A DEFINITION OF KERNELS

In this appendix we provide the definition of each of the kernels functions evaluated. For convenience we restate the definition of the trNTK.

trNTK Recall the definition of the total gradient with respect to $\boldsymbol{\theta}$ at datapoint $\boldsymbol{x}_i$ by

$$\boldsymbol{g}(\boldsymbol{x}_i;\boldsymbol{\theta})^c = \nabla_{\boldsymbol{\theta}} F^c(\boldsymbol{x}_i;\boldsymbol{\theta}).$$

Then the trNTK evaluated at datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is given by

$$\mathrm{trNTK}(\boldsymbol{x_i},\boldsymbol{x_j}) = \frac{\sum_{c=1}^{C} \langle \boldsymbol{g}^c(\boldsymbol{x_i};\boldsymbol{\theta}), \boldsymbol{g}^c(\boldsymbol{x_j};\boldsymbol{\theta})\rangle}{\sum_{c=1}^{C} \langle \boldsymbol{g}^c(\boldsymbol{x_i};\boldsymbol{\theta}), \boldsymbol{g}^c(\boldsymbol{x_i};\boldsymbol{\theta})\rangle^{\frac{1}{2}} \sum_{c=1}^{C} \langle \boldsymbol{g}^c(\boldsymbol{x_j};\boldsymbol{\theta}), \boldsymbol{g}^c(\boldsymbol{x_j};\boldsymbol{\theta})\rangle^{\frac{1}{2}}}.$$

We provide additional details about the exact calculation in Appendix C.

Projection Trace Neural Tangent Kernel. We restate our definition of the proj-trNTK kernel function. We remind the reader that $\boldsymbol{P}$ is a Rademacher or Gaussian random projection matrix $\in \mathbb{R}^{K\times P}$, with K a hyperparameter, P the number of model parameters, and K chosen to be $K << P$. In all experiments K = 10240.

$$\text{proj-trNTK}(\boldsymbol{x}_i,\boldsymbol{x}_j) := \frac{\sum_{c=1}^{C} \langle \boldsymbol{P}\boldsymbol{g}^c(\boldsymbol{x}_i;\boldsymbol{\theta}), \boldsymbol{P}\boldsymbol{g}^c(\boldsymbol{x}_j;\boldsymbol{\theta})\rangle}{\sum_{c=1}^{C} \langle \boldsymbol{P}\boldsymbol{g}^c(\boldsymbol{x}_i;\boldsymbol{\theta}), \boldsymbol{P}\boldsymbol{g}^c(\boldsymbol{x}_i;\boldsymbol{\theta})\rangle^{\frac{1}{2}} \sum_{c=1}^{C} \langle \boldsymbol{P}\boldsymbol{g}^c(\boldsymbol{x}_j;\boldsymbol{\theta}), \boldsymbol{P}\boldsymbol{g}^c(\boldsymbol{x}_j;\boldsymbol{\theta})\rangle^{\frac{1}{2}}}$$

Projection Psuedo Neural Tangent Kernel.

$$\text{proj-pNTK}(\boldsymbol{x}_i,\boldsymbol{x}_j) := \frac{\langle \boldsymbol{P}\sum_{c=1}^{C} \boldsymbol{g}^c(\boldsymbol{x}_i,\ \boldsymbol{\theta}), \boldsymbol{P}\sum_{c=1}^{C} \boldsymbol{g}^c(\boldsymbol{x}_j,\ \boldsymbol{\theta})\rangle}{||\boldsymbol{P}\sum_{c=1}^{C} \boldsymbol{g}^c(\boldsymbol{x}_i,\ \boldsymbol{\theta})|| \cdot ||\boldsymbol{P}\sum_{c=1}^{C} \boldsymbol{g}^c(\boldsymbol{x}_j,\ \boldsymbol{\theta})||}$$

Embedding Akyürek et al. (2022) defines the embedding kernel, which we restate here. The embedding kernel is computed from the correlation of the activations following each layer. Let $\lambda_\ell(\boldsymbol{x}\,;\boldsymbol{\theta})$ be the output of the ℓ-th hidden layer of $F(\boldsymbol{x}\,;\boldsymbol{\theta})$. We denote the ℓ-th embedding kernel at datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ by

$$E_\ell(\boldsymbol{x}_i,\boldsymbol{x}_j) = \frac{\langle \lambda_l(\boldsymbol{x}_i\,;\boldsymbol{\theta}), \lambda_l(\boldsymbol{x}_j\,;\boldsymbol{\theta})\rangle}{\|\lambda_l(\boldsymbol{x}_i\,;\boldsymbol{\theta})\|\|\lambda_l(\boldsymbol{x}_j\,;\boldsymbol{\theta})\|}.$$

Let the full embedding kernel be defined by the normalized sum over the unnormalized embedding kernel at each layer of the NN

$$E(\boldsymbol{x}_i,\boldsymbol{x}_j) = \frac{\sum_{\ell=1}^{L} \langle \lambda_\ell(\boldsymbol{x}_i\,;\boldsymbol{\theta}), \lambda_\ell(\boldsymbol{x}_j\,;\boldsymbol{\theta})\rangle}{\sqrt{\sum_{\ell=1}^{L} \|\lambda_\ell(\boldsymbol{x}_i\,;\boldsymbol{\theta})\|^2\, \|\lambda_\ell(\boldsymbol{x}_j\,;\boldsymbol{\theta})\|^2}}.$$

Embedding kernels are an interesting comparison for the data attribution task when we consider the prominent role they play in transfer learning and auto-encoding paradigms. In both, finding an embedding that can be utilized in down-stream tasks is the objective.

Conjugate Kernel We utilize an the empirical conjugate kernel (CK) to compare to the trNTK. Let the normalized CK be defined by

$$\mathrm{CK}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \frac{\langle \lambda_L(\boldsymbol{x}_i\,;\boldsymbol{\theta}), \lambda_L(\boldsymbol{x}_j\,;\boldsymbol{\theta})\rangle}{\|\lambda_L(\boldsymbol{x}_i\,;\boldsymbol{\theta})\|\|\lambda_L(\boldsymbol{x}_j\,;\boldsymbol{\theta})\|}.$$

The CK is an interesting comparison for a couple of reasons: first, for any network that ends in a fully connected layer, the CK is actually an additive component of the trNTK; therefore, we can

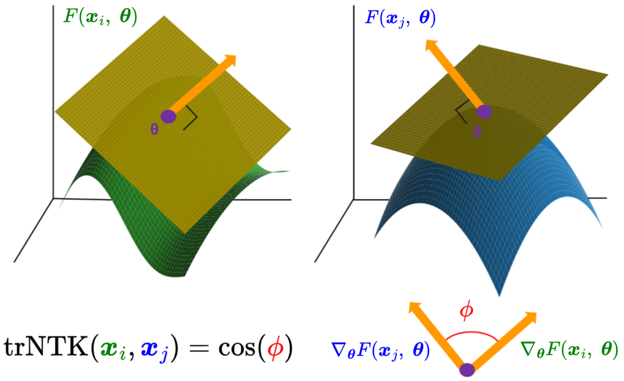

$F(\boldsymbol{x}_i,\ \boldsymbol{\theta})$
θ
$F(\boldsymbol{x}_j,\ \boldsymbol{\theta})$
θ
ϕ
$\text{trNTK}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \cos(\phi)$
$\nabla_{\boldsymbol{\theta}} F(\boldsymbol{x}_j,\ \boldsymbol{\theta})$
$\nabla_{\boldsymbol{\theta}} F(\boldsymbol{x}_i,\ \boldsymbol{\theta})$

Figure 3: **Geometric intuition behind the trNTK.** A NN function is evaluated at two points creating surfaces $F(\boldsymbol{x}_i\,;\boldsymbol{\theta})$ and $F(\boldsymbol{x}_j\,;\boldsymbol{\theta})$. These surfaces are shown with a tangent hyper plane at the same point ($\boldsymbol{\theta}$) in parameter space coinciding with the end of training. The Jacobian vector defines the tangent hyperplane's orientation in parameter space. The trNTK is a kernel whose (i, j)-th element is the cosine angle between averaged Jacobian vectors. The more similar the local geometry between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ local to $\boldsymbol{\theta}$ in parameter-space , the higher the value of $\text{trNTK}(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

evaluate whether a smaller amount of the total trNTK can accomplish the same task. Second, the CK is computed from the final feature vector before a network makes a decision; the NN is exactly a linear model with respect to this final feature vector. NN architectures typically contain bottlenecks that project down to this final feature vector. These projections remove information. While that information might be of no use to the classification task, it may be useful for the attribution task. We can think of the the final information presented to the NN as the CK, and the information contained before these projections as the trNTK, though more work is needed to formalize and explore this comparison.

Unnormalized Pseudo Neural Tangent Kernel To evaluate the effect of the normalization in the trNTK definition we will evaluate the kernel without normalizing. let the unnormalized trNTK be defined as:

$$\text{trNTK}^0(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{g}(\boldsymbol{x}_i; \boldsymbol{\theta})^\top \boldsymbol{g}(\boldsymbol{x}_j; \boldsymbol{\theta}).$$

While neural tangent kernels are not typically cosine-normalized kernels we were drawn to investigate such normalized kernels for a few reasons: Akyürek et al. (2022) remarked that cosine normalization could prevent training data with large magnitude Jacobian vectors from dominating the kernel, and Hanawa et al. (2021) notes a cosine-similarity kernel achieves the best performance among alternative kernels on a data attribution task. Key motivators for our study included that the cosine normalized values are intuitive geometrically, and that it is standard practice to ensure feature matrices such as $\boldsymbol{\kappa}$ are in a small range (such as [-1,1]) for machine learning.

B GEOMETRIC INTUITION BEHIND NEURAL TANGENT KERNELS

In figure 3 we provide a pictorial representation of the geometric interpretation behind the trNTK.

C ADDITIONAL DETAILS REGARDING THE TRACE NEURAL TANGENT KERNEL

In this appendix we provide an expanded definition of the trNTK that highlights how the trNTK is actually computed from a series of individual contributions from each learnable tensor. This layerwise decomposition has been pointed out in previous work (Novak et al., 2022). Let $\boldsymbol{\theta}_l$ be the parameter vector consisting of only the parameters from the l-th layer. Let the number of parameters in the l-th layer be p_l. A Jacobian is a vector of first-order partial derivatives of the NN with respect to the parameters. We will specify each Jacobian through the c-th scalar function (equivalently, c-th output neuron) for the parameters in the l-th layer as:

$$\mathrm{g}_l^c(\boldsymbol{x}_i) = \underbrace{\frac{\partial F(\boldsymbol{x}_i\,;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_l}}_{\in \mathbb{R}^{1\times P_l}}. \tag{7}$$

Note that we have intentionally broken our notation for the vector by using the Gothic capital g for the Jacobian vector. We do this to avoid confusion with the lowercase j used as an index. Let $\mathrm{g}_l(\boldsymbol{x}_i)$ be the concatenation of all such $\mathrm{g}_l^c(\boldsymbol{x}_i)$ for all $c \in \{1, 2, \ldots, C\}$:

$$\mathrm{g}_l(\boldsymbol{x}_i) = \underbrace{\left[\mathrm{g}_l^1(\boldsymbol{x}_i), \mathrm{g}_l^2(\boldsymbol{x}_i), \ldots, \mathrm{g}_l^C(\boldsymbol{x}_i)\right]}_{\in \mathbb{R}^{1\times CP_l}}. \tag{8}$$

Let $\boldsymbol{J}_l(\boldsymbol{X})$ be the matrix formed from column vectors $\mathrm{g}_l(\boldsymbol{x}_i)^\top$ over each training data point $\boldsymbol{x}_i$, where $i \in \{1, 2, \ldots, N\}$:

$$\boldsymbol{G}_l(\boldsymbol{X}) = \underbrace{\left[\mathrm{g}_l(\boldsymbol{x}_1)^\top, \mathrm{g}_l(\boldsymbol{x}_2)^\top, \ldots, \mathrm{g}_l(\boldsymbol{x}_N)^\top\right]}_{\in \mathbb{R}^{CP_l\times N}}. \tag{9}$$

Let the l-th unnormalized pseudo-Neural Tangent Kernel, or trNTK_l, be the Gram matrix formed from the products of $\boldsymbol{J}_l(\boldsymbol{X})$ matrices:

$$\mathrm{trNTK}_l^0 = \underbrace{\boldsymbol{G}_l(\boldsymbol{X})^\top \boldsymbol{G}_l(\boldsymbol{X})}_{\in \mathbb{R}^{N\times N}}. \tag{10}$$

As a Gram matrix, trNTK_l^0 is symmetric and positive semi-definite. Let $\mathrm{trNTK}^0 \in \mathbb{R}^{N\times N}$ be the matrix formed from summing the contributions from all trNTK_l^0.

$$\mathrm{trNTK}^0 = \underbrace{(\sum_{l=1}^{L} \mathrm{trNTK}_l^0)}_{\in \mathbb{R}^{N\times N}} \tag{11}$$

The trNTK^0 itself is symmetric, as the sum of symmetric matrices is symmetric. Finally, we must apply the normalization. Let the matrix B be defined as the element-wise product of the trNTK with the identity:

$$\boldsymbol{B} = \boldsymbol{I} \odot \mathrm{trNTK}^0. \tag{12}$$

Then the normalized trNTK can be computed form the unnormalized trNTK by the following relationship:

$$\mathrm{trNTK} = \boldsymbol{B}^{\frac{-1}{2}}\,\mathrm{trNTK}^0\,\boldsymbol{B}^{\frac{-1}{2}} \tag{13}$$

The relationship between the full neural tangent kernel and the trNTK is described in Appendix D.

