does show how the binding/unbinding dramatically changes the characteristics of the space.

D.1. Ablation Study

We have shown our CSPS is 290-5000× faster than alternative options, empirically robust to subversion against unreasonable powerful adversaries, and maintains high predictive accuracy. We now perform several ablations that demonstrate all the components to our design are needed to obtain our strong results. We will perform ablation tests using ResNet-50 (He et al., 2016) and ReZero (Bachlechner et al., 2020) architectures instead of U-Net, where the secret $\boldsymbol{s}$ is projected down to match the smaller output dimension, we will show the importance of maintaining input/output size of $f_W(\cdot)$. Alternative VSA options including standard HRR, vector-derived transformation binding (VTB) (Gosmann & Eliasmith, 2019), and our own improved VTB (iVTB) demonstrate the importance of our 2D HRR approach to maintaining the symbolic properties in a manner that can be extracted after the CNN. We also test alternative encoding of the 2D structure by using a space-filling Hilbert curve with 1D HRR as an approximation of the total 2D structure. We briefly summarize how each ablation type is performed. Visual examples of how HRR, VTB, iVTB, and Hilbert impact the encoding/decoding are in **??**.

ReZero/ResNet: A ReZero/ResNet architecture with 50 layers is used as $f_W(\cdot)$, resulting in an output size smaller than the input. A two layer fully-connected network is used to project $\boldsymbol{s}$ down to the output shape of the network.

HRR: The standard HRR with a 1D FFT is used and initialized using (Ganesan et al., 2021).

VTB: The VTB is used instead of HRR, as described by (Gosmann & Eliasmith, 2019). VTB replaces the FFT operation with a tiled sparse block-diagonal matrix multiplication that has similar properties as HRR.

iVTB: VTB, but improved by choosing the secrets to be block orthogonal to force properties of the VTB to be exactly true, rather than true in expectation. First, a tensor is randomly sampled from a normal distribution and then QR decomposition is applied in the tensor. The orthogonal part of the QR decomposition is used as the secret $\boldsymbol{s}$, which improves binding retrieval.

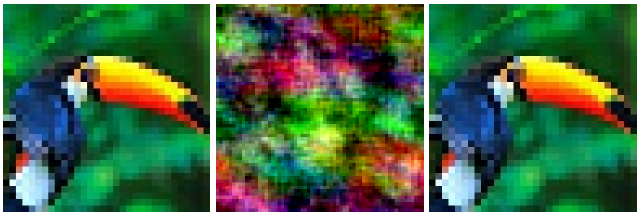
Hilbert Space-Filling Curve: To keep the spatial locality of the image and restore the structural conformity Hilbert Space-Filling Curve is employed. Images are encoded and decoded before and after binding and unbinding.

Table 8: Ablation Study on CIFAR-10.

Method	Accuracy
U-net + HRR 2D (CSPS)	78.21
ResNet-50 + HRR	53.52
ReZero-50 + HRR	57.80
ReZero-50 + VTB	52.21
ReZero-50 + iVTB	54.19
ReZero-50 + HRR + Hilbert	53.81
ResNet-50 + HRR + Hilbert	50.87
ReZero-50 + HRR 2D	49.32

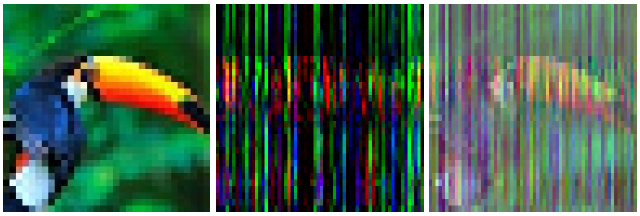
Table 8 shows the accuracy of the different combinations of the network architectures and VSAs. Among them, HRR with ReZero blocks in the main network has the best accuracy of 57.80%, significantly below CSPS. This shows the U-Net design, with 2D HRR for encoding, is critical in combination to obtain our results.

We further show for each of the proposed alternative strategies visualizations of the binding/unbinding process to provide intuition as to how they work or why their results are less effective.



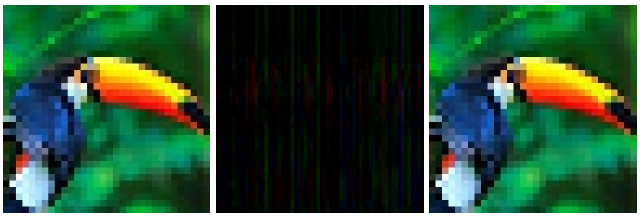
(a) Original Image (b) Bound Image (c) Retrieved Image

Figure 21: A standard HRR applied with the improved initialization of Ganesan et al. (2021), which does not recognize the 2D structure of the data but is implicitly a 1D convolution over the linearization of the pixels. The bound image thus looks random in a different way, but the output is retrievable.



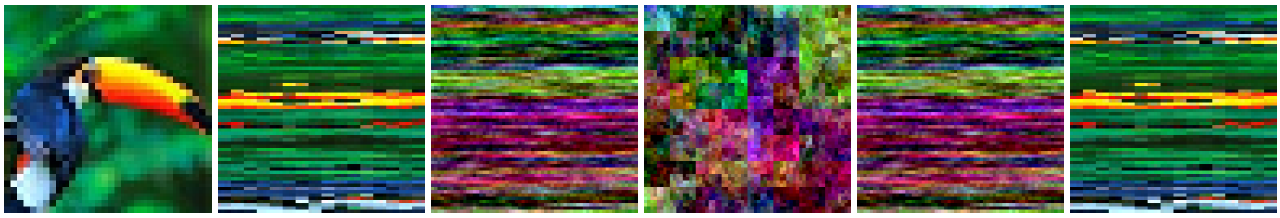
(a) Original Image (b) Bound Image (c) Retrieved Image

Figure 22: The VTB approach proposed by Gosmann & Eliasmith (2019) is applied to the flattened/linearized version of the pixels. While the input is still obfuscated in a different visual pattern, the retrieved image is very noisy. Such noise is inevitable when multiple items are bound together, but given that we have only one item we would desire a higher quality retrieval.



(a) Original Image (b) Bound Image (c) Retrieved Image

Figure 23: Our improved VTB forces the sub-structure of the vector to be orthogonal, allowing for exact retrieval of the input if there is only one bound item.



(a) Original Image (b) Hilbert Encode (c) Bound Image (d) Hilbert Decode (e) Hilbert Encode (f) Unbound Image

(g) Hilbert Decode

Figure 24: Hilbert Transform combined with the standard HRR. The intuition is that the standard HRR is a 1D convolution, and so if we Hilbert encode the input we linearize the pixels in a manner that is retaining much of the 2D spatial locality in a 1D space. This allows successful obfuscation and extraction but did not perform as well.