Supplement to “Unveiling the Hidden Structure of Self-Attention via Kernel Principal Component Analysis”

Table of Contents

A Experiment Details **18**

A.1 Robustness against Data Corruptions . . . 19

A.2 Robustness under Adversarial Attacks . . . 19

A.3 ADE20K Image Segmentation . . . 19

A.4 WikiText-103 Language Modeling . . . 20

B Calculating the Gram Matrix $\widetilde{K}_\varphi$ **20**

C Plotting J_{proj} in Section 2.2.1 **20**

D Principal Component Pursuit **21**

E Additional Experimental Results **22**

E.1 Reparameterization of Self-Attention . . . 22

E.2 Pairwise Absolute Differences of γ_i and γ_j . . . 23

E.3 ImageNet-C Results by Corruption Type . . . 23

E.4 Square Attack . . . 23

E.5 Results on ADE20K Image Segmentation . . . 24

E.6 Results on RPC-SymViT-base . . . 24

E.7 Results on RPC-Attention in FAN . . . 25

E.8 Results on downstream Natural Language Understanding tasks . . . 25

E.9 Computational Efficiency . . . 27

E.10 Results on Robust Asymmetric Attention . . . 27

F Further Discussion on Related Works **27**

G Broader Impacts **28**

A Experiment Details

Implementation Details of RPC-SymViT: Our RPC-SymViT models have 5.2M parameters, the same as the SymViT baseline. We use a standard tiny configuration with 12 transformer layers, 3 attention heads per layer, and a model dimension of 192 and simply replace softmax attention with RPC-Attention. We follow the training settings as in [78] and their implementation is available at https://github.com/facebookresearch/deit. In a Segmenter model, we use the same RPC-SymViT setting to replace the baseline SymViT backbone. We follow the training details in [72] and their code is publicly available as well, https://github.com/rstrudel/segmenter.

There are 3 hyperparameters: 1) μ: this parameter controls the singular value thresholding operator in the PAP algorithm. We set μ to the recommended value given in Definition 1; 2) λ: this is a regularization parameter that controls the sparsity of the corruption matrix $\boldsymbol{S}$. We finetune λ for training and observe that RPC-SymViT with $\lambda = 3$ yields the best performance for models with 2 iterations per layer and $\lambda = 4$ yields the best performance for models with iterations only in the first layer; 3) n: the number of iterations of the PAP algorithm in a RPC-Attention layer.

Implementation Details of RPC-Symlm: For our language model, we use a standard transformer language model [83], with a symmetric attention layer. The model has a dimension of 128 for the keys, queries and values, while the training and evaluation context length is set at 256. There are 16 layers altogether and 8 heads per layer. Similarly, we replace the softmax attention with RPC-Attention only in the first four layers to save on computational overhead. There are the same 3 hyperparameters as in RPC-SymViT and we use the same value for μ, $\lambda = 4$ and $n = 4$. We also follow the standard training settings as in [70, 47] and the code base developed by [70], available here https://github.com/IDSIA/lmtool-fwp.

A.1 Robustness against Data Corruptions

Datasets: We use the ImageNet-1K dataset that contains 1.28M training images and 50K validation images. There are 1000 classes of images and the model learns an image classification task. For robustness to common corruptions, we use ImageNet-C (IN-C) [31] which consists of 15 different types of corruptions applied to the ImageNet-1K validation set with 5 levels of severity. To test robustness to both input data distribution shifts as well as label distribution shifts, we use ImageNet-A (IN-A) and ImageNet-O (IN-O) [32] respectively. Both of these datasets contain a 200 class subset of ImageNet-1K classes with adversarially filtered images. Finally, we test our model on ImageNet-R (IN-R) [30] which contains various artistic renditions of images. This evaluates the model's generalization ability to abstract visual renditions.

Metrics: On ImageNet-1K, ImageNet-A and ImageNet-R, we report the top-1 accuracies for all experiments. We include top-5 accuracies on ImageNet-1K. On ImageNet-C, the standard metric for evaluation is the mean Corruption Error (mCE). To calculate this, we average the top-1 error rate for each corruption type across the 5 levels of severity and divide them by AlexNet's average errors, then take the final average across all corruption types. We report the area under the precision-recall curve (AUPR) for ImageNet-O which requires anomaly scores. The score is obtained by taking the negative of the highest softmax probability output by the model. The direction of increasing or decreasing values of these metrics signifying greater robustness will be indicated in the table with an arrow.

A.2 Robustness under Adversarial Attacks

Attacks: We evaluate the robustness of our method against adversarial attacks using CleverHans [59] and AutoAttack [18]. All attacks are executed on ImageNet-1K's validation set, and each model is evaluated on the whole set. In particular, we use untargeted, white box attacks such as PGD, FGSM, SLD and CW. In addition, we provide results on gradient-free black box attack, SPSA, diverse AutoAttack, in the standard setting and a simple noise-adding attack. AutoAttack consists of untargeted APGD-CE, targeted APGD-DLR, targeted Fast Adaptive Boundary (FAB) and Square Attack. For further justification of the benefit of our method, we evaluate a variant of our model, RPC-SymViT (6iter/layer1) solely on the black-box Square Attack and show that we are robust against black box attacks as well. This is provided in Appendix E.4. We use a perturbation budget of $\epsilon = 1/255$ with the l_∞ norm to manipulate the images and evaluate each model with an incremental increase in perturbation for PGD and FGSM. In SPSA, we run 40 steps per attack with a perturbation budget of $\epsilon = 0.1$. PGD attack uses a step size of $\alpha = 0.15$ for 20 steps, where at each step the image is adjusted slightly to maximize the model's loss. Similarly, FGSM does this for a single step. For each attack, we report the top-1 and top-5 accuracy of the model on the distorted dataset with the maximum perturbation budget in Table 2 and across all different perturbations in Figure 3.

