A Connection sampling distributions

In this section we provide a more detailed discussion on the distributions which were considered for sampling the subset of inactive connections.

A.1 Gradient magnitude upper bound (GraBo)

The gradient magnitude for a single connection $(a, b) \in \mathbb{W}$ can be expressed as follows:

$$\left|\nabla\theta^{[l]}_{a,b}\right| = \left|\sum_{i=1}^{B} \boldsymbol{h}^{[l-1]}_{a,i}\boldsymbol{\delta}^{[l]}_{b,i}\right| \implies \left|\nabla\theta^{[l]}\right| = \left|\boldsymbol{h}^{[l-1]}\left(\boldsymbol{\delta}^{[l]}\right)^{\mathsf{T}}\right| \tag{1}$$

where $\boldsymbol{h}^{[l]}$ and $\boldsymbol{\delta}^{[l]}$ are the activations and gradients of the loss at the output units of the l-th layer, respectively. When the batch size B is one, the sum disappears, simplifying the gradient magnitude to a rank one matrix which can be used to sample connections efficiently, inducing a joint probability distribution that is proportional to the gradient magnitude only when the batch size is one. In practice, however, training samples come in mini batches to reduce the variance of the gradient. Therefore, we experiment with sampling the connections proportionally to the following upper bound of the gradient magnitude instead, which enables efficient sampling, even with mini batches:

$$\left|\nabla\theta^{[l]}\right| = \left|\boldsymbol{h}^{[l-1]}\left(\boldsymbol{\delta}^{[l]}\right)^{\mathsf{T}}\right| \leq \left(\left|\boldsymbol{h}^{[l-1]}\right|\mathbf{1}\right)\left(\left|\boldsymbol{\delta}^{[l]}\right|\mathbf{1}\right)^{\mathsf{T}} \tag{2}$$

The proof for Equation 2 uses the triangle inequality and is provided in Appendix B. This upper bound has the property that it becomes an equality when $B = 1$. Connections are then sampled from the probability distributions in Equation 3. The implementation of this distribution does not require any modifications to the model, and only needs a single forward and backward pass to evaluate.

$$\mathrm{f}^{[l]} \coloneqq \frac{\left|\boldsymbol{h}^{[l-1]}\right|\mathbf{1}}{\mathbf{1}^{\mathsf{T}}\left|\boldsymbol{h}^{[l-1]}\right|\mathbf{1}}, \quad \mathrm{g}^{[l]} \coloneqq \frac{\left|\boldsymbol{\delta}^{[l]}\right|\mathbf{1}}{\mathbf{1}^{\mathsf{T}}\left|\boldsymbol{\delta}^{[l]}\right|\mathbf{1}} \tag{3}$$

A.2 Gradient magnitude estimate (GraEst)

The reason we use a bound on the absolute gradient is that in general the sum of products cannot be written as the product of sums, that is, $\sum_i x_i y_i \neq (\sum_i x_i)(\sum_i y_i)$. However, it is possible to satisfy this equality in expectation. This was shown by Alon et al. (1996) who used it to estimate the second frequency moment, or the L^2 norm of a vector. Their work is called the AMS sketch, AMS is an initialism derived from the last names of the authors. They then showed that this also works in the more general case to estimate inner products between vectors (Alon et al., 1999). This is achieved by assigning each index of the vector a random sign, i.e. $s_i \sim \mathcal{U}\{-1, +1\}$, resulting in the following equality:

$$\sum_i x_i y_i = \mathbb{E}\left[\left(\sum_i s_i x_i\right)\left(\sum_i s_i y_i\right)\right] \tag{4}$$

We can use their finding to construct probability distributions $\mathrm{f}^{[l]}$ and $\mathrm{g}^{[l]}$ in Equation 5 such that their induced joint probability distribution is proportional to the gradient magnitude in expectation.