D RELATIONSHIP TO THE EMPIRICAL NTK

To calculate the full eNTK, first find the c-th class Jacobian vector, g^c, with respect to $\boldsymbol{\theta}$ backwards through the network for each $\boldsymbol{x}_i$ in the data matrix $\boldsymbol{X}$. Explicitly, the c-th logit's Jacobian i-th column-vector corresponds to datapoint $\boldsymbol{x}_i$ and is defined:

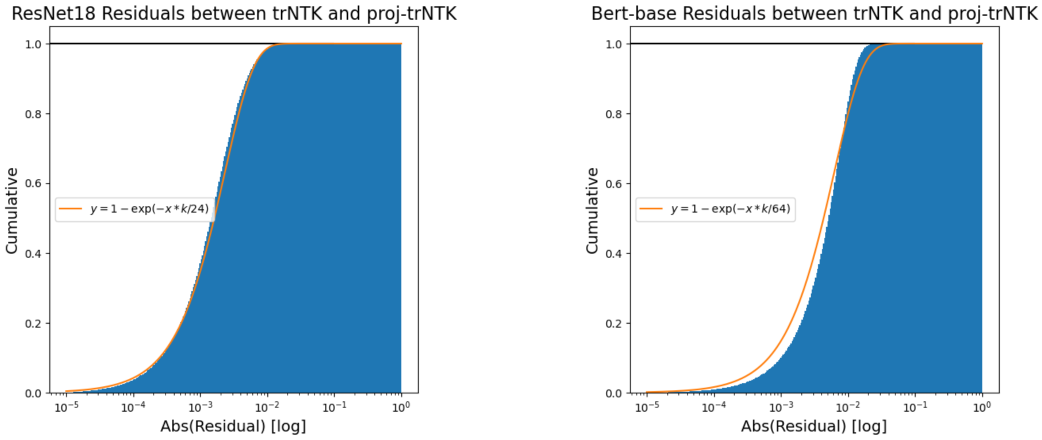
$$\mathrm{g}^c(\boldsymbol{x}_i) = \frac{\partial F^c(\boldsymbol{x}_i, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{14}$$

From which we can define the Jacobian matrix as:

$$\boldsymbol{G}^c = [\mathrm{g}^c(x_0),\ \mathrm{g}^c(x_1),\ \ldots,\ \mathrm{g}^c(x_N)] \tag{15}$$

The eNTK is the block-matrix whose (k,j)-th block, where both $k, j = \{1, 2, \ldots, C\}$, is the linear kernel formed between the Jacobians of the (k,j)-th logits.

$$\mathrm{NTK}_{k,j} = (\boldsymbol{G}^j)^\top(\boldsymbol{G}^k) \tag{16}$$



(a) (b)

Figure 4: **trNTK and proj-trNTK cosine-similarity residuals fall exponentially.** For both ResNet18 (4a) and Bert-base (4b) we plot the cumulative histogram of residuals between the trNTK and proj-trNTK. The orange line is an exponential function with k=10240. The orange line is fit "by eye" rather than some best-fit, the objective being to reference the exponential shape of the residual distribution.

The NTK is therefore a matrix $\in \mathbb{R}^{CN \times CN}$. The relationship between the unnormalized trNTK and the NTK is simply:

$$\mathrm{trNTK}^0 = \sum_{c=1}^{C} \mathrm{NTK}_{c,c} \tag{17}$$

We chose to study the trNTK instead of the NTK for simplicity, computational efficiency, and reduced memory footprint. Follow on work could attempt to use the entire NTK to form the surrogate models. We were additionally motivated by the approach taken in Chen et al. (2021) and Chen et al. (2022), and we refer the reader to Mohamadi and Sutherland (2022) for a deeper discussion of the qualities of similar approximations.

E NOTES ON THE PROJECTED VARIANTS OF THE NTK

TRAK Park et al. (2023) utilizes the Johnson Lindenstrauss lemma (Johnson and Lindenstrauss, 1984) to justify the use of the projection matrix K. The Johnson Lindenstrauss lemma bounds the error between any two vectors and the same two vectors projected under a projection matrix $\boldsymbol{P}$. The lemma can be used to show a bound on the cosine similarity between two vectors and two projected vectors (Lin et al., 2019). However, this bound relates the probability of the residual for all vectors being less than some small ϵ. From an applied perspective we might care only that the residuals of cosine similarity are small with high probability. We empirically observe that the absolute residuals of of the trace-NTK and proj-trNTK fall away as $\exp(-x * \beta)$, where β is the decay rate. In figures 4a and 4b, we show the residuals for our ResNet18 and Bert-base experiments, with an overlaid exponential decay model for reference. We are unaware of a formal proof that would dictate the form of the distribution of residuals, but we use these plots to empirically justify the exploration of the projected-variants as close approximations for the original kernels with large enough K. Intuitively, we expect that there is a trade-off between size of the dataset, size of the model, and K.

F FORMAL DEFINITION OF EVALUATION METRICS

In this appendix we restate all the metrics used throughout this study.

Kendall-τ rank correlation

For a paired sequence $S_\tau = \{(a_1, b_1), \ldots, (a_N, b_N)\}$ a pair (a_i, b_i) and (a_j, b_j) with $i \neq j$ are concordant if either both $a_i > a_j$ and $b_i > b_j$ or $a_i < a_j$ and $b_i < b_j$. Otherwise, the pair is discordant. We count the total number of concordant, NC, and number of discordant pairs, ND. Then, τ_K is defined as:

$$\tau(S_\tau) = \frac{(\mathrm{NC} - \mathrm{ND})}{\mathrm{NC} + \mathrm{ND}}$$

Test accuracy differential (TAD) We track the test accuracy differential, or TAD, given by the difference between the kGLM and NN's test accuracy,

$$\mathrm{TAD} = \mathrm{TestAcc}_{\mathrm{kGLM}} - \mathrm{TestAcc}_{\mathrm{NN}}, \tag{18}$$

to demonstrate that kGLM have similar performance to the underlying NN. A value of 0 is preferred.

Precision and Recall To evaluate whether our attributions are performant at discriminating between perturbed and unperturbed test datapoints, we use precision as a measure of how valid the flags given by our attribution model are, and recall as a measure of how complete these attributions were at identifying poisoned test data. A perfect model would have both precision and recall $= 1$. Precision and recall are defined:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FP})}$$
$$\mathrm{Recall} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})},$$

where TP is the true positive rate, FP is the false positive rate, and FN is the false negative rate.

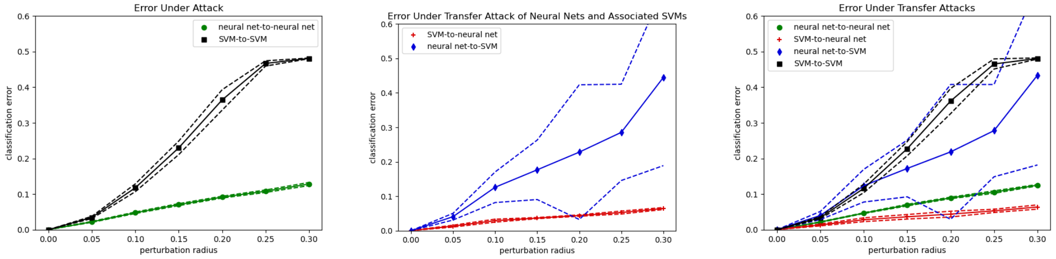
Coefficient of Determination $\mathbf{R^2}$ The coefficient of determination is used as a goodness-of-fit to assess the viability of our linearization of the NN (described below in Appendix I). It is possible to have a high τ_K but small R^2 if the choice of invertible mapping function is wrong or if the fit of said function does not converge. Such cases can be inspected visually to determine the relationship between the logits.

For a sequence of observations (in the context of this paper, the natural logarithm of probability of the correct class for the NN and kGLM) $S_{R^2} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, let the sample average of the y_i observations be $\bar{y} = \frac{1}{N}\sum_i^N y_i$. Then let the total sum of squares be $SS_{\mathrm{tot}} = \sum_i^N (y_i - \bar{y})^2$, and the sum of squared residuals be $SS_{\mathrm{res}} = \sum_i^N (y_i - x_i)^2$. Then let the goodness-of-fit R^2 function be defined:

$$R^2(S_{R^2}) = 1 - \frac{SS_{\mathrm{ret}}}{SS_{\mathrm{tot}}}$$

G ADVERSARIAL ATTACKS

We trained NN models on the MNIST dataset. In order to avoid combinatorial considerations, the classifier was trained on just two classes– we used 7's and 1's because these digits look similar. Subsequently, we extracted the NTKs and used these kernels to train SVMs. To attack both types of models, we considered ℓ_∞ perturbations, computed using the projective gradient descent algorithm (Madry et al., 2019) with 7 steps (PGD-7). Our experiments leverage PyTorch's auto-differentiation engine to compute second-order derivatives to effectively attack the SVMs. In contrast, prior work (Tsilivis and Kempe, 2023) derived an optimal one-step attack for the NTK at the limit and and used this approximation to compute adversarial examples. To compare neural nets with kernel regression, (Tsilivis and Kempe, 2023) compute the cosine similarity between the FGSM adversarial attack and the optimal 1-step attack for kernel machine, computed analytically by taking the limit for an infinitely wide neural net. Their results show (Figures 3 and 7 of (Tsilivis and Kempe, 2023)) that throughout training, the cosine similarity of this optimal 1-step attack and the empirical attack on the neural net decreases. This observation suggests that in practice, the NTK limit is not a good surrogate model for



(a) (b) (c)

Figure 5: **Error Under Adversarial Attacks:** (5a) White-box attack: Attacking a neural net and the associated NTK SVM directly. (5b) White-box attack: Attacking a neural net using perturbed examples for the associated SVM and attacking an NTK SVM by using perturbed examples for the associated neural net. (5c) Black-box attack: Attacking neural nets and SVMs using perturbed examples from independently trained SVMs and neural nets. This demonstrates a limitation of our surrogate model method: we find that the SVM's performance does not scale the same as the NN's performance with increasing perturbation radius, across multiple kinds of attack.

a neural net under an adversarial attack. Our plots (Figure 5) confirm this observation as SVMs are much more vulnerable to attacks that the associated neural nets. To better compare with prior work, we trained our SVMs using NTKs rather than pNTKs.

In considering security of neural nets, attacks are categorized as either *white-box* or *black-box*. White-box attacks assume that the adversary has access to all the weights of a neural net while black box attacks do not assume that an adversary has this information. A common strategy for creating a black box attack is training an independent NN and then using perturbations calculated from attacking this new NN to attack the model in question. Such attacks are called *transfer attacks*; see (Papernot et al., 2016b;a) for examples of successful black-box and transfer attacks.

In line with this framework, we test our models against two white-box attacks and a black box attack. First, we test neural nets and SVMs by directly attacking the models. Next, to better understand the similarities between a neural net and the associated SVM, we evaluate the SVM on attacks generated from the associated neural net and the neural net on attacks generated from the associated SVM. For the black box attacks, we test: 1) neural nets on adversarial examples generated from independently trained neural nets, 2) SVMs on adversarial examples from SVMs trained with an NTK from an independently trained neural net, 3) Neural nets on adversarial examples from SVMs trained with an NTK from and independently trained neural net, 4) SVMs on adversarial examples from independently trained neural nets.

The error bars for all three figures are on 10 trials. For the black box figure, each model was tested against 9 other independently trained models; the plotted quantities are the average of all these black box attacks.

G.1 ADVERSARIAL EXPERIMENT DETAILS

When performing PGD to find adversarial examples to our models, we did not restrict pixel values of the attacked images to the interval $[0, 1]$. See (Madry et al., 2019) for more information on using the PGD algorithm in an adversarial context. Notice that in the PGD algorithm, attacking the SVM trained with the NTK involves computing second derivatives of the neural net. Due to this consideration, using ReLUs as neurons in this experiment was impractical– the second derivative of a piecewise linear function at any point is either zero or non-existent. Hence the nets are constructed from sigmoid neurons.

The model architecture was 3 fully connected layers of 100 neurons. The models were trained for 100 epochs and with learning rate 10^{-4} with AdamW optimizer and minibatches of size 64 in PyTorch on the cross-entropy loss. The error bars in both Figures 5a and 5b figures are computed from the standard deviation calculated from 10 independent experimental trials set with different random seeds.

The SVMs were trained using `sklearn`'s SVM package.

H ADDITIONAL EXPERIMENTAL DETAILS

In this appendix we detail the specific choice of architecture, hyperparameters, and training times for each experiment.

H.1 DATASETS

Our experiments utilize common benchmark datasets: MNIST, FMNIST, CIFAR10, and COLA. we will quickly introduce each in turn. The Modified National Institute of Science Technology (MNIST) (Lecun et al., 1998) handwritten digit database is a grey-scale image classification dataset comprised of handwritten numerical digits and label pairs created from combining the National Institute of Science and Technology special datasets 1 and 3. MNIST has over 50,000 training and 10,000 testing data-label pairs. "Fashion"-MNIST (FMNIST) (Xiao et al., 2017) is another image classification dataset that was specifically introduced to serve as drop in replacement to MNIST. It was created by reducing images from an online European fashion catalogue to the same 28x28 pixel resolution as MNIST and to grey-scale. FMNIST has 10 classes of different kinds of garments, with 7,000 examples of each garment, split into 60,000 training and 10,000 test data. Canadian Institute for Advanced Research-10 (CIFAR10) is a 10-class supervised image classification dataset comprised of 32x32 pixel 3-color channel hand-labeled subset of the TinyImages dataset (Torralba et al., 2008) featuring everyday objects and animals. CIFAR10 is composed of 50,000 training and 10,000 test data, evenly split among the 10 classes. Finally, the Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2018) is a dataset composed of sentences and labels corresponding to the grammatical correctness of the sentence compiled from texts on grammar. CoLA includes 9515 training sentences and 1049 test sentences. CoLA was included in the original GLUE (Wang et al., 2018) set of benchmarks for NLP, which became the de-facto benchmark set of tasks for general language modeling.