A.3 ADE20K Image Segmentation

Dataset: The ADE20K dataset includes complex scenes featuring highly detailed labels, making it one of the most challenging segmentation tasks. The training set contains 20,210 images spread across 150 distinct semantic categories. The validation set includes 2,000 images, while the test set comprises 3,352 images. The metrics report for this task are the Mean Accuracy (%) and Mean Intersection-Over-Union (IOU).

A.4 WikiText-103 Language Modeling

Dataset: The WikiText-103 dataset [47] is derived from Wikipedia articles and is designed to capture long-range contextual dependencies. The training set contains about 28,000 articles, with a total of 103 million words. Each article is divided into text blocks with approximately 3,600 words. The validation and test sets have 218,000 and 246,000 words, respectively, with both sets comprising 60 articles and totaling about 268,000 words. For evaluation, we use a batch size of 1 and apply a sliding window of length L to process the text sequences. When calculating perplexity (PPL), we focus only on the final position, except in the first segment, where we evaluate all positions. We corrupt the both validation and test datasets to demonstrate the robustness of RPC-Attention using TextAttack's word swap attack [48] to create the attacked WikiText-103 dataset. This adversarial attack randomly replaces words in the dataset with a generic "AAA" for evaluation making it difficult for the model to predict the next word in the sequence correctly.

B Calculating the Gram Matrix $\widetilde{\boldsymbol{K}}_{\boldsymbol{\varphi}}$

In this section, we will show that the centered Gram matrix $\widetilde{\boldsymbol{K}}_{\boldsymbol{\varphi}}$ can be computed from the uncentered Gram matrix $\boldsymbol{K}_{\boldsymbol{\varphi}}$ with elements $\boldsymbol{K}_{\boldsymbol{\varphi}}(i,j) = k_{\boldsymbol{\varphi}}(\boldsymbol{k}_i, \boldsymbol{k}_j) = \boldsymbol{\varphi}(\boldsymbol{k}_i)^\top \boldsymbol{\varphi}(\boldsymbol{k}_j)$. In particular, $\widetilde{\boldsymbol{K}}_{\boldsymbol{\varphi}} = \boldsymbol{K}_{\boldsymbol{\varphi}} - \mathbf{1}_N \boldsymbol{K}_{\boldsymbol{\varphi}} - \boldsymbol{K}_{\boldsymbol{\varphi}} \mathbf{1}_N + \mathbf{1}_N \boldsymbol{K}_{\boldsymbol{\varphi}} \mathbf{1}_N$, where $\mathbf{1}_N$ denotes the $N \times N$ matrix in which every element takes the value $1/N$. Our centered feature vector has the form

$$\tilde{\boldsymbol{\varphi}}(\boldsymbol{k}_j) = \boldsymbol{\varphi}(\boldsymbol{k}_j) - \frac{1}{N}\sum_{j'=1}^{N} \boldsymbol{\varphi}(\boldsymbol{k}_{j'}).$$

Then the elements of the centered Gram matrix are given as follows

$$\begin{aligned}
\widetilde{\boldsymbol{K}}_{\boldsymbol{\varphi}}(i,j) &= \tilde{k}_{\boldsymbol{\varphi}}(\boldsymbol{k}_i, \boldsymbol{k}_j) \\
&= \tilde{\boldsymbol{\varphi}}(\boldsymbol{k}_i)^\top \tilde{\boldsymbol{\varphi}}(\boldsymbol{k}_j) \\
&= \boldsymbol{\varphi}(\boldsymbol{k}_i)^\top \boldsymbol{\varphi}(\boldsymbol{k}_j) - \frac{1}{N}\sum_{j'=1}^{N} \boldsymbol{\varphi}(\boldsymbol{k}_i)^\top \boldsymbol{\varphi}(\boldsymbol{k}_{j'}) - \frac{1}{N}\sum_{j'=1}^{N} \boldsymbol{\varphi}(\boldsymbol{k}_{j'})^\top \boldsymbol{\varphi}(\boldsymbol{k}_j) + \frac{1}{N^2}\sum_{j'=1}^{N}\sum_{l=1}^{N} \boldsymbol{\varphi}(\boldsymbol{k}_{j'})^\top \boldsymbol{\varphi}(\boldsymbol{k}_l) \\
&= k_{\boldsymbol{\varphi}}(\boldsymbol{k}_i, \boldsymbol{k}_j) - \frac{1}{N}\sum_{j'=1}^{N} k_{\boldsymbol{\varphi}}(\boldsymbol{k}_i, \boldsymbol{k}_{j'}) - \frac{1}{N}\sum_{j'=1}^{N} k_{\boldsymbol{\varphi}}(\boldsymbol{k}_{j'}, \boldsymbol{k}_j) + \frac{1}{N^2}\sum_{j'=1}^{N}\sum_{l=1}^{N} k_{\boldsymbol{\varphi}}(\boldsymbol{k}_{j'}, \boldsymbol{k}_l)
\end{aligned}$$

which expressed in matrix form, gives the result.

C Plotting J_{proj} in Section 2.2.1

In Section 2.2.1, we plotted the reconstruction loss at each epoch of training. Here, we provide the details of the calculation of this loss. From Eqn. (13),

$$\begin{aligned}
L &= \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{\varphi}(\boldsymbol{q}_i) - \sum_{d=1}^{D_v} h_{id} u_d\|^2 \\
&= \frac{1}{N}\sum_{i=1}^{N} (\|\boldsymbol{\varphi}(\boldsymbol{q}_i)\|^2 - \|\boldsymbol{h}_i\|^2).
\end{aligned}$$

In the above, $\boldsymbol{h}_i$ is simply our attention output and $\|\boldsymbol{\varphi}(\boldsymbol{q}_i)\|^2 = \boldsymbol{\varphi}(\boldsymbol{q}_i)^\top \boldsymbol{\varphi}(\boldsymbol{q}_i) = \frac{e^{\boldsymbol{q}_i^\top \boldsymbol{q}_i/\sqrt{D}}}{(\sum_{j=1}^{N} e^{\boldsymbol{q}_i^\top \boldsymbol{k}_j}/\sqrt{D})^2}$ for $i = 1, \dots, N$, can be calculated using the kernel trick as follows. Let $\boldsymbol{A}$ be the softmax attention matrix, $\boldsymbol{a}_1$ its first column and $\boldsymbol{a}_1^2$ denote the element-wise product.