This distribution can also be computed in a single forward and backward pass and requires no modifications to the model, apart from introducing a mechanism to sample the random signs $\boldsymbol{s}$.

$$\mathrm{f}^{[l]} \coloneqq \frac{\left|\boldsymbol{h}^{[l-1]}\boldsymbol{s}\right|}{\mathbf{1}^\mathsf{T}\left|\boldsymbol{h}^{[l-1]}\boldsymbol{s}\right|}, \quad \mathrm{g}^{[l]} \coloneqq \frac{\left|\boldsymbol{\delta}^{[l]}\boldsymbol{s}\right|}{\mathbf{1}^\mathsf{T}\left|\boldsymbol{\delta}^{[l]}\boldsymbol{s}\right|} \tag{5}$$

B Gradient magnitude upper bound proof

Lemma B.1. *The gradient magnitude of the loss with respect to the parameters of a fully-connected layer l has the following upper bound:*

$$\left|\nabla\theta^{[l]}\right| \leq \left(\left|\boldsymbol{h}^{[l-1]}\right|\mathbf{1}\right)\left(\left|\boldsymbol{\delta}^{[l]}\right|\mathbf{1}\right)^\mathsf{T}$$

where $\theta^{[l]} \in \mathbb{R}^{n^{[l-1]}\times n^{[l]}}$, $\boldsymbol{h}^{[l]} \in \mathbb{R}^{n^{[l]}\times B}$, and $\boldsymbol{\delta}^{[l]} \in \mathbb{R}^{n^{[l]}\times B}$ are the weight matrix, activations and gradients of the loss at the output units of the l-th layer, respectively. The number of units in the l-th layer is denoted by $n^{[l]} \in \mathbb{N}^+$, and $B \in \mathbb{N}^+$ is the batch size.

Proof. The proof starts with the definition of the gradient of the fully-connected layer, which is calculated during back propagation as the activations of the previous layer multiplied with the back propagated gradient of the output units of the layer. We then write this definition for a single connection in the weight matrix and use the triangle inequality to specify the first upper bound. In Equation 8 we rewrite the upper bound of Equation 7 as all the cross products between the batch dimensions minus all the non-identical cross terms. Note that all the non-identical cross terms are positive because of the absolute, therefore we get another upper bound by removing the non-identical cross terms which we then write in matrix notation in Equation 9.

$$\left|\nabla\theta^{[l]}\right| = \left|\boldsymbol{h}^{[l-1]}\left(\boldsymbol{\delta}^{[l]}\right)^\mathsf{T}\right| \tag{6}$$

$$\implies \left|\nabla\theta^{[l]}_{a,b}\right| = \left|\sum_{i=1}^{B}\boldsymbol{h}^{[l-1]}_{a,i}\boldsymbol{\delta}^{[l]}_{b,i}\right| \leq \sum_{i=1}^{B}\left|\boldsymbol{h}^{[l-1]}_{a,i}\boldsymbol{\delta}^{[l]}_{b,i}\right| \tag{7}$$

$$= \sum_{i}^{B}\left|\boldsymbol{h}^{[l-1]}_{a,i}\right|\sum_{i}^{B}\left|\boldsymbol{\delta}^{[l]}_{b,i}\right| - \sum_{r,s\neq r}^{B}\left|\boldsymbol{h}^{[l-1]}_{a,r}\boldsymbol{\delta}^{[l]}_{b,s}\right| \leq \sum_{i}^{B}\left|\boldsymbol{h}^{[l-1]}_{a,i}\right|\sum_{i}^{B}\left|\boldsymbol{\delta}^{[l]}_{b,i}\right| \tag{8}$$

$$\implies \left|\nabla\theta^{[l]}\right| \leq \left(\left|\boldsymbol{h}^{[l-1]}\right|\mathbf{1}\right)\left(\left|\boldsymbol{\delta}^{[l]}\right|\mathbf{1}\right)^\mathsf{T} \tag{9}$$

□

C Data normalization

We normalize all the training and test data to have zero mean and a standard deviation of one. The dataset statistics that we used are specified below, where μ is the mean and σ the standard deviation. The values correspond to the red, green, and blue color channels of the image, respectively.