H.2 EXPERIMENTS

H.2.1 100 FULLY CONNECTED MNIST2 MODELS

Using the first two classes of MNIST, (MNIST2), we train 100 independent 4-layer fully connected NNs using PyTorch. The network layer widths were [100,100,100,1], and each had a Rectified Linear Unit (ReLU) activation function, except for the final layer. We define all of our networks to terminate without a final activation for the sake of calculating our trNTK; however, we use the sigmoid link function to map the activations onto a value we interpret as probability of class 1. As is typical in NTK parameterization, we divided each activation map by the square root of the preceding layer's width. The input space of MNIST was modeled as a 784-size feature vector that we preprocessed to have values between 0 and 1 by dividing by the maximum pixel value of 255. For simplicity, we down sampled the test dataset to share an equal amount of class 0 and class 1 examples, giving 980 examples of each class. We initialized the layers using the normal distribution.

Each model instance had the same hyperparameters, architecture, and approximate training time. The only differences were the initialization given by seed and the stochastic sequence of datapoints from a standard PyTorch data-loader. We trained our model to minimize the binary cross entropy loss between the true labels and the prediction function. We chose to optimize our model using stochastic gradient descent with no momentum and static learning rate 1e-3. Training 100 models sequentially takes approximately 8 hours on a single A100 GPU.

H.2.2 100 CNN MNIST2, FMNIST2, AND CIFAR2 MODELS

We use the same CNN architecture for our 100 MNIST2, FMNIST2, and CIFAR2 models; for brevity, we will describe the model once. Each model is a 12-layer NN where the first 9 layers are a sequence of 2D convolutional layers and 2D Batch Normalization layers. The final 3 layers are fully connected. The first nine layers are split into three sections operating on equal feature map sizes (achieved with padding). The first layer in each section is a convolutional layer with kernel size 3 and padding size 1 followed by a batch normalization layer, followed by a second convolutional layer with kernel size 3 and padding size 1 but with stride = 2 to reduce the feature map in half. The number of filters steadily increases throughout each convolutional layer as [8,8,16,24,32,48,64]. After the convolutional layers, a flattening operation reduces the image dimensions into a 1-dimensional vector. Next, fully connected layers of widths [256, 256, 1] are applied. After each convolutional layer and fully connected layer we apply the rectified linear unit (ReLU) activation. Training times for 100 models on MNIST2, CIFAR2, and FMNIST 2 were 15 hours (100 epochs), 5 hours (100 epochs), and 48 hours (200 epochs), respectfully, on a single A100 GPU. The difference in times can be explained by the different choices of batch size and number of epochs, which were 4, 64, and 4, respectfully. We chose these batch sizes, and all other hyperparameters, by hand after a small search that stopped after achieving comparable performance the many examples of models available online for these benchmark tasks. One oddity we believe worth mentioning is that we subtract the initial model's final activation vector for the CIFAR2 model, after observing that this lead to a modest improvement. Initial LRs were 1e-3 for each model, but the optimizers were chosen as SGD, Adam, and Adam for MNIST2, CIFAR2, and FMNIST2, respectfully.

H.2.3 4 COLA BERT-BASE MODELS

To train the 4 BERT-base models, we downloaded pre-trained weights available on the HuggingFace repository for BERT-base no capitalization. We then replaced the last layer with a two-neuron output fully connected layer using HuggingFace's API for classification tasks. We set different seeds for each model instance, which sets the random initialization for the final layer. We train our model on the COLA dataset for binary classification of sentence grammatical correctness. We train our model using the the AdamW optimizer (Loshchilov and Hutter, 2017) with an initial learning rate η = 2e-5. We allow every layer to update. Training is done over 10 epochs after which the training accuracy is seen to exceed 99% performance on each model. Training takes a few minutes on an A100 GPU. Calculating the NTK is achieved by splitting the parameter vector into each learnable tensor's contribution, then parallelizing across each tensor. Each tensor's trNTK computation time depends upon the tensor's size. In total the computation takes 1200 GPU hours, on single A100 GPUs.

H.2.4 LARGE COMPUTER VISION MODELS

We downloaded 3 pre-trained model weights files from an independent online repository (Phan, 2021). ResNet18 and Resnet34 architectures can be found described in He et al. (2015b), and MobileNetV2 can be found described in Sandler et al. (2018). Each model's trNTK was computed by parallelizing the trNTK computation across each learnable tensor. the computation time varies as a function of the learnable tensor's size, but the total time to compute each of ResNet18, ResNet34, and MobilenetV2 was 389, 1371, and 539 GPU hours, respectfully, on single A100 GPUs.

H.2.5 CNN FOR POISONED DATA EXPERIMENT

We trained a 22 layer CNN with architecture described in the repository alongside Shan et al. (2022) and restated here. The architecture's first 15 layers are composed of a 5 layer repeating sequence of convolution, batch normalization, convolution, batch normalization, and max pooling. After the 15th layer, we flatten the feature vector, apply another max pooling operation, and then apply dropout with probability 0.2. The next parameterized layers consist of the sequence fully connected layer, batch normalization, fully connected layer, batch normalization and final fully connected layer. A ReLU activation is applied between each hidden layer. The repository of Shan et al. (2022) generates BadNet cifar10 images as a data artifact. We translate their architecture to PyTorch and train our own model. The model was trained to minimize the cross entropy loss on the poisoned image dataset with stochastic gradient descent with an initial learning rate of 1e-2. The total number of parameters for this model is 820394. We take a different approach to calculate the trNTK of this model and choose not to parallelize the computation across each learnable tensor. The total trNTK calculation completed in 8 hours on a single A100 GPU.

H.3 COMPUTING EMBEDDING KERNELS

To compute an embedding kernel we must make a choice of what constitutes a “layer”. This has some slight nuance, as for example, the most complete Embedding kernel would be computed after every modification to the feature space. In a typical fully connected layer there would be 2-3 modifications that occur: 1) the weight matrix multiplication; 2) the bias vector addition; 3) the activation function. Typically, we would take each of these modifications as part of the same fully connected layer and sample an activation for the Embedding following all three. Next, consider residual blocks and similar non-feed forward or branching architectures. We must make a choice of where to sample in the branch that may have an impact on how the final Embedding kernel behaves. In this appendix, we list our choice of layers to sample the activation for each experiment. We chose to balance completeness and computation time. Follow on work could investigate how these choices affect the final embedding kernel.

H.3.1 RESNET18

Table 5 shows where the components of the embedding kernel were calculated.

Table 5: Embedding Layers ResNet18 with $x \in \{1, 2, 3, 4\}$

Layername
conv1
bn1
maxpool
layer.x
layer.x .0
layer.x.0.conv1
layer.x.0.bn1
layer.x.0.conv2
layer.x.0.bn2
layer.x.1
layer.x.1.conv1
layer.x.1.bn1
layer.x.1.conv2
layer.x.1.bn2
avgpool
fc

H.3.2 BERT-BASE

The layers used to calculate Bert-base embedding kernel are shown in Table 6.

H.3.3 POISONED CNN

Table 7 shows after which modules the embedding kernel was calculated for the data poisoning CNN.

Table 6: Bert-base Layers with Embedding Kernel calculation, $x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

Layername
bert.embeddings
bert.embeddings.word_embeddings
bert.embeddings.position_embeddings
bert.embeddings.token_type_embeddings
bert.embeddings.LayerNorm
bert.encoder.layer.x
bert.encoder.layer.x.attention
bert.encoder.layer.x.attention.self
bert.encoder.layer.x.attention.self.query
bert.encoder.layer.x.attention.self.key
bert.encoder.layer.x.attention.self.value
bert.encoder.layer.x.attention.output
bert.encoder.layer.x.attention.output.dense
bert.encoder.layer.x.attention.output.LayerNorm
bert.encoder.layer.x.intermediate
bert.encoder.layer.x.intermediate.dense
bert.encoder.layer.x.intermediate.intermediate_act_fn
bert.encoder.layer.x.output
bert.encoder.layer.x.output.dense
bert.encoder.layer.x.output.LayerNorm
bert.pooler
bert.pooler.dense
classifier

Table 7: Embedding Layers Poisoned CNN

Layername
conv2d
batch_normalization
conv2d_1
batch_normalization_1
max_pooling2d
conv2d_2
batch_normalization_2
conv2d_3
batch_normalization_3
max_pooling2d_1
conv2d_4
batch_normalization_4
conv2d_5
batch_normalization_5
max_pooling2d_2
max_pooling1d
dense
batch_normalization_6
dense_1
batch_normalization_7
dense_2

I METHODOLOGY FOR LINEARIZING NNS VIA KGLMS

We describe the procedure to achieve a linearization of the NN via a kGLM surrogate model. First, we fit a supervised NN using standard techniques. Next, we compute the trNTK. This kernel acts as the feature space of the kGLM. we fit the kGLM (Pedregosa et al., 2011) (`sklearn.linear_model.SGDClassifier`) using the kernels computed from the same training data as the NN is trained upon. The dimensionality of the output vector from the kGLM will be the same as the NN, and is equal to the number of classes.

We are concerned with demonstrating that after applying an invertible mapping function Φ, the NN decision function is approximately equal to the kGLM decision function. Because the decision function is typically only a function of the probabilities of each class, this objective can be achieved by showing the following approximation holds:

$$\sigma(F(\boldsymbol{x}\,;\boldsymbol{\theta})) \approx \Phi(\mathrm{kGLM}(\boldsymbol{x})).$$

Across many models and datasets we generally observed that the trend between the NN activation and the kGLM activation was “S-shaped”, or else was already linear. The analytic class of function that are “S-shaped” are sometimes called sigmoid functions. The following three functions are used to map the kGLM to the NN.

$$\begin{aligned}
\Phi^1(x) &= \nu x + \mu, \\
\Phi^2(x) &= \nu \frac{\exp(\frac{x-\alpha}{\beta})}{1+\exp(\frac{x-\alpha}{\beta})} + \mu \\
\Phi^3(x) &= \frac{\nu}{\pi} \arctan\left(-\frac{x-\alpha}{2\beta}\right) + \frac{1}{2} + \mu.
\end{aligned}$$

Φ^1 is a linear re-scaling. Both Φ^2 and Φ^3 are sigmoid-shaped functions that map $(-\infty, \infty)$ to (0,1). All choices of Φ are invertible. We made these choices for ϕ after observing the relationship between the kGLM and the NN. We fit Φ functions with an iterative optimizer (Virtanen et al., 2020) on the L_2 loss between $F(\tilde{\boldsymbol{X}}\,;\boldsymbol{\theta})_c)$ and $\Phi(\mathrm{kGLM}(\boldsymbol{X})^c)$, where c is chosen to be class 1 in the case of binary classification (we describe changes necessary for multi-classification below). Fits are completed over a partition of half the test dataset and evaluated on the remaining half. The linearizations are visualized in Appendix I.1.

To visualize we use scale using the logit function. We define the logit function as the scalar-valued function that acts on the softmax probability $p \in (0,1)$ of a single class and outputs a “logit” as:

$$\mathrm{logitfn}(\boldsymbol{x}) = \log \frac{\boldsymbol{x}}{1-\boldsymbol{x}}$$

Using the logit creates a better visualization of the probabilities themselves by smoothing out the distribution of values across the visualized axes. As a final implementation note, we observed some numerical instability due to values being so close to p=1 that errors occur in re-mapping back into logits. We choose to mask out these values from our fit, our visualization, and the R^2 metric.

I.1 VISUALIZATIONS OF POINT-FOR-POINT LINEAR REALIZATIONS FOR EACH EXPERIMENT

What follows are visualization of the linearizations of the NN logits with respect to the kGLM logits. A perfect fit would line up with parity, shown as a diagonal dashed line in each plot. The coefficient of determination or R^2 is shown in text for each plot. Seeds are shown in each panel’s title. For the classification models ResNet18, ResNet34, and MobileNetV2, we flatten out the regressed vector and choose to plot the distribution as a KDE estimate of the correct class and incorrect classes instead of a scatter plot, due to the large number of points.