$$log(\boldsymbol{a}_1^2) = log(\begin{bmatrix} \frac{e^{2\boldsymbol{q}_1^\top \boldsymbol{k}_1/\sqrt{D}}}{(\sum_{j=1}^N e^{\boldsymbol{q}_1^\top \boldsymbol{k}_j/\sqrt{D}})^2} \\ \vdots \\ \frac{e^{2\boldsymbol{q}_N^\top \boldsymbol{k}_1/\sqrt{D}}}{(\sum_{j=1}^N e^{\boldsymbol{q}_N^\top \boldsymbol{k}_j/\sqrt{D}})^2} \end{bmatrix})$$

$$= log(\begin{bmatrix} e^{2\boldsymbol{q}_1^\top \boldsymbol{k}_1/\sqrt{D}} \\ \vdots \\ e^{2\boldsymbol{q}_N^\top \boldsymbol{k}_1/\sqrt{D}} \end{bmatrix}) - log(\begin{bmatrix} (\sum_{j=1}^N e^{\boldsymbol{q}_1^\top \boldsymbol{k}_j/\sqrt{D}})^2 \\ \vdots \\ (\sum_{j=1}^N e^{\boldsymbol{q}_N^\top \boldsymbol{k}_j/\sqrt{D}})^2 \end{bmatrix})$$

$$\implies log(\begin{bmatrix} (\sum_{j=1}^N e^{\boldsymbol{q}_1^\top \boldsymbol{k}_j/\sqrt{D}})^2 \\ \vdots \\ (\sum_{j=1}^N e^{\boldsymbol{q}_N^\top \boldsymbol{k}_j/\sqrt{D}})^2 \end{bmatrix}) = log(\begin{bmatrix} e^{2\boldsymbol{q}_1^\top \boldsymbol{k}_1/\sqrt{D}} \\ \vdots \\ e^{2\boldsymbol{q}_N^\top \boldsymbol{k}_1/\sqrt{D}} \end{bmatrix}) - log(\boldsymbol{a}_1^2)$$

$$= \begin{bmatrix} 2\boldsymbol{q}_1^\top \boldsymbol{k}_1/\sqrt{D} \\ \vdots \\ 2\boldsymbol{q}_N^\top \boldsymbol{k}_1/\sqrt{D} \end{bmatrix} - log(\boldsymbol{a}_1^2)$$

$$\implies \begin{bmatrix} (\sum_{j=1}^N e^{\boldsymbol{q}_1^\top \boldsymbol{k}_j/\sqrt{D}})^2 \\ \vdots \\ (\sum_{j=1}^N e^{\boldsymbol{q}_N^\top \boldsymbol{k}_j/\sqrt{D}})^2 \end{bmatrix} = e^{\begin{bmatrix} 2\boldsymbol{q}_1^\top \boldsymbol{k}_1/\sqrt{D} \\ \vdots \\ 2\boldsymbol{q}_N^\top \boldsymbol{k}_1/\sqrt{D} \end{bmatrix} - log(\boldsymbol{a}_1^2)}$$

Then,

$$\begin{bmatrix} \|\boldsymbol{\varphi}(\boldsymbol{q}_1)\|^2 \\ \vdots \\ \|\boldsymbol{\varphi}(\boldsymbol{q}_N)\|^2 \end{bmatrix} = \begin{bmatrix} e^{\boldsymbol{q}_1^\top \boldsymbol{q}_1/\sqrt{D}} \\ \vdots \\ e^{\boldsymbol{q}_N^\top \boldsymbol{q}_N/\sqrt{D}} \end{bmatrix} / e^{\begin{bmatrix} 2\boldsymbol{q}_1^\top \boldsymbol{k}_1/\sqrt{D} \\ \vdots \\ 2\boldsymbol{q}_N^\top \boldsymbol{k}_1/\sqrt{D} \end{bmatrix} - log(\boldsymbol{a}_1^2)}$$

D Principal Component Pursuit

Let $\boldsymbol{M}, \boldsymbol{L}, \boldsymbol{S} \in \mathbb{R}^{N \times D}$ be the matrix of our corrupted measurements, the low-rank matrix we seek to recover and a sparse corruption matrix respectively. The optimization problem we would like to solve is

$$\begin{array}{ll} \text{minimize}_{\boldsymbol{L},\boldsymbol{S}} & \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 \\ \text{subject to} & \boldsymbol{L} + \boldsymbol{S} = \boldsymbol{M} \end{array} \tag{15}$$

Under minimal assumptions that the low-rank component $\boldsymbol{L}$ is not sparse (i.e., $\boldsymbol{L}$ satisfies the incoherence condition defined in [11]), and the sparse component $\boldsymbol{S}$ is not low-rank (i.e., the sparsity pattern of S is selected uniformly at random), we will follow the author's choice of algorithm to solve the PCP which is to use the Alternating Direction Method of Multipliers (ADMM).