CIFAR-10

$$\mu = (0.4914, 0.4822, 0.4465), \quad \sigma = (0.2470, 0.2435, 0.2616)$$

CIFAR-100

$$\mu = (0.5071, 0.4865, 0.4409), \quad \sigma = (0.2673, 0.2564, 0.2762)$$

ImageNet

$$\mu = (0.485, 0.456, 0.406), \quad \sigma = (0.229, 0.224, 0.225)$$

D Data augmentation

We applied standard data augmentation as part of our training data pipeline. The specific augmentation techniques are specified below. We apply the data normalization described in Appendix C after the augmentation transformations.

CIFAR-10 We pad the image with 4 black pixels on all sides and randomly crop a square of 32 by 32 pixels. We then perform a random horizontal flip of the crop with a probability of 0.5.

CIFAR-100 We pad the image with 4 black pixels on all sides and randomly crop a square of 32 by 32 pixels. We then perform a random horizontal flip of the crop with a probability of 0.5.

ImageNet We randomly crop a square of 224 by 224 pixels. We then perform a random horizontal flip of the crop with a probability of 0.5. For the test set, we resize the images to be 256 pixels followed by a square center crop of 224 pixels.

E Convolutional layer subset sampling

In this section we describe how our growing and pruning algorithm can be applied to convolutional layers. While a weight in a fully-connected layer is specified by the input and output units that it connects, a weight in a 2D convolutional layer is specified by the input channel, its x and y coordinate on the filter, and the output channel. It thus seems that it would require four discrete probability distributions to sample a weight of a 2D convolutional layer, instead of two for fully-connected layers. However, one can alternatively view a convolutional layer as applying a fully-connected layer on multiple subsections (patches) of the input, flattening the input channels and filters makes the weights of a convolutional layer identical to a fully-connected layer. The multiple subsections of the input can then simply be treated as additional batches. With this realization all the methods that we explain throughout the paper for fully-connected layers directly apply to convolutional layers.

F Sparse graph initialization comparison

In Figure 5, we plot the accuracy obtained by assigning the sparsity uniformly over all the layers compared to using the Erdős–Rényi initialization for SET, RigL, and our method. Note that Ours Uniform (Erdős–Rényi) denotes uniform sampling of the subset connections and the Erdős–Rényi initialization. The Erdős–Rényi initialization obtains better accuracy while ensuring that the number of active connections per layer is proportional to the layer width.

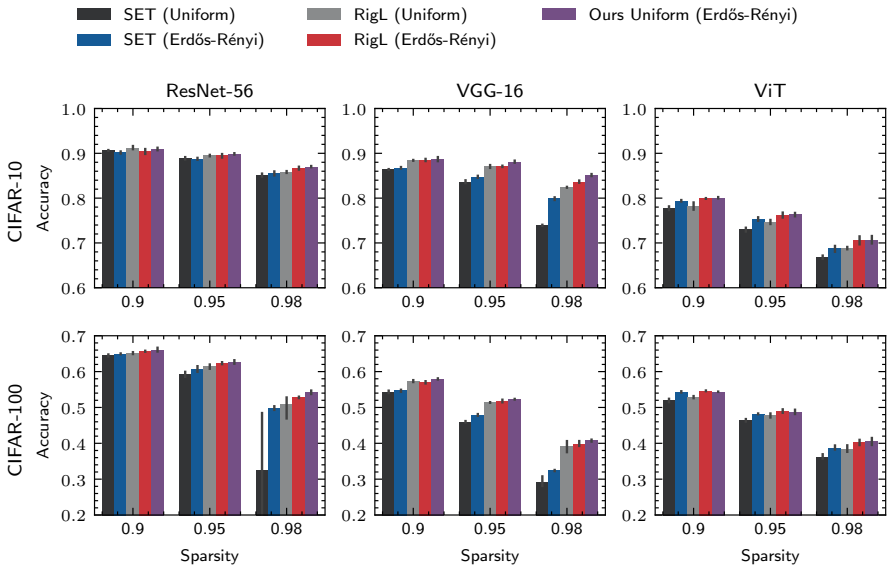

Figure 5: Accuracy comparison of uniform and Erdős–Rényi sparsity assignment.