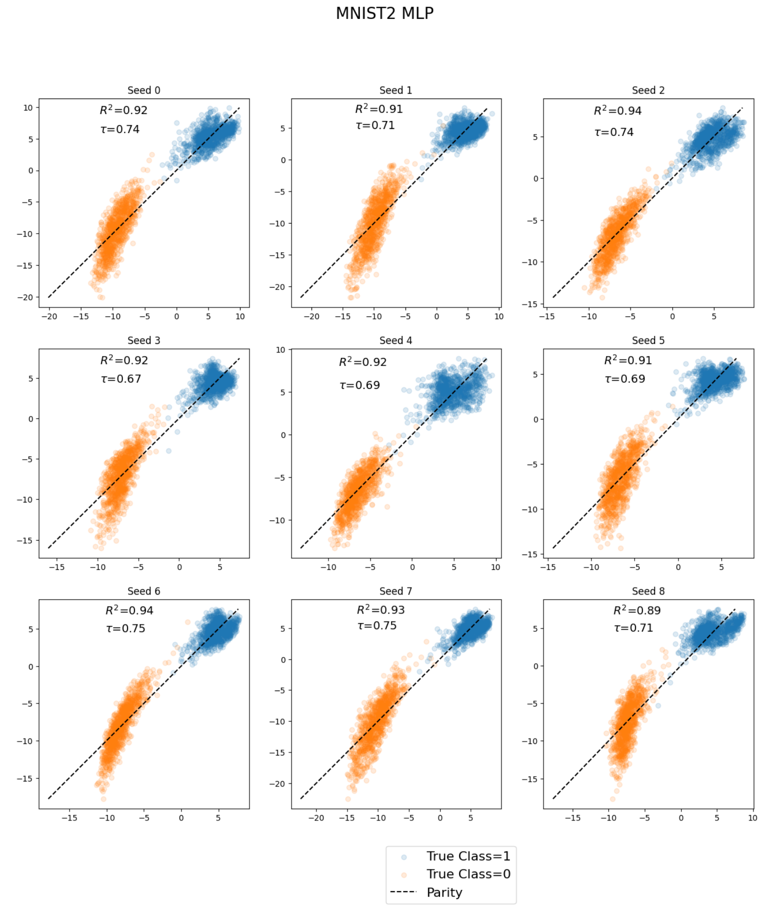


Figure 6: **MNIST2 MLP Linearization**

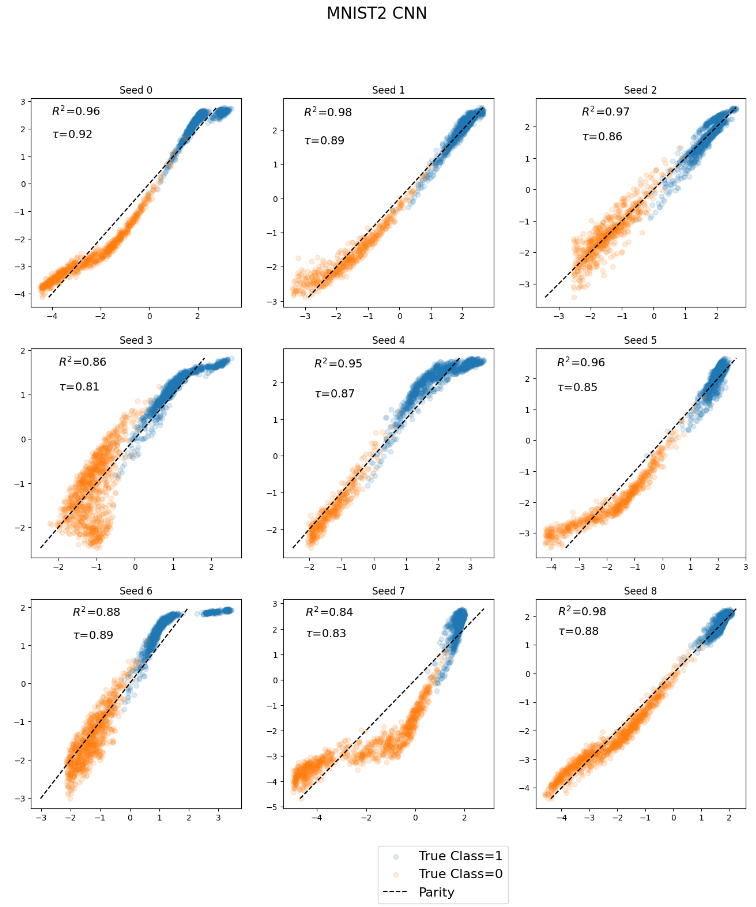


Figure 7: **MNIST2 CNN Linearization**

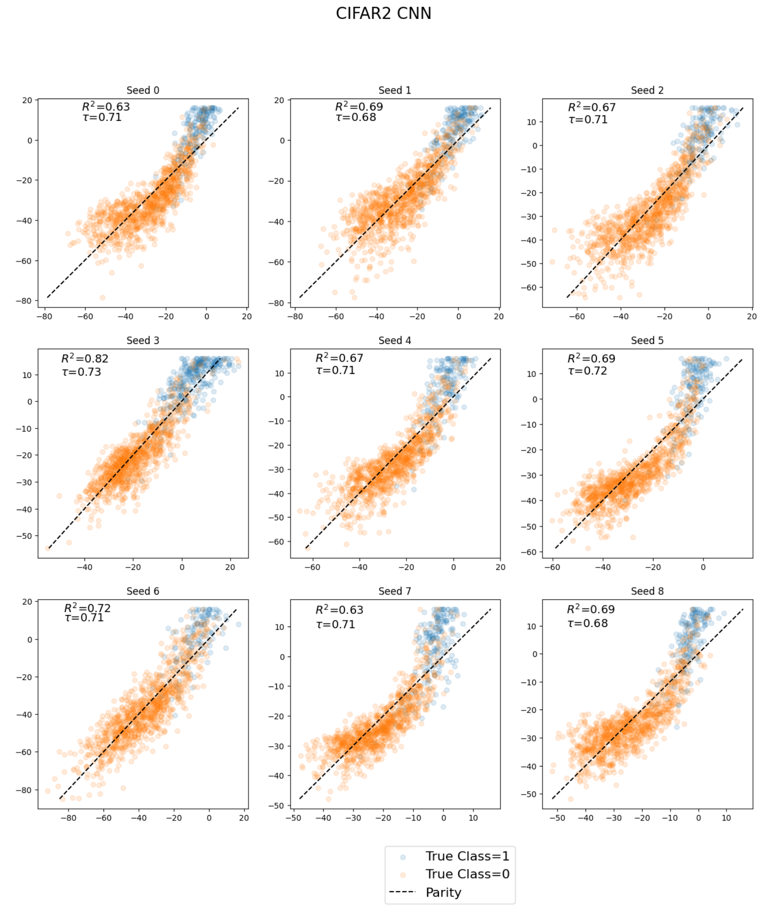


Figure 8: **CIFAR2 CNN Linearization**

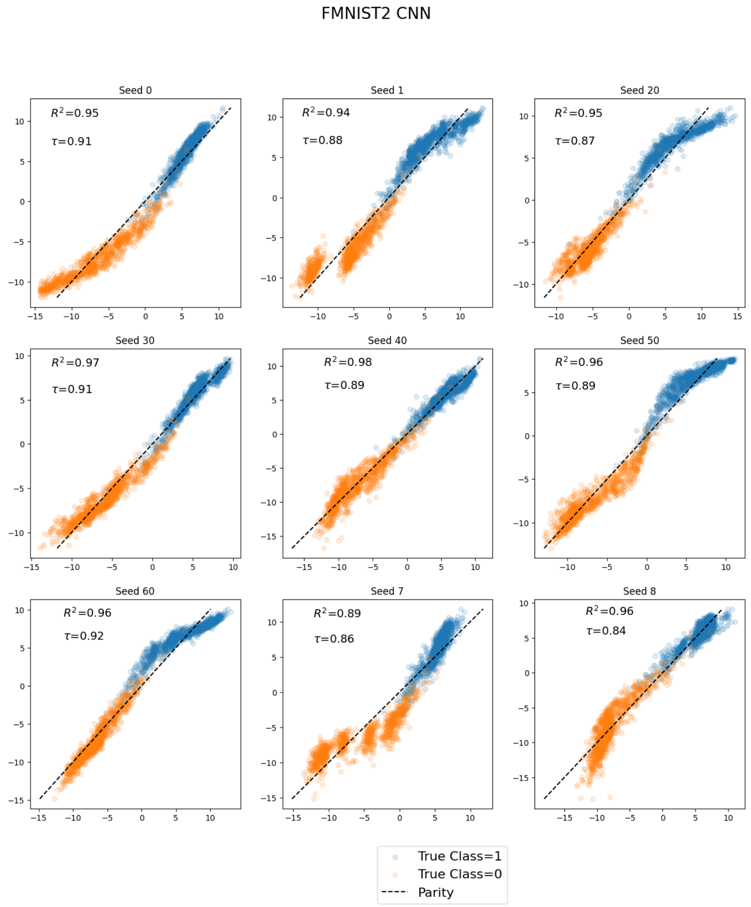


Figure 9: **FMNIST2 CNN Linearization**

J ATTRIBUTION VISUALIZATIONS EXPLAINED

In this appendix we describe the methodology used to visualize the attribution in greater detail. Our kGLM architecture gives each kernel value a unique weight for each output neuron in the NN. For example, in our visualized CIFAR10 ResNet18 network, there are 10 learned weights for each kernel value. For each column we plot a line representing the average attribution given by training examples in that class. By design, multiplying the average attribution from each class by the number of points in each class (in CIFAR10 this is a uniform 5,000 for each class) and summing will result in the logit value of the $kGLM$ in that class. We can therefore use these visualizations to quickly compare this

$$N \times (\frac{1}{N}\sum_{i=1}^{N} A(\boldsymbol{x}, \boldsymbol{x}_i)) = \mathrm{kGLM}(\boldsymbol{x}) \tag{19}$$

When visualizing, we choose to hide the attribution from each training datapoint to the activation of the class c if the training datapoint's true label is not c, by slightly modifying the attribution. Let N_c be the number of datapoints in class c. Let S_c be the set of training datapoint indices with true label $\boldsymbol{z} = c$. Let $S_{\not c}$ be the set of training datapoint indices with true label $\boldsymbol{z} \neq c$. Finally, assume the classes are balanced, as is the case for CIFAR10. Therefore, the length of the set $S_{\not c} = N - N_c$. Then A_{viz} gives the attribution we visualize for $i \in S_c$:

$$A_{\text{viz}}(\boldsymbol{x}, \boldsymbol{x}_i) = \sum_{i \in S_c}^{N_c} W_{c,i}\, \boldsymbol{\kappa}(\boldsymbol{x}, \boldsymbol{x}_i) + \frac{B_c}{N_c} + \frac{1}{N_c}\sum_{j \in S_{\not c}}^{N-N_c} W_{c,i}\, \boldsymbol{\kappa}(\boldsymbol{x}, \boldsymbol{x}_j). \tag{20}$$

In words, we have evenly distributed the attribution from training datapoints not in class c to the training datapoints in class c. Future work can investigate the human-AI interaction from different methods of visualization to determine a most informative visualization technique.

J.1 ADDITIONAL ATTRIBUTION VISUALIZATIONS

In the following subsection we visualize additional examples of attribution from the ResNet18 CIFAR10 experiment as boxplots. In the first subsection, we visualize the mean value of attribution for each logit. In the second subsection, we focus down onto the correct logit and visualize the distribution of attribution explaining that logit's value.

J.1.1 MEAN VALUE OF ATTRIBUTION IN EACH LOGIT

J.1.2 VISUALIZING PREDICTED CLASS ATTRIBUTION MASS

Each figure shows the attribution distribution from each training data class for the predicted logit. Each sub-panel shows a different kernel function with the logit visualized labeled in the title. Each sub-panel is a boxplot with a dark line representing the mean contribution of attribution mass from that class. For our most consistent performing trNTK kernel function, the mean contribution is within the inner quartile range for every test image.

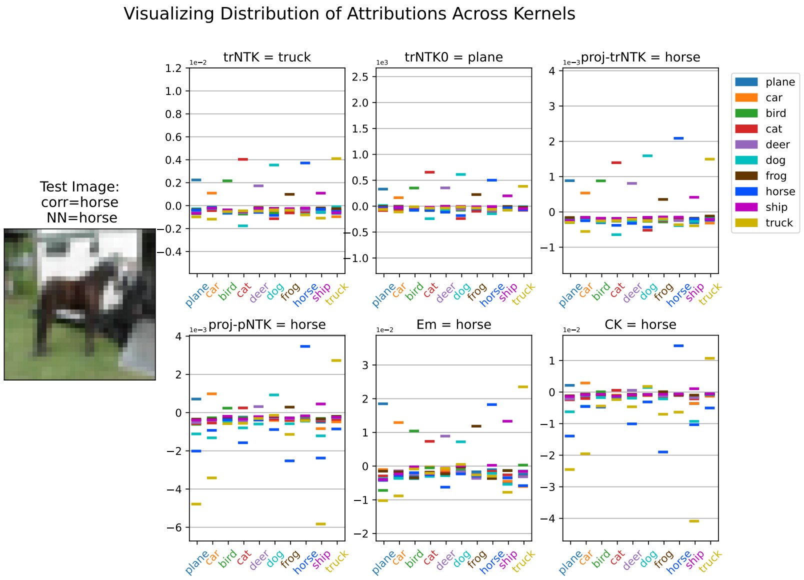


Figure 10: An image of a horse with human handler (right side) standing in front of a trailer or truck. The NN correctly classifies the image as a horse with close runner-up secondary classification as a truck, which we might consider excusable given the presence of both a horse and truck in the image.