The ADMM algorithm uses the augmented Lagrangian,

$$l(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{Y}) = \|\boldsymbol{L}\|_* + \lambda\|\boldsymbol{S}\|_1 + \langle \boldsymbol{Y}, \boldsymbol{M} - \boldsymbol{L} - \boldsymbol{S}\rangle + \mu/2\|\boldsymbol{M} - \boldsymbol{L} - \boldsymbol{S}\|_F^2$$

and solves a sequence of optimization problems. We iterate through setting $\boldsymbol{S}_{k+1} = \arg\min_{\boldsymbol{S}} l(\boldsymbol{L}_k, \boldsymbol{S}, \boldsymbol{Y}_k)$ and $\boldsymbol{L}_{k+1} = \arg\min_{\boldsymbol{L}} l(\boldsymbol{L}, \boldsymbol{S}_{k+1}, \boldsymbol{Y}_k)$ before updating the Langrange multiplier matrix $\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \mu(\boldsymbol{M} - \boldsymbol{L}_{k+1} - \boldsymbol{S}_{k+1})$. The advantage of the algorithm is that it turns a complicated optimisation problem into sub problems that have straightforward and efficient solutions. Without much difficulty we can show that:

$$\arg\min_{\boldsymbol{S}} l(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{Y}) = \mathcal{S}_{\lambda/\mu}(\boldsymbol{M} - \boldsymbol{L} + \mu^{-1}\boldsymbol{Y})$$
$$\arg\min_{\boldsymbol{L}} l(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{Y}) = \mathcal{D}_{\mu}(\boldsymbol{M} - \boldsymbol{S} - \mu^{-1}\boldsymbol{Y})$$

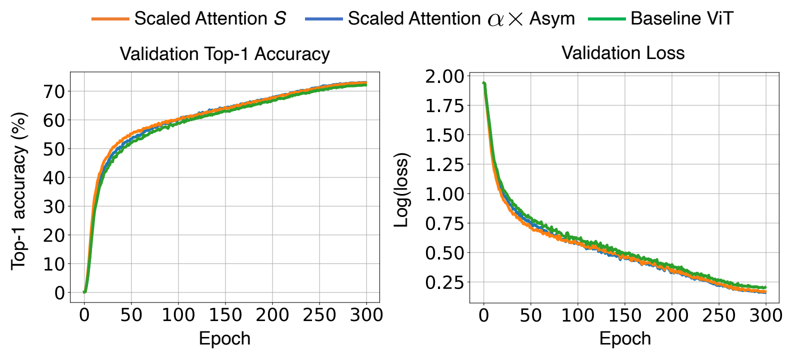

Figure 4: Plot of the validation top-1 accuracy (%) and loss on a log scale of the baseline asymmetric attention ViT and two variants with the parameterization of Remark. 3. The curves are plotted for the full training time and show $\boldsymbol{S}$ trained as a matrix parameter as well as a scalar parameter scaling a symmetric attention matrix.

where $\mathcal{S}_\tau(x) = \operatorname{sgn}(x)\max(|x| - \tau, 0)$ is the shrinkage operator, extended to matrices by applying it element-wise and $\mathcal{D}_\tau(\boldsymbol{X}) = \boldsymbol{U}\mathcal{S}_\tau(\Sigma)\boldsymbol{V}^*$ is the singular value thresholding operator with the singular value decomposition of $\boldsymbol{X} = \boldsymbol{U}\Sigma\boldsymbol{V}^*$.

The algorithm is summarised as below

Algorithm 2 ADMM for Principal Components Pursuit

```
initialize: $\boldsymbol{S}_0 = \boldsymbol{Y}_0 = \boldsymbol{0}$; $\mu, \lambda > 0$.
while not converged do
    compute $\boldsymbol{S}_{k+1} = \mathcal{S}_{\lambda/\mu}(\boldsymbol{M} - \boldsymbol{L}_k + \mu^{-1}\boldsymbol{Y}_k)$;
    compute $\boldsymbol{L}_{k+1} = \mathcal{D}_\mu(\boldsymbol{M} - \boldsymbol{S}_{k+1} - \mu^{-1}\boldsymbol{Y}_k)$;
    compute $\boldsymbol{Y}_{k+1} = \boldsymbol{Y}_k + \mu(\boldsymbol{M} - \boldsymbol{L}_{k+1} - \boldsymbol{S}_{k+1})$;
end while
output: $\boldsymbol{L}$, $\boldsymbol{S}$.
```

E Additional Experimental Results

E.1 Reparameterization of Self-Attention

Let $\boldsymbol{A}_{sym} = [a_{ij}]$ be the softmax attention matrix. Then, from Remark 3, $\boldsymbol{S}$ has the following form and we multiply the numerator and denominator by $1/K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})$ to obtain a much more convenient expression.

$$
\begin{aligned}
s_{j'j} =& \frac{g(\boldsymbol{k}_j)}{g(\boldsymbol{k}_{j'})} \\
=& \frac{g(\boldsymbol{k}_j)}{g(\boldsymbol{k}_{j'})} \times \frac{1/K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})}{1/K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})} \\
=& \frac{g(\boldsymbol{k}_j)}{K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})} \div \frac{g(\boldsymbol{k}_{j'})}{K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})} \\
=& \frac{g(\boldsymbol{k}_j)}{K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})} \times \frac{K(\boldsymbol{k}_j, \boldsymbol{k}_{j'})}{g(\boldsymbol{k}_{j'})} \\
=& \frac{a_{j'j}}{a_{jj'}} \\
\implies \boldsymbol{S} =& \frac{1}{N}\boldsymbol{A}_{sym} \odot 1/\boldsymbol{A}_{sym}^\top
\end{aligned}
$$

In Fig. 4, we plot the full training curve of the two versions of Scaled Attention, Scaled Attention S and Scaled Attention $\alpha \times$ Asym. We observe that the parameterized models with Scaled Attention do converge more quickly and even obtain a slightly higher validation top-1 accuracy of 73.02% for the scalar variant as compared to the standard asymmetric ViT at 72.18%. The final validation loss is also lower at 1.18 and 1.23 respectively.