G Pseudocode

We provide pseudocode for our complete dynamic sparse training process in the case of the GraBo or GraEst distributions in Algorithm 2. The uniform distribution is simpler as it does not require any aggregation of activity. For simplification, the pseudocode assumes that the prune-grow step only uses a single batch of training samples, allowing the aggregation for the probability distributions f and g to be computed in the same pass as the generation of the various connection sets. This is also the scenario we tested with in all our experiments.

Algorithm 2 Efficient periodic growing and pruning of connections

Input: Network f_θ, dataset $\mathcal{D}$, loss function $\mathcal{L}$, prune-grow schedule $T, T_{\text{end}}, \alpha$, number of active connections multiplier ϵ, and subset sampling factor γ.
for each layer l **do**
  $\mathbb{A} \leftarrow$ sample_graph$(n^{[l-1]}, n^{[l]}; \epsilon)$ {Erdős–Rényi random bipartite graph}
  $\theta \leftarrow$ sample_weights$(\mathbb{A})$
end for
for each training step t **do**
  $\boldsymbol{x}, \boldsymbol{y} \sim \mathcal{D}$ {Sample a mini batch}
  if $t \bmod T = 0$ **and** $t \leq T_{\text{end}}$ **then**
    $\boldsymbol{h} \leftarrow \boldsymbol{x}$
    for each layer l **do**
      $\boldsymbol{a}^{[l]} \leftarrow$ aggregate$(\boldsymbol{h})$ {Aggregate is distribution specific}
      $\boldsymbol{h} \leftarrow$ forward$(\boldsymbol{h})$ {Sparse forward pass}
    end for
    $\boldsymbol{g} \leftarrow$ grad$(\mathcal{L}(\boldsymbol{h}, \boldsymbol{y}))$ {Get the gradients at the outputs}
    for each layer l in inverse order **do**
      $\boldsymbol{b}^{[l]} \leftarrow$ aggregate$(\boldsymbol{g})$
      f $\leftarrow$ normalize$(\boldsymbol{a}^{[l]})$ {Create probability distribution}
      g $\leftarrow$ normalize$(\boldsymbol{b}^{[l]})$
      $\mathbb{S} \leftarrow$ sample_connections(f, g, $\lceil \gamma |\mathbb{A}| \rceil) \setminus \mathbb{A}$ {Sample the connections subset}
      $\alpha_t \leftarrow$ cosine_decay$(t; \alpha, T_{\text{end}})$ {Maximum prune fraction}
      $k \leftarrow \min(\lceil \alpha_t |\mathbb{A}| \rceil, |\mathbb{S}|)$ {Amount to prune and grow}
      $\mathbb{G} \leftarrow$ topk$(|\text{grad}(\mathbb{S})|, k)$ {Get grown connections set}
      $\mathbb{P} \leftarrow$ topk$(-|\theta|, k)$ {Get pruned connections set}
      $\mathbb{A} \leftarrow (\mathbb{A} \setminus \mathbb{P}) \cup \mathbb{G}$ {Construct new active connections set}
      $\theta \leftarrow$ update_weights$(\theta, \mathbb{A}, \mathbb{P}, \mathbb{G})$
      $\boldsymbol{g} \leftarrow$ backward$(\boldsymbol{g})$ {Sparse backward pass}
    end for
  else
    SGD$(\theta, \mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y}))$ {Regular SGD training}
  end if
end for

H Training progress

In Figure 6, we provide plots for the test accuracy during training for each baseline method and our method. It can be seen that our method consistently achieves among the highest test accuracy throughout training compared to the sparse training methods. The training progress of Gradual stands out because its accuracy increases the most at first, this is because it starts out as a dense model while it gradually increases the model sparsity, as a result, the accuracy goes down towards the second learning rate drop when the target sparsity is reached.