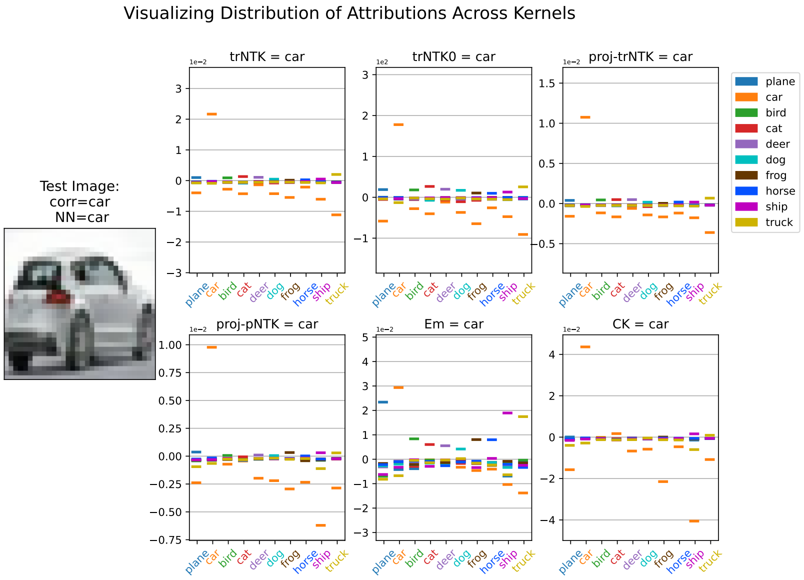


Figure 11: An image of a silver car is correctly classified as a car.

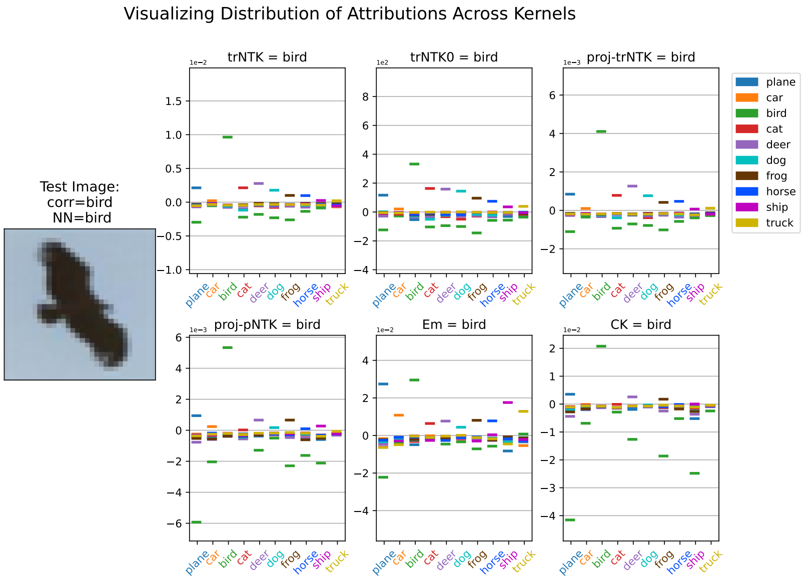


Figure 12: A bird against a blue sky is correctly classified as a bird.

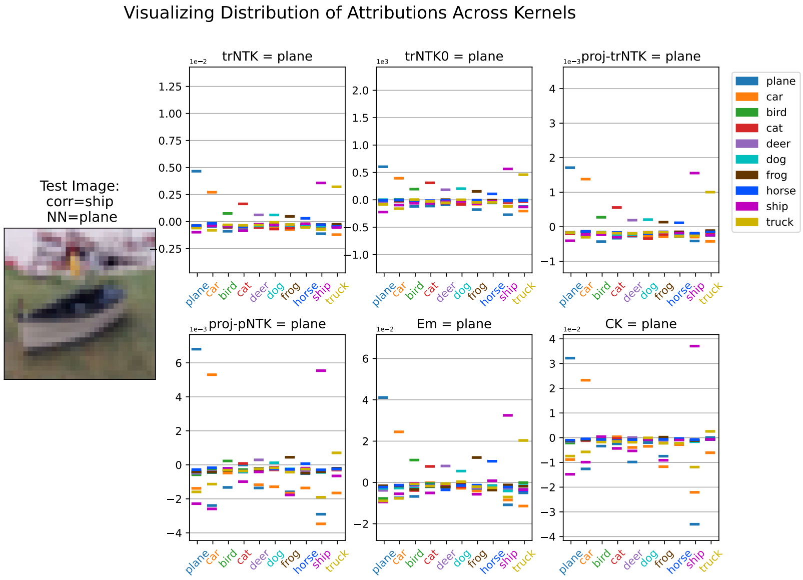


Figure 13: A small boat resting on grass is incorrectly classified as plane by the NN. We show that many kernels also follow the networks misclassification, which is an important property for a surrogate model.

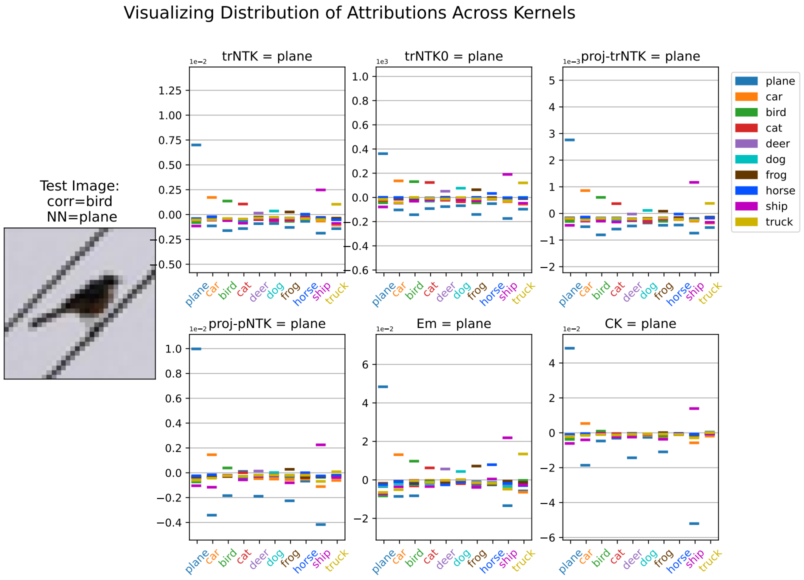


Figure 14: A bird resting on a wire is misclassified as a plane by the NN.

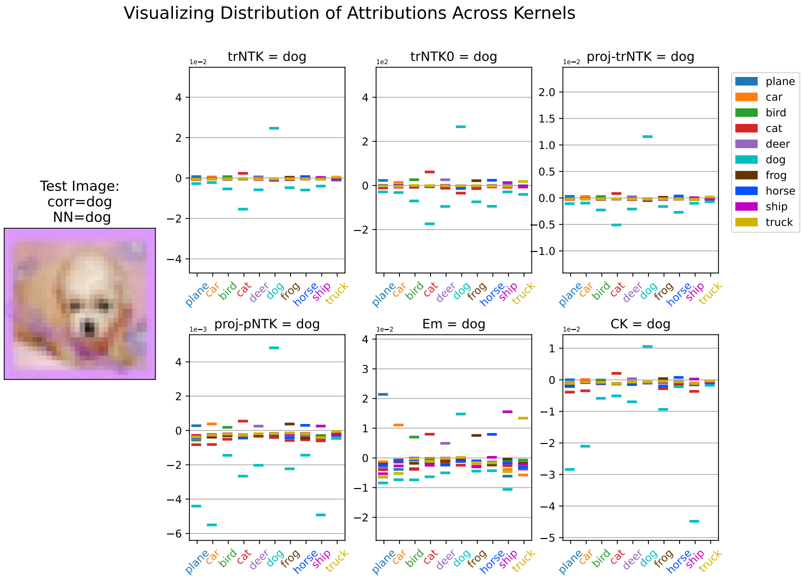


Figure 15: A dog in a pink background frame is classified correctly as a dog.

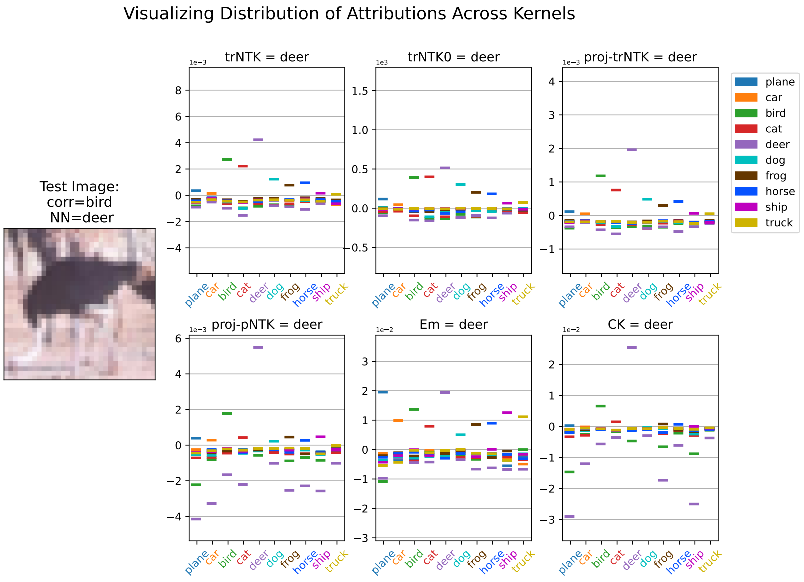


Figure 16: A large bird is misclassified by the NN as a deer.

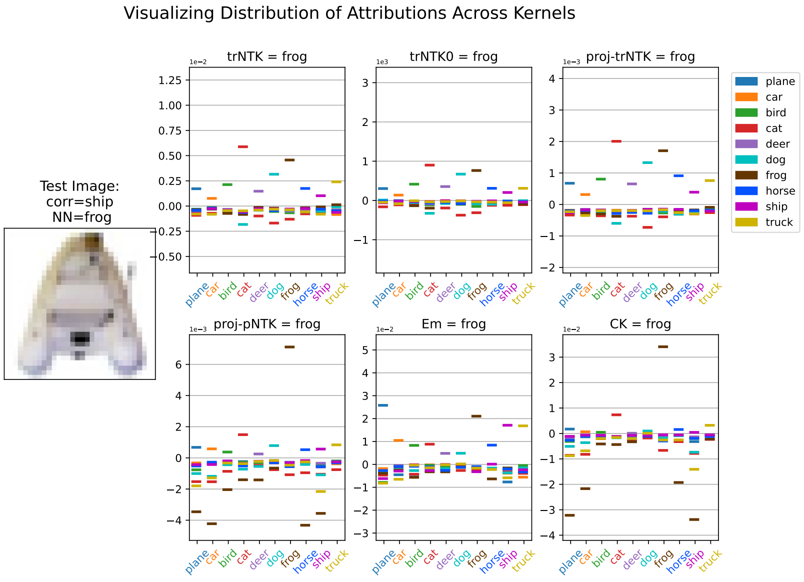


Figure 17: An inflatable boat is misclassified as a frog by the NN.

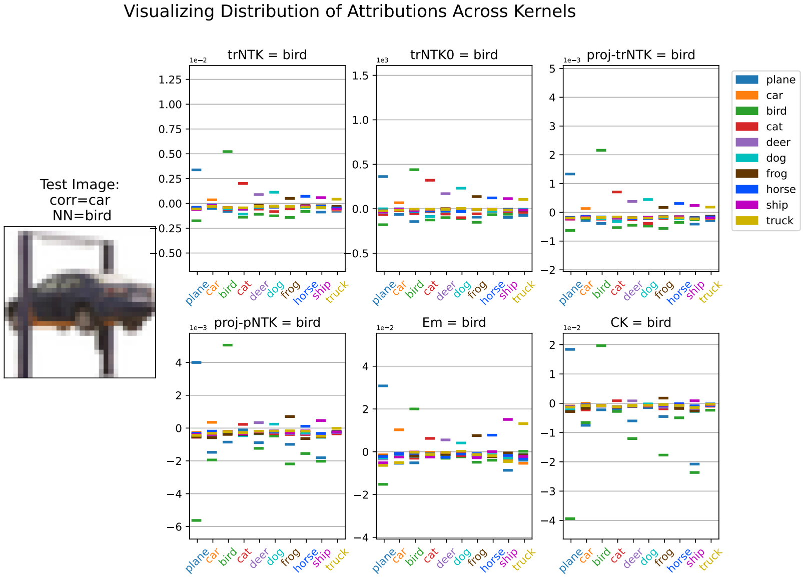


Figure 18: A car elevated on a platform is misclassified as bird by the NN.

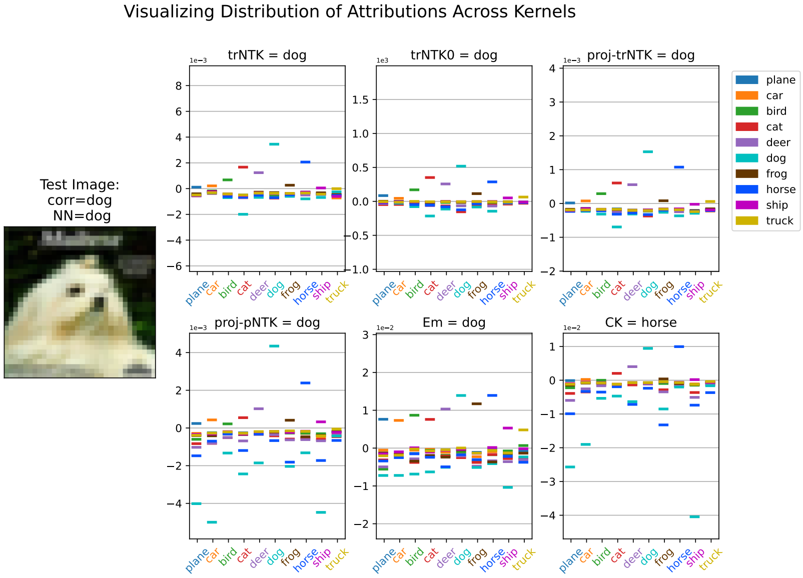


Figure 19: A dog with blurry text overhead is correctly classified as a dog.