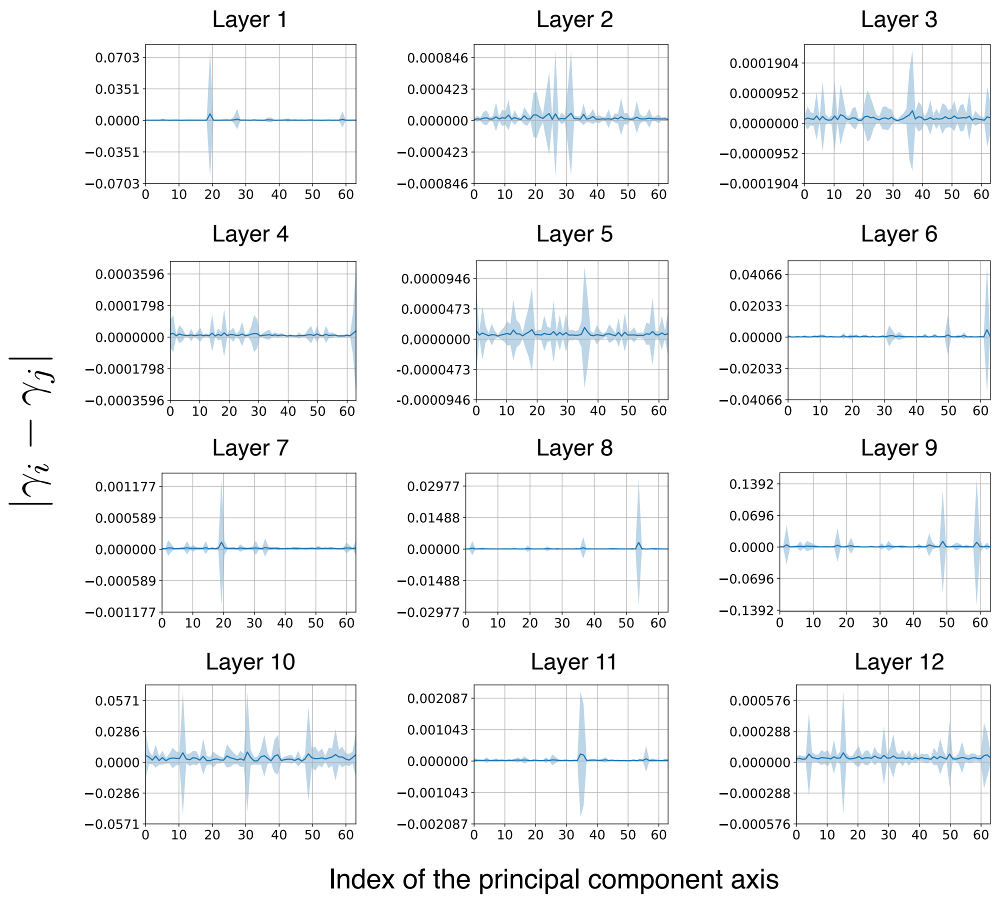

Figure 5: Plot of the mean and standard deviation of the differences in coordinate values of constant vector $\mathbf{1}\lambda_d$ for $d = 1, \dots, D_v$ for all 12 layers of a ViT-tiny model. The mean should be 0 with small standard deviations when $v_{dj} \approx \frac{a_{dj}}{g(\boldsymbol{k}_j)} - \frac{1}{N}\sum_{j'=1}^{N} \frac{a_{dj'}}{g(\boldsymbol{k}_j)}$.

E.2 Pairwise Absolute Differences of γ_i and γ_j

In Figure 5, we provide the plot of the absolute differences of the coordinates of $\frac{\widetilde{\boldsymbol{K}_{\varphi}}\hat{\boldsymbol{a}}_d}{N\hat{\boldsymbol{a}}_d}$ in all 12 layers of a ViT-tiny model as elaborated in Section 2.2.2. This would be a constant vector within our framework and the plots provide empirical evidence to justify our claim. In all of the layers, the means are noticeably close to 0 and the standard deviations are also small suggesting that we indeed recovered a constant vector.

E.3 ImageNet-C Results by Corruption Type

In Table 4, we provide the full results of various RPC-SymViT models against the standard SymViT on ImageNet-C for all corruption types averaged over the 5 severity levels. In all types, except for Zoom Blur, the RPC-SymViT model with 6 iterations in the 1st layer outperforms SymViT.

E.4 Square Attack

Square attack [3] is a score-based black box attack that does not use local gradient information, hence it is not affected by gradient masking. It operates based on a random search scheme that modifies square-shaped regions such that at each iteration, the perturbation lies approximately at the boundary of the feasible set. We evaluate our RPC-SymViT(6iter/layer1) variant on square attacked ImageNet-1K validation set and compare it to the baseline. Our result of the top-1 and top-5 accuracy in Table 5 illustrates that the effectiveness of RPC-Attention against adversarial attacks is not solely due to a form gradient masking as we still significantly outperform the baseline on square.

Table 4: Top-1 accuracy (%) and mean corruption error (mCE) of all RPC-SymViT variants and SymViT on each corruption type in ImageNet-C. RPC-SymViT ($n$iter/layer1) applies n PAP iterations only at the first layer. RPC-SymViT ($n$iter/all-layer) applies n PAP iterations at all layers.

Corruption Type	Model/ Metric	SymViT	RPC-SymViT (2iter/all-layer)	RPC-SymViT (4iter/layer1)	RPC-SymViT (5iter/layer1)	RPC-SymViT (6iter/layer1)
Brightness	Top-1 ↑	63.21	62.97	63.74	64.19	64.31
	mCE ↓	65.16	65.69	64.22	63.43	63.22
Contrast	Top-1 ↑	50.25	49.59	49.30	49.84	50.18
	mCE ↓	58.53	59.08	59.42	58.79	58.39
Defocus Blur	Top-1 ↑	35.47	35.61	35.54	35.26	36.38
	mCE ↓	78.71	78.53	78.62	78.96	77.59
Elastic Transform	Top-1 ↑	44.26	44.52	44.68	44.84	44.72
	mCE ↓	86.28	85.87	85.63	85.38	85.56
Fog	Top-1 ↑	42.72	45.49	46.03	46.59	46.40
	mCE ↓	69.91	66.53	65.87	65.19	65.42
Frost	Top-1 ↑	44.55	45.06	45.93	46.24	46.27
	mCE ↓	67.08	66.46	65.41	65.04	65.01
Gaussian Noise	Top-1 ↑	40.97	41.64	42.22	41.67	42.51
	mCE ↓	66.59	65.84	65.19	65.80	64.85
Glass Blur	Top-1 ↑	27.79	28.12	28.48	27.87	28.05
	mCE ↓	87.39	87.00	86.56	87.30	87.07
Impulse Noise	Top-1 ↑	38.56	38.98	40.24	39.46	40.33
	mCE ↓	66.59	66.14	64.77	65.61	64.67
JPEG Compression	Top-1 ↑	46.28	46.23	47.92	48.41	48.63
	mCE ↓	88.58	88.65	85.87	85.06	84.70
Motion Blur	Top-1 ↑	40.47	40.00	41.72	41.81	42.22
	mCE ↓	75.75	76.43	74.16	74.04	73.51
Pixelate	Top-1 ↑	39.36	39.81	41.13	41.86	42.25
	mCE ↓	84.47	83.84	82.01	81.00	80.45
Shot Noise	Top-1 ↑	38.83	39.20	39.52	39.03	39.71
	mCE ↓	68.38	67.98	67.61	68.16	67.40
Snow	Top-1 ↑	38.27	37.87	38.75	39.12	38.90
	mCE ↓	71.22	71.67	70.66	70.23	70.48
Zoom Blur	Top-1 ↑	30.69	29.49	29.97	30.20	30.48
	mCE ↓	86.82	88.32	87.72	87.43	87.08