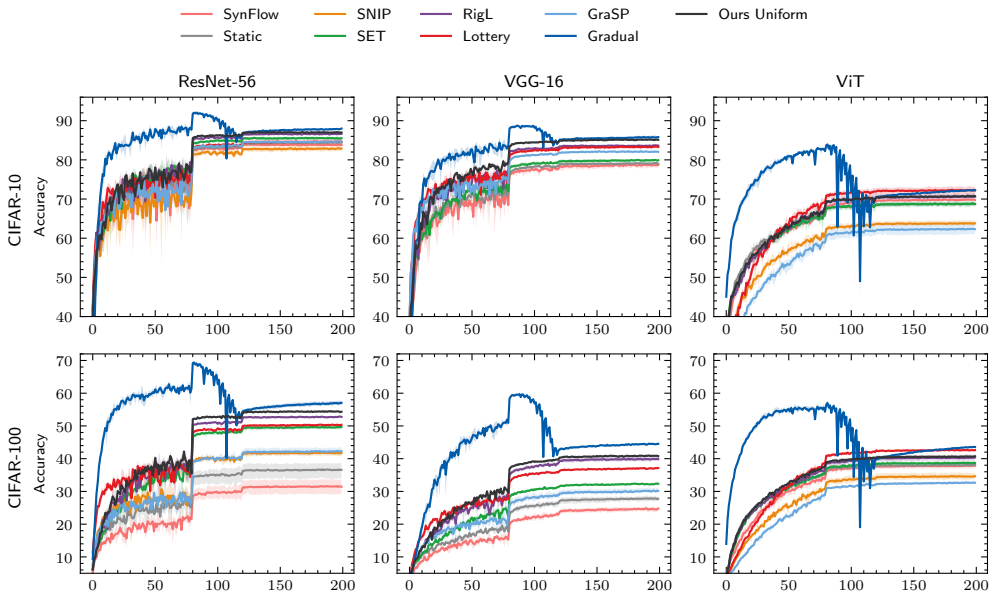

Figure 6: Training progress in test accuracy compared to related work at 98% sparsity.

I Extended training

The following extended training results are for CIFAR-100 at 98% sparsity with 1x, 1.5x, and 2x the number of training epochs. The learning rate drops and the final prune-grow step are also scaled by the same amount, all other hyperparameters stay the same. It is clear that all models benefit significantly from extended training. The ViT model in particular gains 7% in accuracy by training for twice as long.

Table 3: Accuracy of extended training on CIFAR-100 at 98% sparsity

Epochs	200 (1x)	300 (1.5x)	400 (2x)
ResNet-56	54.3±0.5	56.4±0.3	57.9±0.1
VGG-16	40.8±0.3	44.1±0.4	45.9±0.2
ViT	40.7±1.0	45.3±1.1	47.8±0.5

J Model scaling training progress

In Figure 7, we provide plots for the test and training accuracy during training for our method on CIFAR-100 while increasing the width of the model by 1x, 2x, 3x, and 4x. The number of active connections is kept constant between all experiments, see Section 4.5 for details. In the CNN models, ResNet-56 and VGG-16, the increase in width causes the models to train significantly faster and obtain higher accuracy. For the vision transformer (ViT) model, the trend is reversed, wider models train slower and obtain lower accuracy.