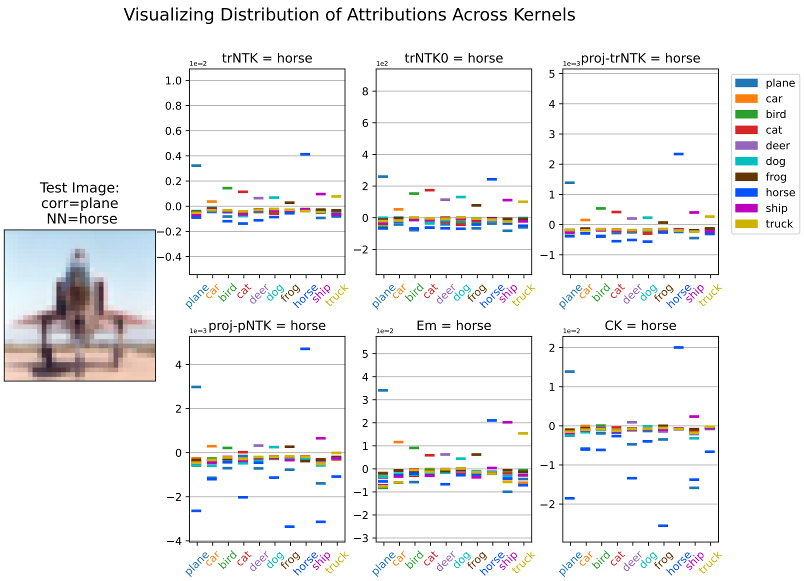


Figure 20: A person sitting on the nose of a large plane faces the camera and is misclassified as a horse.

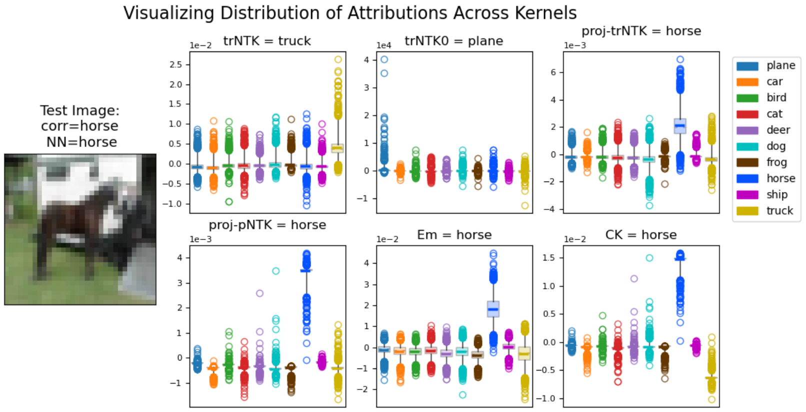


Figure 21: An image of a horse with human handler (right side) standing in front of a trailer or truck. The NN correctly classifies the image as a horse with close runner-up secondary classification as a truck, which we might consider excusable given the presence of both a horse and truck in the image.

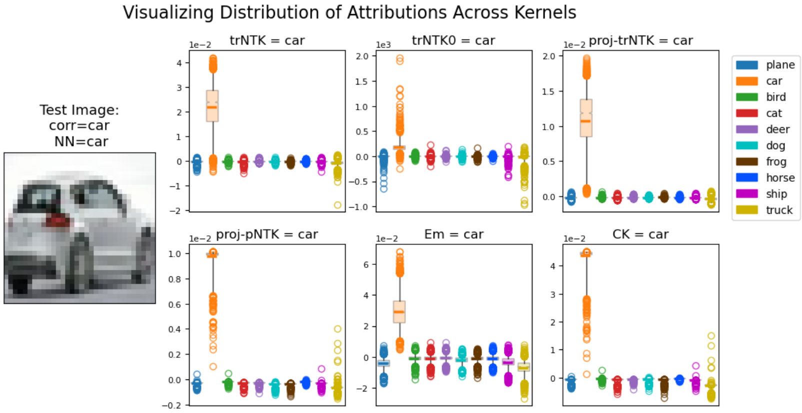


Figure 22: An image of a silver car is correctly classified as a car.

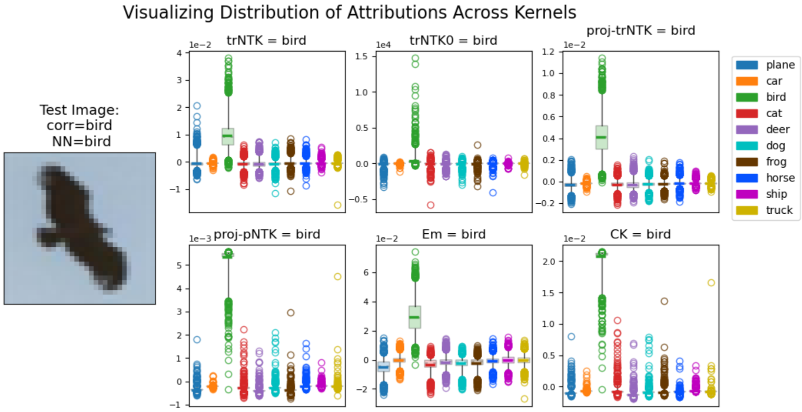


Figure 23: A bird against a blue sky is correctly classified as a bird.

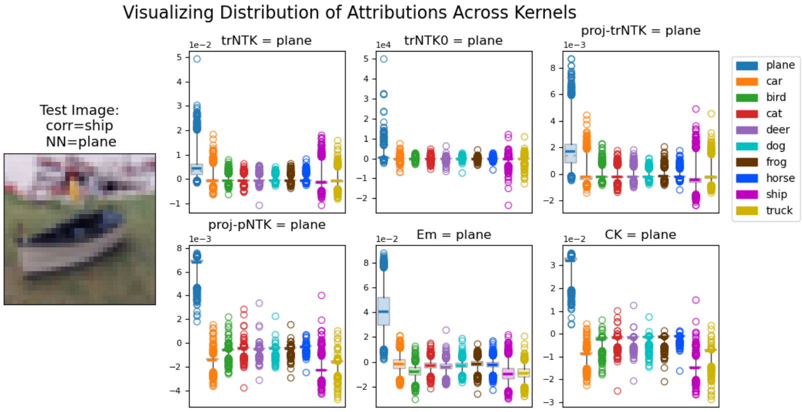


Figure 24: A small boat resting on grass is incorrectly classified as plane by the NN. We show that many kernels also follow the networks misclassification, which is an important property for a surrogate model.

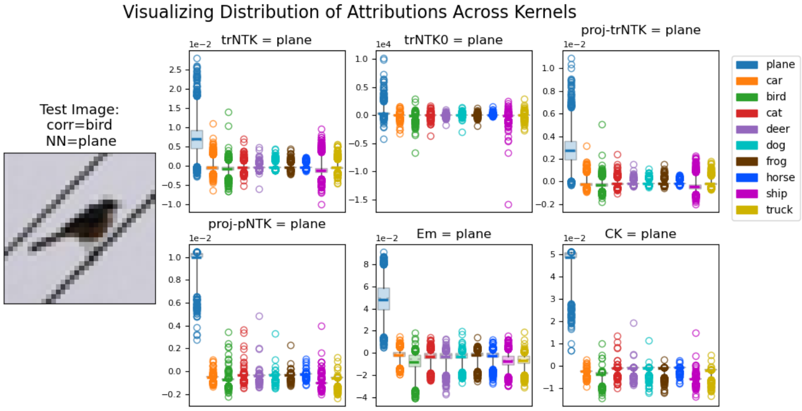


Figure 25: A bird resting on a wire is misclassified as a plane by the NN.

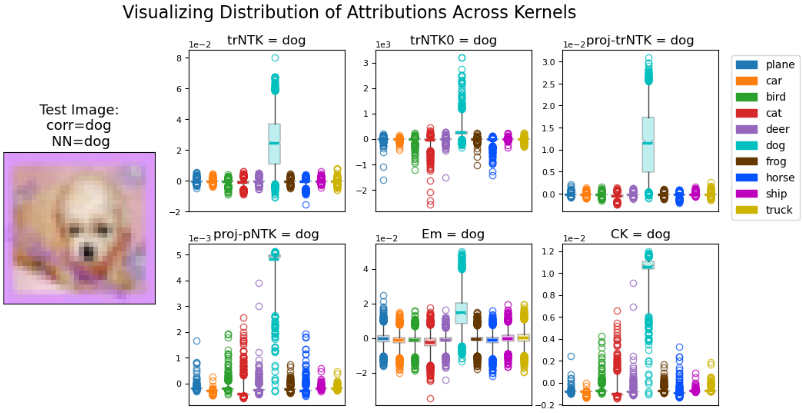


Figure 26: A dog in a pink background frame is classified correctly as a dog.

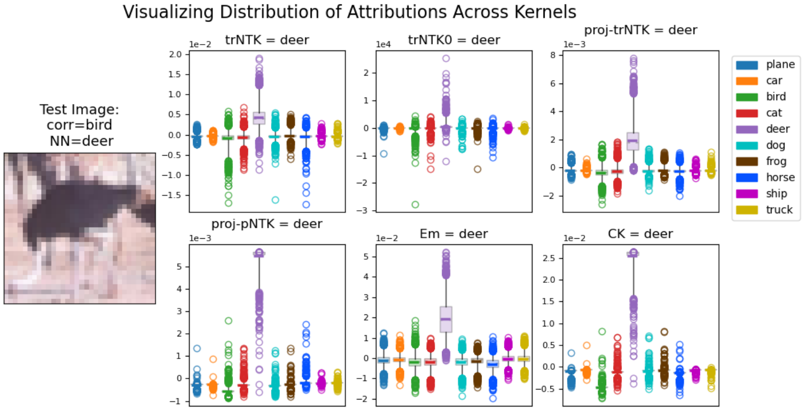


Figure 27: A large bird is misclassified by the NN as a deer.

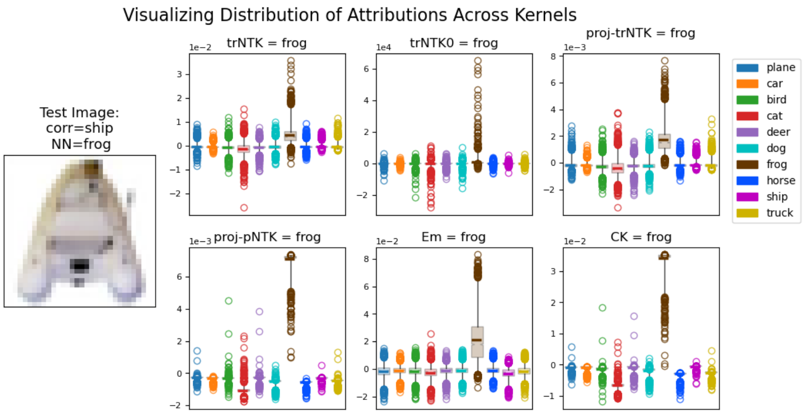


Figure 28: An inflatable boat is misclassified as a frog by the NN.

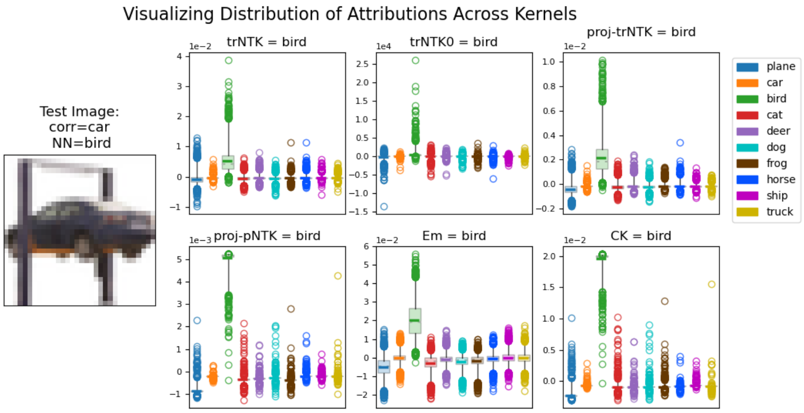


Figure 29: A car elevated on a platform is misclassified as bird by the NN.

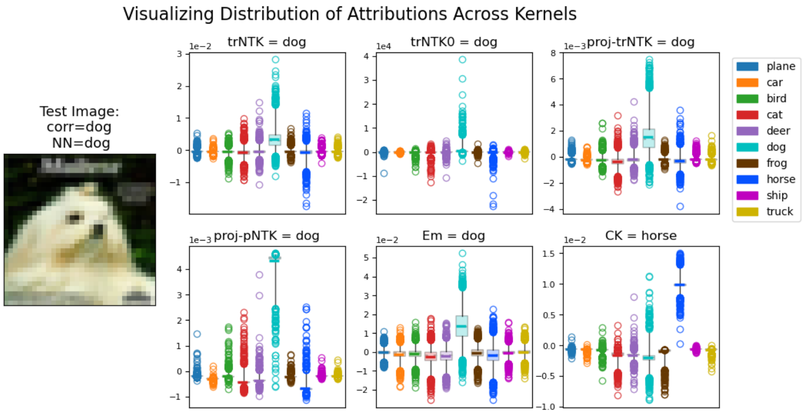


Figure 30: A dog with blurry text overhead is correctly classified as a dog.