Table 5: Top-1 and Top-5 accuracy (%) of RPC-SymViT(6iter/layer1) and baseline SymViT on square attacked ImageNet-1K validation data. RPC-SymViT ($n$iter/layer1) applies n PAP iterations only at the first layer.

Model	Square attack	
	Top-1 ↑	Top-5 ↑
SymViT (baseline)	41.50	79.79
RPC-SymViT (6iter/layer1)	**43.64**	**80.81**

E.5 Results on ADE20K Image Segmentation

We further evaluate the benefits of our method by implementing RPC-Attention in a Segmenter model [72] and providing results on the ADE20K image segmentation task [95] in Table 6. We report the Mean Accuracy and Mean Intersection-Over-Union (IOU) for a Segmenter with a RPC-SymViT backbone and compare it against a baseline SymViT backbone. To assess the robustness of each model, we corrupt the ADE20K dataset with the same 15 corruption types in ImageNet-C and report the same metrics averaged over all corruption types in Table 6 as well. Details of our implementation can be found in Appendix A.3. On both datasets, RPC-SymViT outperforms the baseline substantially.

E.6 Results on RPC-SymViT-base

To show that RPC-Attention is not limited to small-scale models, we conduct experiments on a larger model, SymViT-base with 12 transformer layers, 12 heads per layer, and a hidden dimension of 768. We train a SymViT-base with RPC-Attention in all layers using 2 iterations on ImageNet-1K for

Table 6: Mean accuracy (%) and mean Intersection-Over-Union (IOU) of RPC-SymViT and SymViT on clean ADE20K and corrupted ADE20K dataset. RPC-SymViT ($n$iter/layer1) applies n PAP iterations only at the first layer. RPC-SymViT ($n$iter/all-layer) applies n PAP iterations at all layers.

Model	ADE20K		Corrupted ADE20K	
	Mean Acc. ↑	Mean IOU ↑	Mean Acc. ↑	Mean IOU ↑
SymViT	44.27	34.00	14.85	10.47
RPC-SymViT (4iter/layer1)	**45.61**	**34.69**	16.06	11.47
RPC-SymViT (5iter/layer1)	45.51	34.63	16.15	11.35
RPC-SymViT (6iter/layer1)	45.27	34.24	**16.44**	**11.60**
RPC-SymViT (2iter/all-layer)	43.61	33.5	15.04	10.64

Table 7: Top-1, top-5 accuracy (%) , mean corruption error (mCE), and area under the precision-recall curve (AUPR) of RPC-SymViT-base and SymViT-base on clean ImageNet-1K data and popular standard robustness benchmarks for image classification. RPC-SymViT-base ($n$iter/all-layer) applies n PAP iterations at all layers.

Model	IN-1K		IN-A	IN-C		IN-O
	Top-1 ↑	Top-5 ↑	Top-1 ↑	Top-1 ↑	mCE ↓	AUPR ↑
SymViT-base (baseline)	80.62	94.78	24.03	58.88	52.57	23.96
RPC-SymViT-base (2iter/all-layer)	**80.72**	**94.82**	**24.97**	**59.29**	**52.00**	**26.01**

Table 8: Top-1 and top-5 accuracy (%) of RPC-SymViT-base and SymViT-base on PGD and FGSM attacked ImageNet-1K validation data with the highest perturbation budget. RPC-SymViT-base ($n$iter/all-layer) applies n PAP iterations at all layers.

Model	PGD		FGSM	
	Top-1 ↑	Top-5 ↑	Top-1 ↑	Top-5 ↑
SymViT-base (baseline)	12.11	25.55	53.61	78.05
RPC-SymViT-base (2iter/all-layer)	**13.41**	**26.68**	**54.68**	**78.06**

300 epochs. We refer to this model as RPC-SymViT-base (2iter/all-layer). We compare our RPC-SymViT-base with the baseline on the same standard robustness benchmarks as before, ImageNet-C, ImageNet-A, and ImageNet-O, as well as on white box attacks PGD and FGSM with the highest perturbation budgets. These results are in Table 7 and 8 respectively and show that RPC-Attention is also effective in SymViT-base.

E.7 Results on RPC-Attention in FAN

To validate that our RPC-Attention is also complementary with SOTA transformer models and can be combined with those methods to improve the SOTA results, we have conducted additional experiments in which we incorporate our RPC-Attention with FAN [96]. FAN is one of the top transformer models that obtain SOTA results on accuracy and robustness. A FAN model augments the MLP layer that follows the standard self-attention with a new channel attention (CA) block. This CA computes an attention matrix along the channel dimension, taking advantage of the feature covariance and allowing the model to filter out irrelevant information.