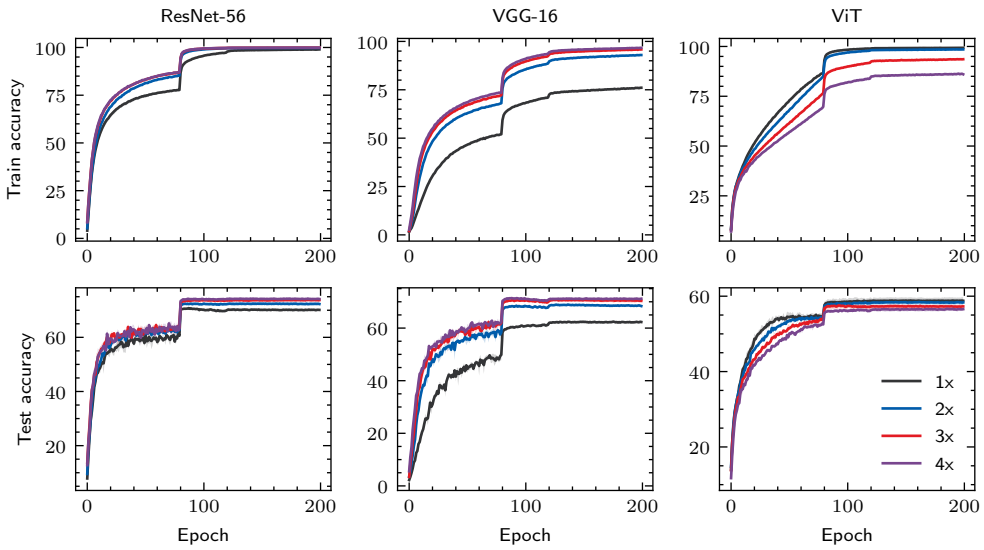

Figure 7: Training progress in train and test accuracy for various model widths on CIFAR-100.

K Scaling dense ViT

To further investigate the behavior of ViT, which decreases in accuracy when using larger models with the same number of activate parameters, as discussed in Section 4.5 and Appendix J. In Table 4, we report the accuracy of ViT for increasing model width but kept dense, thus increasing the active connections count. We see the accuracy quickly plateaus at 62.2%, gaining only 2% accuracy while even the sparse CNNs gained 6.5% on average at 4 times wider. This indicates that the decreasing accuracy of ViT reported in Section 4.5 is not due to our sparse training method as even the dense model does not benefit much from increased scale.

Table 4: Accuracy of scaling dense ViT on CIFAR-100

Model width	1x	2x	3x	4x
Accuracy	60.3±0.6	62.5±0.5	62.2±0.4	62.2±0.1

L Sparse matrix multiply benchmark

In Figure 8, we show the execution time of the sparse times dense matrix-matrix multiply on two execution devices: Intel Xeon Gold 6148 CPU at 2.40GHz and NVIDIA Tesla V100 16GB. The sparse matrix is a square matrix of size units times units, representing an unstructured sparse random weight matrix. The dense matrix is of size units times 128 and represents a batch of layer inputs. We compare three storage formats, namely: dense, where all the zeros are explicitly represented; Coordinate format (COO), which represents only the nonzero elements and their coordinates; and Compressed Sparse Row (CSR), which is similar to COO but compresses the row indices. Each data point in Figure 8 is the average of 10 measurements.

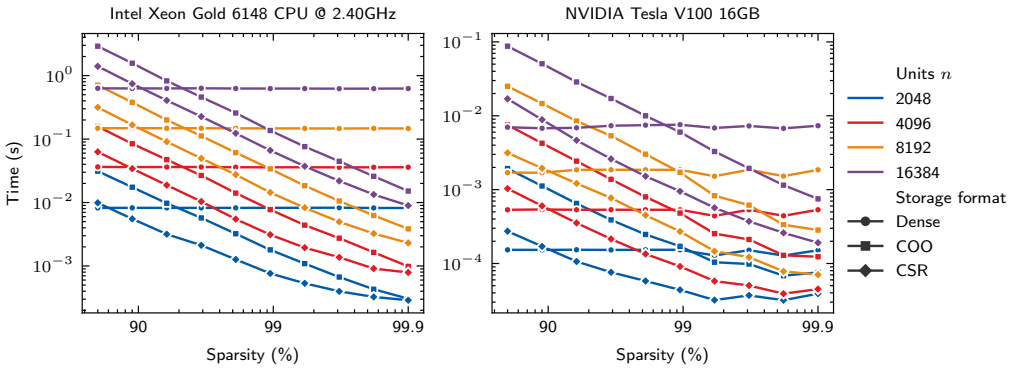

Figure 8: Execution time of sparse times dense matrix-matrix multiply.

The results show that the CSR format generally outperforms the COO format. Moreover, the execution time of the CSR format improves upon the dense execution time starting around 90% sparsity. At 99% sparsity, the CSR format is up to an order of magnitude faster than the dense format. These results are similar between the two execution devices, although the NVIDIA Tesla V100 16GB is up to two orders of magnitude faster than the Intel Xeon Gold 6148 CPU at 2.40GHz.

M Layer-wise sparsity

In Figure 9, we show the sparsity of each layer, in terms of compression, obtained by each sparsification methods at the end of training. In the plots the CIFAR-10 and CIFAR-100 results are averaged. These plots can be used to gain insight into how each sparsification method distributes the active connections over the models.