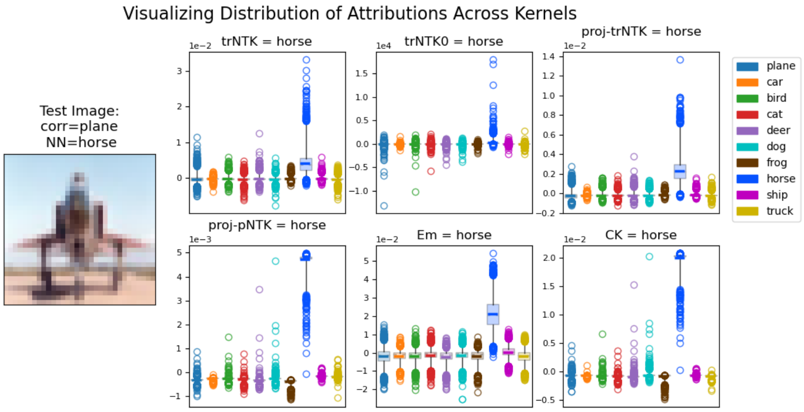


Figure 31: A person sitting on the nose of a large plane faces the camera and is misclassified as a horse.

J.2 TOP FIVE EXEMPLAR ATTRIBUTION VISUALIZATIONS

In the following plots, we visualize the differences between kernels by plotting the top five most similar training images for the same selection of images as in the last appendix. Qualitatively, we observe that test data often share conceptual similarities with the most similar training data as evaluated by the trNTK, and that what is chosen as most similar often reveal something about the kernel itself. For example, the CK kernel is created from the final representation of the neural network. For NN trained until convergence this final representation should have all inner-class variance collapsed (Papyan et al., 2020). Therefore, we expect the CK to mostly show that the test image is highly similar to ALL training images of the predicted class. This is problematic because the inner-class variance is exactly the information we need to understand how the neural network arrived at its decision. Because the top most similar are not tied directly to our kernel surrogate model any explanations we generate from these visualizations are admittedly up to interpretation. Future work could endeavor to evaluate different kernel surrogate models such as a K nearest neighbors, which would tie these visualizations directly to the surrogate model's prediction. This would be a way to recover explain-by-example.

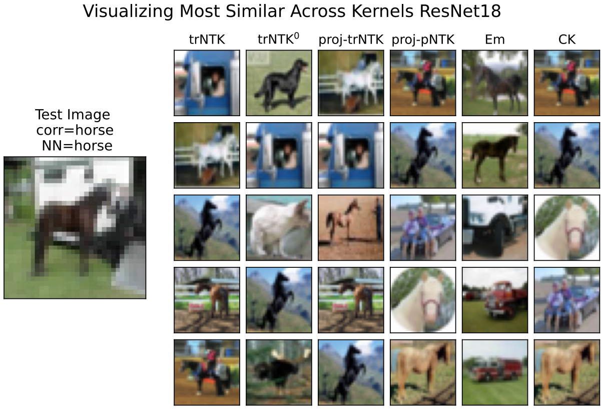


Figure 32: A horse stands next to a human and in front of a trailer or truck is correctly classified as horse by the NN model. Many of the attributed animals are shown in profile, as the subject horse of the original image stands.

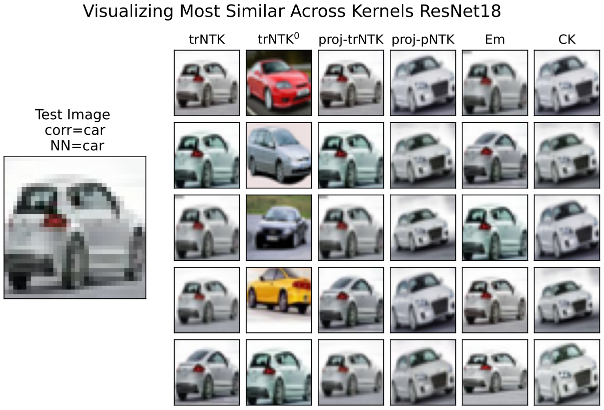


Figure 33: A silver car is correctly classified by the NN. Many similar images (seemingly the same image with different crops) exist in the training dataset.

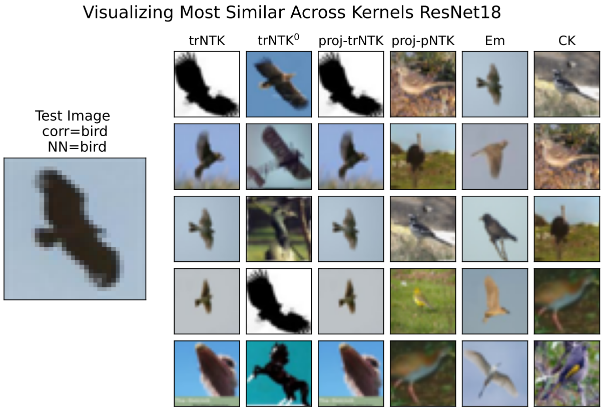


Figure 34: A bird flies with wings spread in a blue sky background and is correctly classified by the NN. Many of the bids attributed to by the evaluated kernels are also flying in a similar manner in a blue sky background.

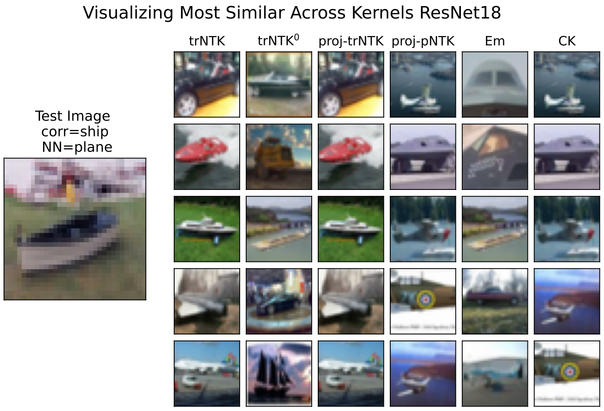


Figure 35: A boat resting on grass is misclassified as a plane by the NN. The most similar attributions are varied, perhaps demonstrating a weakness in this kind of visualization.

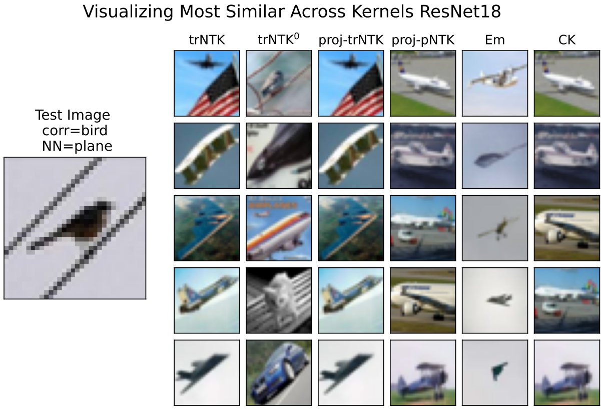


Figure 36: A bird resting on a wire that spans the image diagonally is misclassified as a plane. Many of the highest attributed images from the trNTK and trNTK^0 have a similar diagonal quality, even if the underlying class of the subject of the image is much different than the true or classified class.

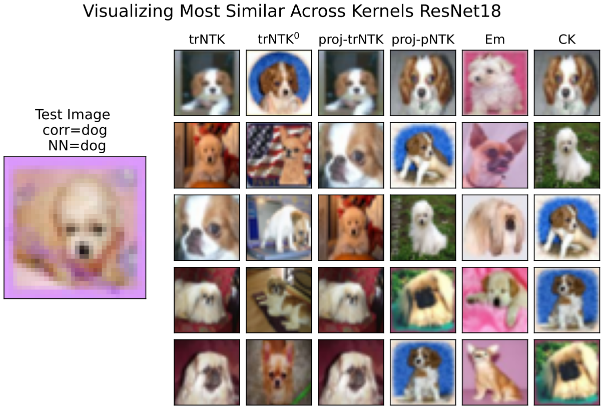


Figure 37: A small puppy in a pink background looking out of the screen (“at the camera”) is correctly classified as a dog. Many of the most similar images are dogs that look out of the screen. The Embedding kernel seems very focused on the background pixel values, as many of the attributions are pink centered.

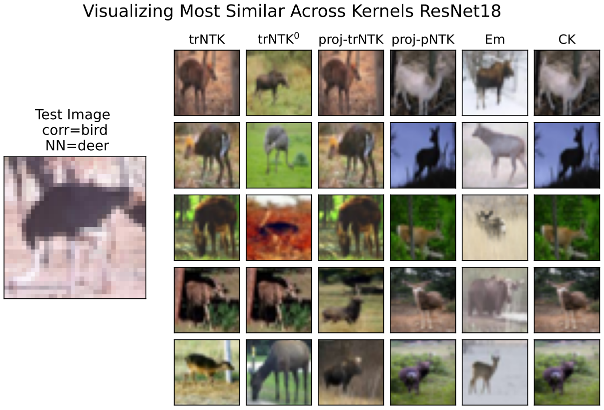


Figure 38: A large bird is misclassified as a deer. The attributed images are varied, perhaps demonstrating a weakness in this kind of visualization.

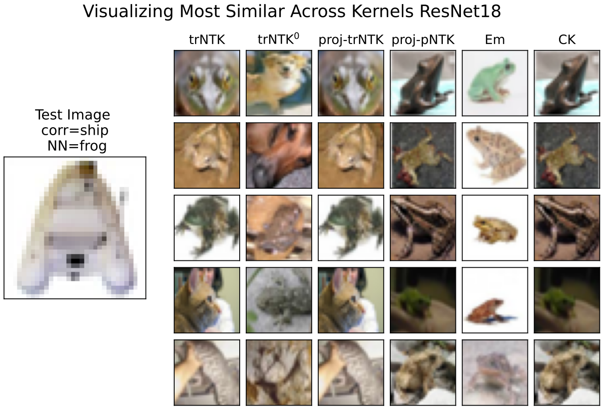


Figure 39: A white inflatable boat is misclassified as a frog. The attributed images are varied, perhaps demonstrating a weakness in this kind of visualization.

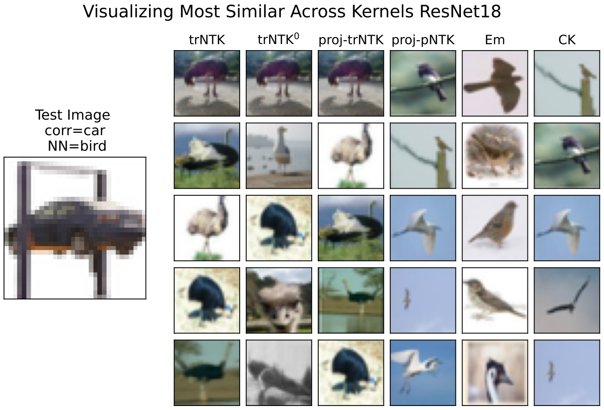


Figure 40: A car resting on a raised platform is misclassified as a bird. Many of the bird attributed to by the trNTK and trNTK^0 are large bird with rotund black bodies and stalky legs, perhaps suggesting a pathway for the misclassification.

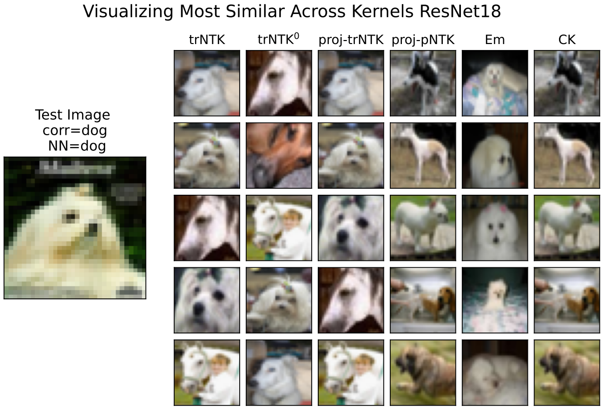


Figure 41: A dog with blurry text overhead is correctly classified as a dog. The attributed images in the trNTK and trNTK^0 are mostly images of animals with white fur "looking left" mirroring the test image of the dog "looking right".

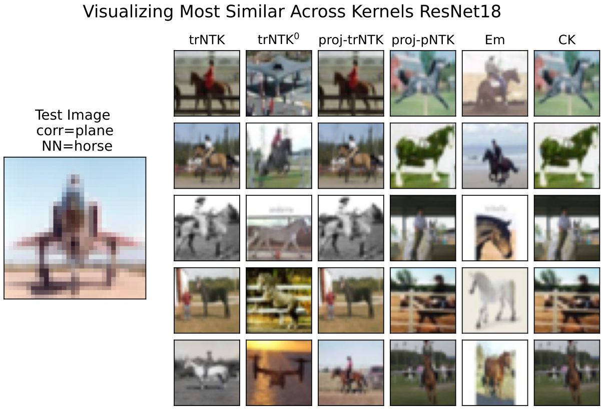


Figure 42: An image of a person sitting on the nose of a plane facing towards the camera. The NN misclassified this example as a horse. The trNTK shows many example of people riding horses, mirroring the person "riding" the plane.

K ADDITIONAL DATA POISONING ATTRIBUTION VISUALIZATIONS

In this appendix we provide additional visualization for the data poisoning experiment attributions. We show the same selection of images as the previous section for comparison. Because the NN classifies nearly every poisoned image as the targeted class deer, we expect that a good surrogate model would reflect this fact by attributing highly to poisoned examples. Because the model trained is a different architecture than the ResNet, it can be interesting to compare the top attributions to the previous appendix section.