We use the FAN-ViT-tiny (FAN-tiny) variant with a symmetric attention for training. In our RPC-Attention + FAN (RPC-FAN-tiny), we replace the attention blocks in the first layer of FAN with our RPC-Attention that runs 4 PAP iterations with hyperparameter values of $\lambda = 4$ and $\mu = ND/4\|\boldsymbol{K}\|_1$. Both our RPC-FAN-tiny and the baseline FAN-tiny are trained for 300 epochs on the ImageNet-1K object classification task. We summarize our results in Tables 9 and 10. RPC-FAN-tiny outperforms the baseline FAN-tiny on all evaluated benchmarks, including ImageNet-1k, ImageNet-R, and ImageNet-A. Additionally, on PGD and FGSM attacked data, RPC-FAN-tiny significantly outperforms FAN-tiny by over 3%.

E.8 Results on downstream Natural Language Understanding tasks

We conduct additional experiments on several downstream Natural Language Understanding tasks to illustrate the effectiveness of RPC-Attention during fine-tuning as well. The Stanford Sentiment

Table 9: Top-1, top-5 accuracy (%) of RPC-FAN-tiny and FAN-tiny on clean ImageNet-1K data and popular standard robustness benchmarks for image classification. RPC-FAN-tiny ($n$iter/layer1) applies n PAP iterations only at the first layer.

Model	IN-1K		IN-R	IN-A
	Top-1 ↑	Top-5 ↑	Top-1 ↑	Top-1 ↑
FAN-tiny (baseline)	77.89	94.20	41.79	13.40
RPC-FAN-tiny (4iter/layer1)	**77.98**	**94.27**	**42.02**	**13.55**

Table 10: Top-1 and top-5 accuracy (%) of RPC-FAN-tiny and FAN-tiny on PGD and FGSM attacked ImageNet-1K validation data with the highest perturbation budget. RPC-FAN-ViT ($n$iter/layer1) applies n PAP iterations only at the first layer.

Model	PGD		FGSM	
	Top-1 ↑	Top-5 ↑	Top-1 ↑	Top-5 ↑
FAN-tiny (baseline)	2.92	4.86	32.01	61.72
RPC-FAN-tiny (4iter/layer1)	**6.25**	**10.01**	**35.12**	**63.40**

Table 11: Train and validation accuracy (%) of RPC-SymSMoE, SymSMoE, RPC-SMoE (asymmetric) and SMoE (asymmetric) on SST-2 and IMDB downstream tasks when pre-trained on WikiText-103. RPC is only implemented during fine-tuning, not pre-training and runs for 2 iterations for all layers in each model.

Model	SST-2		IMDB	
	Train Acc ↑	Valid Acc ↑	Train Acc ↑	Valid Acc ↑
SymSMoE (baseline)	63.12	69.20	-	-
RPC-SymSMoE	**74.48**	**76.27**	-	-
SMoE (baseline)	53.20	69.38	64.12	65.74
RPC-SMoE	**58.87**	**70.63**	**74.36**	**73.96**

Table 12: Validation and test accuracy of RPC-Attention implemented in a pre-trained transformer language model (RPC-LM) during fine-tuning versus the baseline transformer language model (Baseline-LM). We fine-tune both models on the 5-class Stanford Sentiment Treebank (SST-5) task with 2 iterations for all layers in the RPC model.

Model	Valid Acc ↑	Test Acc ↑
Baseline-LM	46.51	49.23
RPC-LM	**48.68**	**50.36**

Treebank v2 (SST-2) and IMDB Sentiment Analysis (IMDB) tasks are binary classifications where the goal is to determine if sentences have positive or negative sentiments. We use 2 baseline Sparse Mixture of Experts (SMoE) models, one with symmetric attention (SymSMoE) and one with asymmetric attention (SMoE), pre-trained on WikiText-103 without RPC, then fine-tuned with (RPC-SymSMoE/RPC-SMoE) and without RPC-Attention for comparison. Their results can be found in Table 11. We observe that RPC-models outperform the baseline models significantly on these tasks.

On the 5-class sentiment classification task, Stanford Sentiment Treebank (SST-5), we use a pre-trained transformer language model from the NAACL 2019 tutorial on "Transfer Learning in Natural Language Processing" for fine-tuning. Their publicly available code can be found at https://github.com/huggingface/naacl_transfer_learning_tutorial. The objective of the task is to determine if the sentences have negative, somewhat negative, neutral, somewhat positive, or positive sentiments. We implement RPC-Attention during fine-tuning only (RPC-LM) and compare the results with the baseline model (Baseline-LM) on SST-5 in Table 12. As can be seen from the table, our RPC-Attention is applicable to pre-trained language models and performs significantly better than the baseline. Hence, RPC-Attention is highly effective in downstream natural language understanding tasks and versatile in its application.

Table 13: Flop per sample, run time per sample, memory and number of parameters of each RPC-SymViT variant compared to the SymViT baseline. RPC-SymViT ($n$iter/layer1) applies n PAP iterations only at the first layer. RPC-SymViT ($n$iter/all-layer) applies n PAP iterations at all layers.

Model	Flop/Sample	Sec/Sample (Training)	Sec/Sample (Test)	Memory (Training)	Memory (Test)	Parameters
SymViT (baseline)	1.17M	0.079	0.010	1435MB	1181MB	5.2M
RPC-SymViT (4iter/layer1)	1.22M	0.082	0.010	1441MB	1181MB	5.2M
RPC-SymViT (5iter/layer1)	1.23M	0.084	0.010	1443MB	1181MB	5.2M
RPC-SymViT (6iter/layer1)	1.25M	0.085	0.011	1443MB	1181MB	5.2M
RPC-SymViT (2iter/layer)	1.35M	0.092	0.017	1461MB	1181MB	5.2M

Table 14: Top-1, top-5 accuracy (%) and AUPR of an implementation of RPC-Attention on asymmetric attention evaluated on ImagetNet-1K validation set, ImageNet-R, ImageNet-A and ImageNet-O. These results are compared to the standard asymmetric ViT.