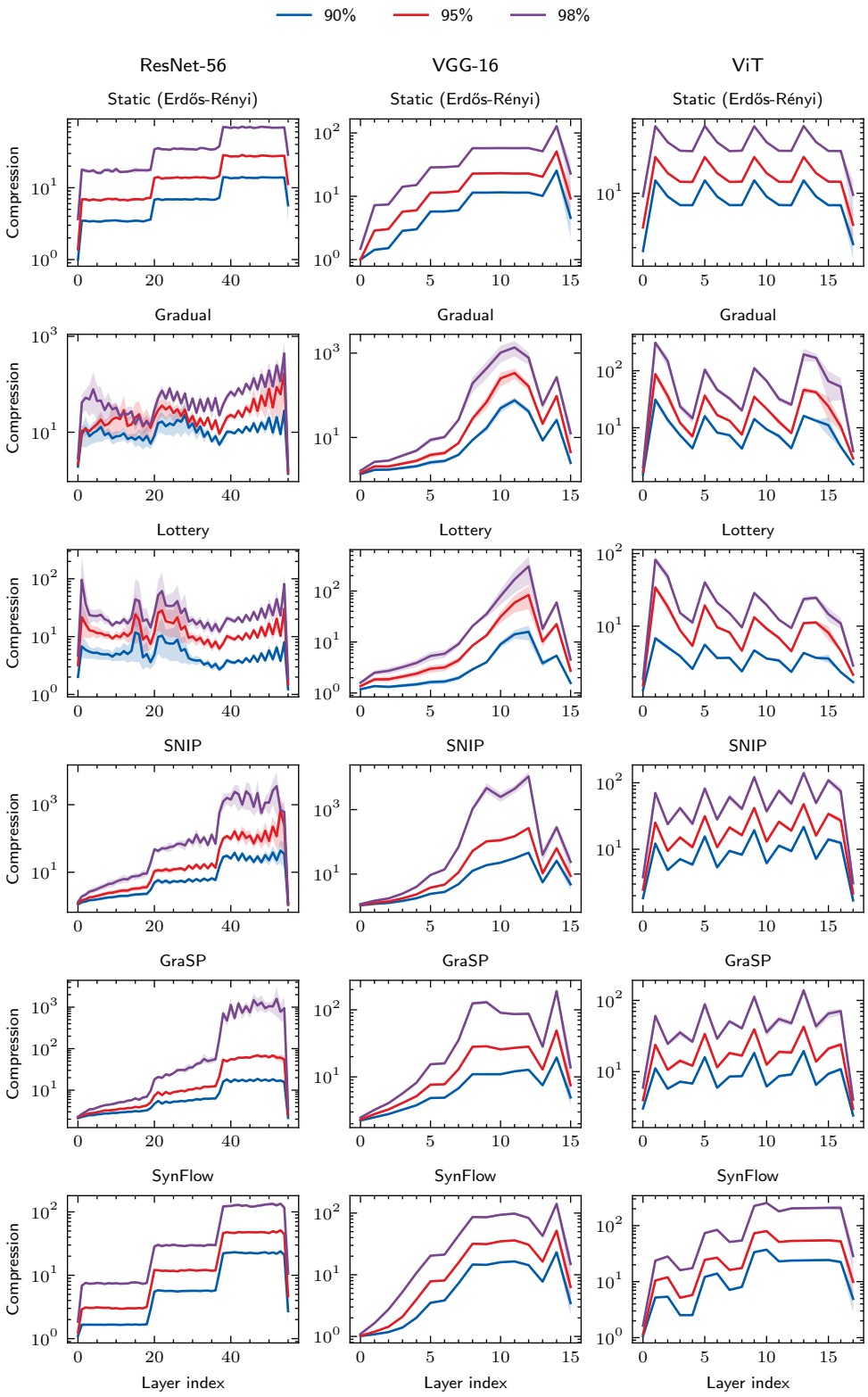

Figure 9: (Part 1) Layer-wise sparsity from each sparsification method at the end of training. Showing averaged results on the CIFAR-10/100 datasets.