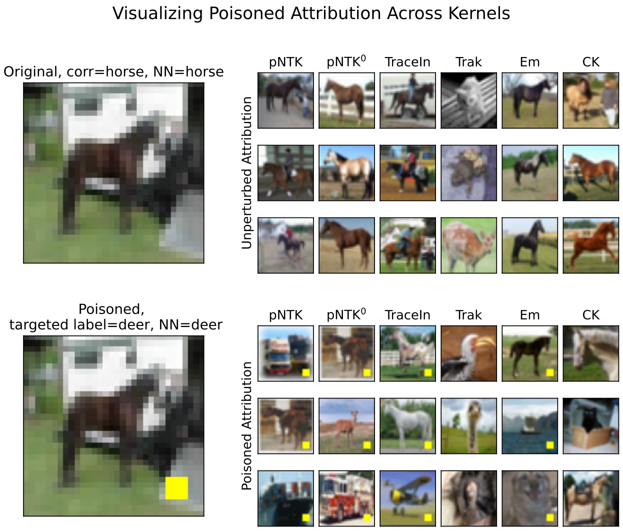


Figure 43

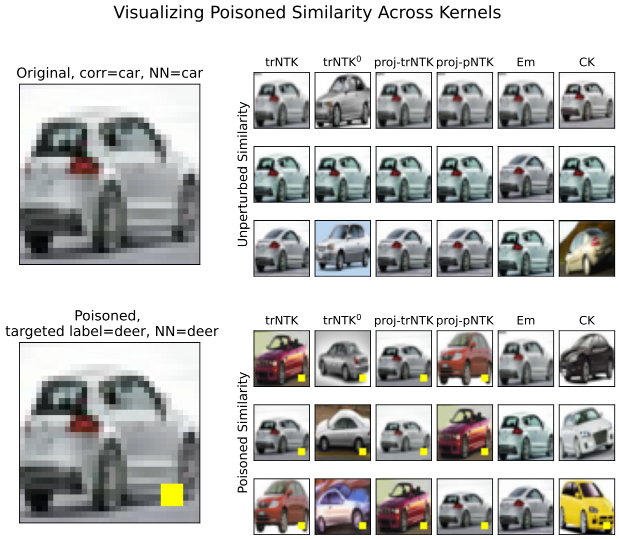


Figure 44

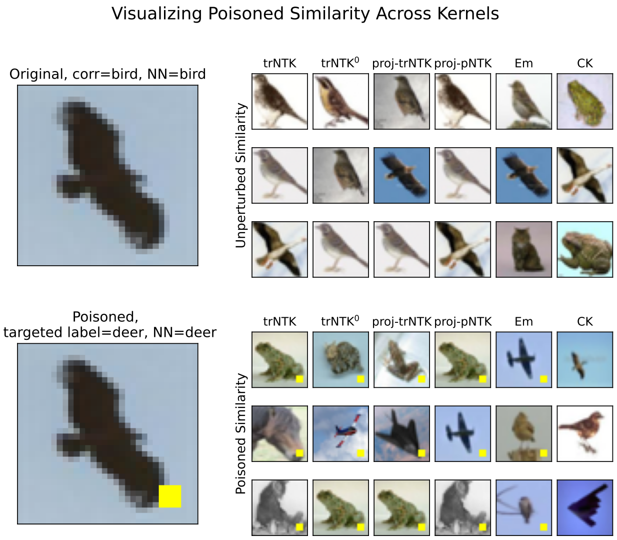


Figure 45

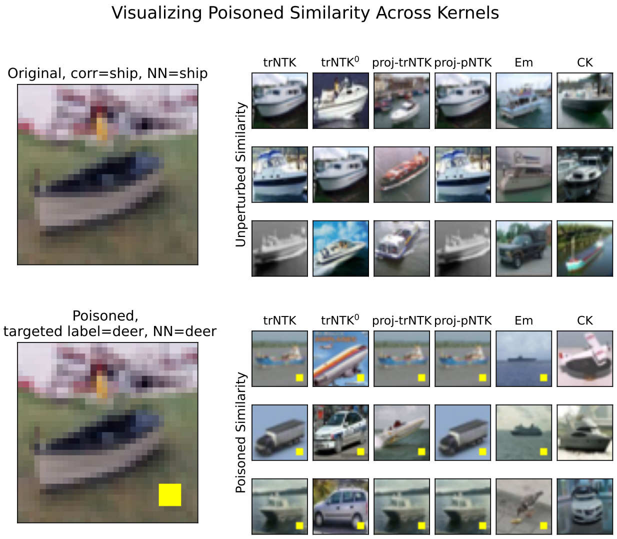


Figure 46

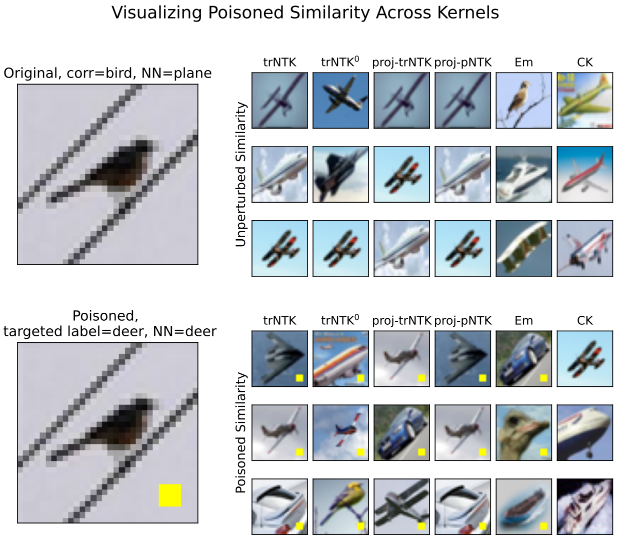


Figure 47

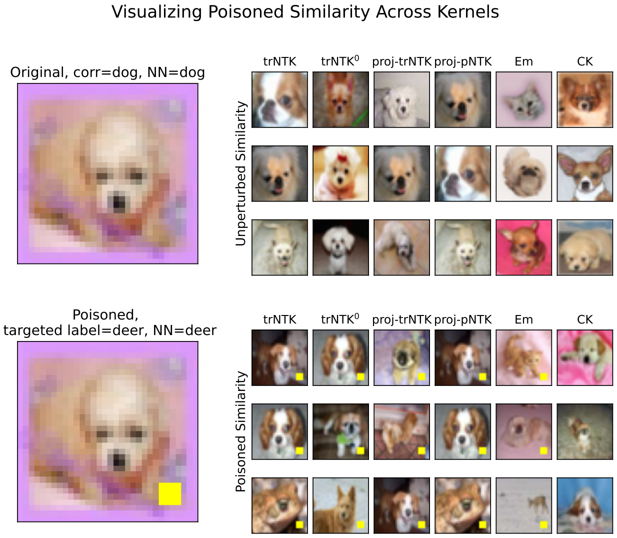


Figure 48

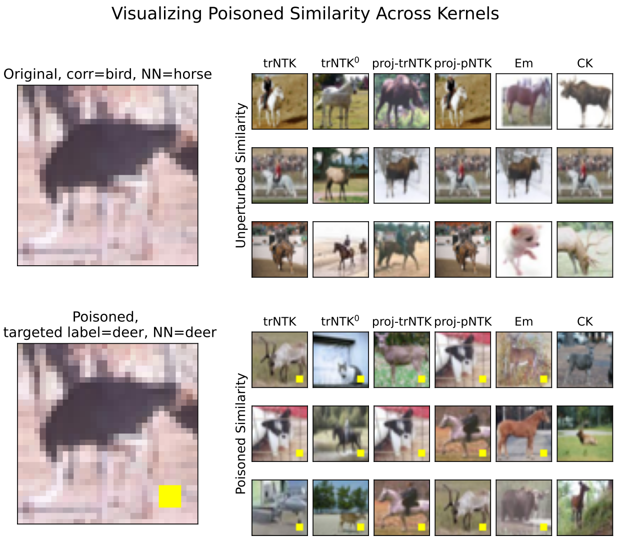


Figure 49

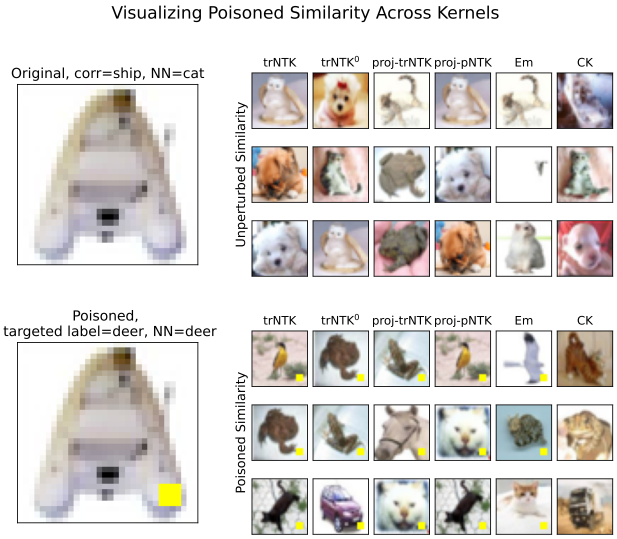


Figure 50

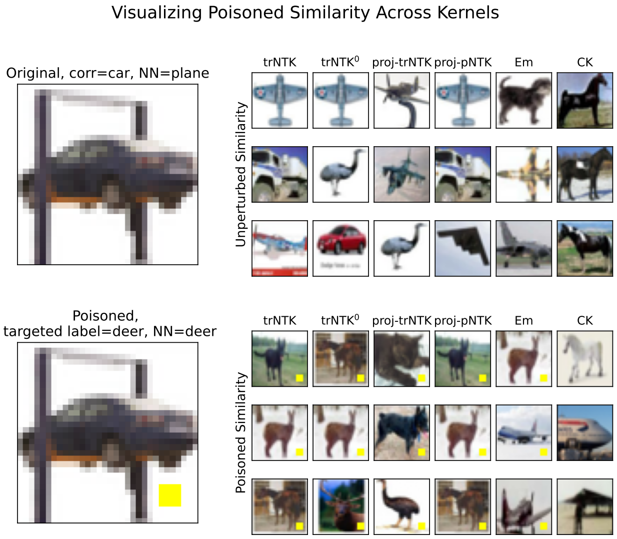

Visualizing Poisoned Similarity Across Kernels
Original, corr=car, NN=plane
Poisoned,
targeted label=deer, NN=deer
trNTK
trNTK⁰
proj-trNTK
proj-pNTK
Em
CK
Unperturbed Similarity
Poisoned Similarity

Figure 51

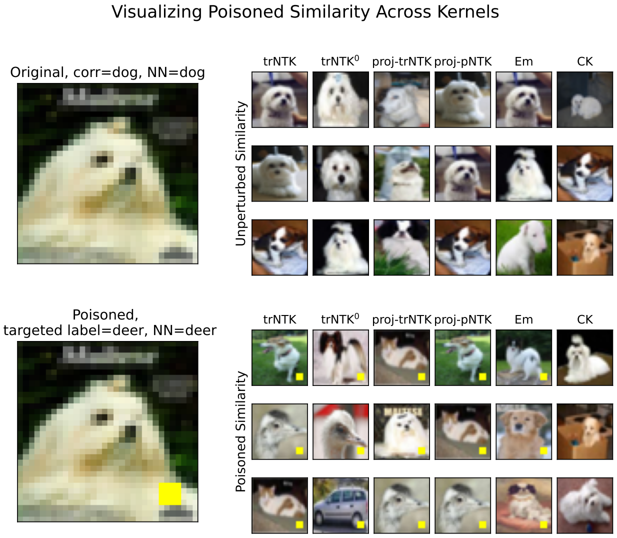

Visualizing Poisoned Similarity Across Kernels
Original, corr=dog, NN=dog
Poisoned,
targeted label=deer, NN=deer
trNTK
trNTK⁰
proj-trNTK
proj-pNTK
Em
CK
Unperturbed Similarity
Poisoned Similarity

Figure 52

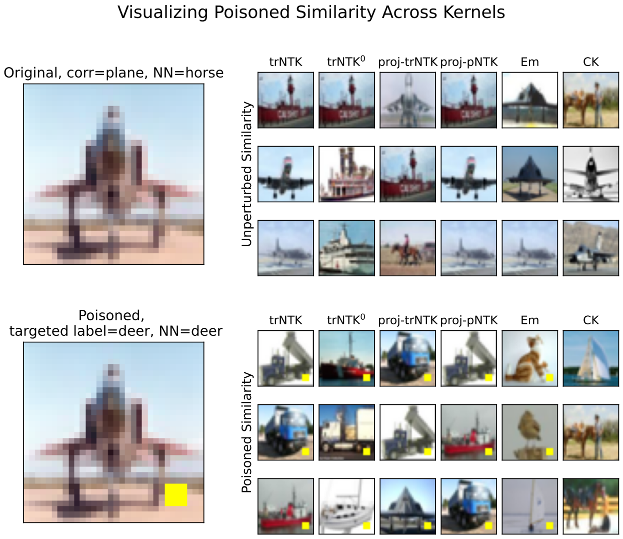

Visualizing Poisoned Similarity Across Kernels
Original, corr=plane, NN=horse
trNTK
trNTK^0
proj-trNTK
proj-pNTK
Em
CK
Unperturbed Similarity
Poisoned,
targeted label=deer, NN=deer
trNTK
trNTK^0
proj-trNTK
proj-pNTK
Em
CK
Poisoned Similarity

Figure 53