Model	IN-1K		IN-R	IN-A	IN-O
	Top-1 ↑	Top-5 ↑	Top-1 ↑	Top-1 ↑	AUPR ↑
ViT	72.11	90.97	32.41	7.65	17.36
RPC-AsymViT (4iter/layer1)	72.34	91.12	32.23	7.75	17.54
RPC-AsymViT (Sym/Asym)	72.34	91.38	32.79	7.23	17.58

E.9 Computational Efficiency

A possible limitation of introducing an iterative scheme into a deep model is a significant increase in computational overhead. We aim to alleviate that concern by providing the number of flops per sample, run time per sample, memory and number of parameters of each RPC-SymViT variant and the SymViT baseline during both training and test time in Table 13. We observe that RPC-Attention is comparable to the baseline softmax attention across all metrics at test time while only slightly less efficient than the baseline in terms of the number of flops, run time per sample, and memory during training.

E.10 Results on Robust Asymmetric Attention

In this section, we report the results of extending the RPC-SymViT model to the asymmetric attention. However, as the PAP Algorithm. 1 is not designed for multiple data matrices, it is not as effective in the asymmetric case. We implement two variations of the algorithm in an asymmetric attention ViT-tiny with 12 layers. In the 4iter/layer1 version, similar to the symmetric attention case, we run 4 iterations of the algorithm only in the 1st layer of the model by replacing $\mathrm{Softmax}(\boldsymbol{K}-\boldsymbol{S}_{k+1}-\mu^{-1}\boldsymbol{Y}_k, \boldsymbol{K}-\boldsymbol{S}_{k+1}-\mu^{-1}\boldsymbol{Y}_k)$ with $\mathrm{Softmax}(\boldsymbol{Q}, \boldsymbol{K}-\boldsymbol{S}_{k+1}-\mu^{-1}\boldsymbol{Y}_k)$. For the second version, labeled Sym/Asym, we run the algorithm for 4 iterations in a symmetric attention mechanism in the 1st layer, followed by the standard asymmetric attention in layers 2 to 12. We compare these RPC-AsymViT models to the asymmetric softmax attention ViT.

As we can see from Table 14, both variants only improve slightly over the benchmark on most of the corrupted datasets. Such a result confirms our intuition that the ADMM algorithm does not extend easily to multiple corrupted matrices as it only solves an objective function involving a single low-rank data matrix and its sparse corruption matrix.

F Further Discussion on Related Works

[15] provides a new perspective by emphasizing the asymmetry of the softmax kernel and recovers the self-attention mechanism from an asymmetric Kernel Singular Value Decomposition (KSVD) using the duality of the optimization problem. Separately, our kernel PCA perspective derives softmax attention and addresses the asymmetry of attention through a projection of the query vectors in feature space. While there are similarities between KSVD and kernel PCA, in the sense of finding low rank approximations and low dimensional representations of the data, the primal objective functions are different and we do not need to consider the dual form. Another related work views transformers from the perspective of Support Vector Machines [76]. Though, kernel PCA can be formulated in a similar fashion as a least squares SVM problem as explained in [74], our work focuses on the forward

pass of attention and show that it can recover a kernel PCA solution. In contrast, [74] examines the backward pass and optimization geometry of an attention layer towards a hard margin SVM solution that separates optimal tokens from non-optimal ones, under certain assumptions of the loss function, initial conditions, and certain SVM constraints. Furthermore, using our framework, we are able to predict the exact explicit form of the value matrix in self-attention, demonstrating that this matrix captures the eigenvectors of the Gram matrix of the key vectors in a feature space. To the best of our knowledge, our work is the first to show this insight.

Hamburger in [25] models the global context discovery as the low-rank recovery of the input tensor and solves it via matrix decomposition. Both Hamburger and our Attention with Robust Principal Components (RPC-Attention) try to recover clean signal subspaces via computing a low-rank approximation of a given matrix. The key differences between our RPC-Attention and Hamburger are: (1) Our RPC-Attention finds a low-rank approximation of the key matrix while Hamburger finds a low-rank approximation of the input matrix, and (2) Our RPC-Attention models the corruption by a sparse matrix while Hamburger does not enforce this condition. The entries of this sparse corruption can have an arbitrarily large magnitude and help model grossly corrupted observations in which only a portion of the observation vector is contaminated by gross error. Numerous critical applications exist where the data being examined can be naturally represented as a combination of a low-rank matrix and a sparse contribution, such as video surveillance, face recognition, and collaborative filtering [11].

[89] derives each component in a Graph Neural Network (GNN) from the unfolded iterations of robust descent algorithms applied to minimizing a principled graph regularized energy function. In particular, propagation layers and nonlinear activations implement proximal gradient updates, and graph attention results from iterative reweighted least squares (IRLS). While this is an interesting approach, it has not been extended to explaining the architecture of transformers, including self-attention, yet. In contrast, our kernel principal component analysis (kernel PCA) allows us to derive self-attention in transformers, showing that the attention outputs are projections of the query vectors onto the principal components axes of the key matrix in a feature space.

[2] and [6] implement each layer as an optimization and fixed-point solver, respectively. In particular, an OptNet layer in [2] solves a quadratic program, and a Deep Equilibrium layer in [6] computes the fixed point of a nonlinear transformation. Different from these layers, our RPC-Attention solves a Principal Component Pursuit - a convex program. Also, both OptNet layer in [2] and Deep Equilibrium layer in [6] do not shed light on the derivation and formulation of self-attention, which our kernel PCA framework does.

G Broader Impacts

Our research improves both clean data processing and robust performance, especially in areas with significant social relevance. Specifically, we demonstrate enhanced results in image segmentation, benefiting self-driving cars, and language modeling, enhancing AI chatbot assistants. We show notable improvements in resisting adversarial attacks, aiming to protect crucial AI systems from malicious actors. Additionally, we achieve competitive performance in language modeling with contaminated data, which is often encountered in real-world situations. Despite the potential for AI misuse, our research presents substantial advancements in fundamental architectures and theory, which we hope will inspire further socially beneficial developments.