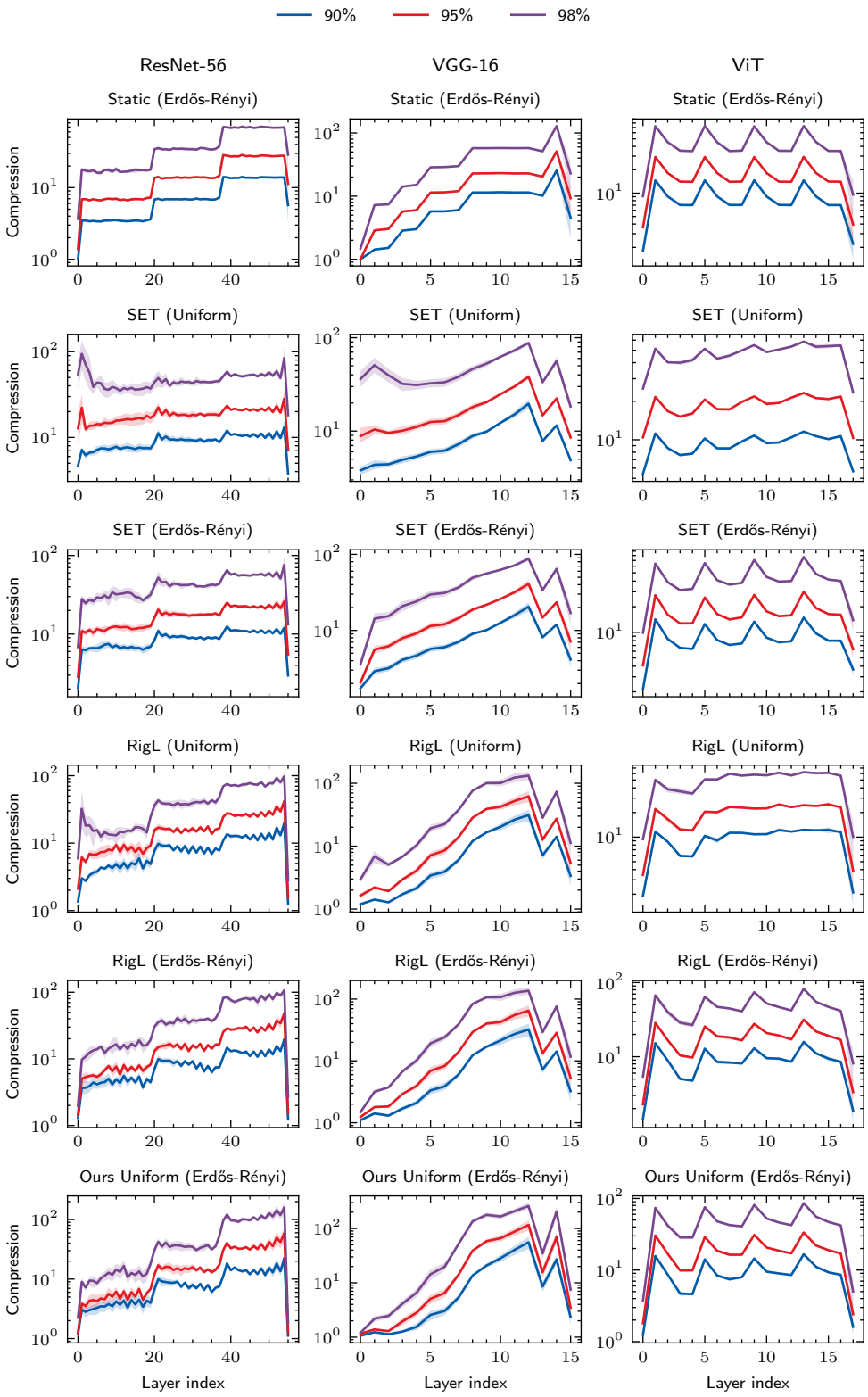

Figure 9: (Part 2) Layer-wise sparsity from each sparsification method at the end of training. Showing averaged results on the CIFAR-10/100 datasets.