Appendices

A Sparsity and Output-Norm Variance

Consider a SNN with ReLU activations, where each neuron has on average k connections to the previous layer (i.e., fan-in). It has been shown by Evci et al. (2022), that by normalizing the weights on initialization by a factor of $\sqrt{2/k}$, one achieves the following desirable normalization property for each layer ℓ with output z^ℓ:

$$\mathbb{E}\left(\frac{||z^{\ell+1}||^2}{||z^\ell||^2}\right)=1,$$

Meaning that on average the variance of the norm of each layer's output is constant. However, the variance of this ratio is non-trivial. In networks with large depth, it can accumulate, leading to exponentially large variance at the final layer (Li et al., 2021). Minimizing this variance on initialization has been shown to have a positive effect on training dynamics in some network models (Littwin et al., 2020), as it stabilizes the gradients. We therefore analyze the output norm variance as a guiding quantity for sparsity-type selection.

In the following, we consider three different types of sparsity distributions, which respectively correspond to different degrees of sparsity *structure* in the SNN, and derive analytic expressions for the behaviour of output norm variance in SNNs with the given sparsity type. The derivations for the following results can be found in Appendix B:

- **"Bernoulli sparsity"**: A connection between each neuron in layer $\ell+1$ and each neuron in layer ℓ appears *independently* with probability $p=\frac{k}{n}$, resulting in each neuron having k connections *on average* and each layer having nk connections *on average*. The variance is:

$$\mathbf{Var}_{\text{Bernoulli}}\left(\frac{||z^{\ell+1}||^2}{||z^\ell||^2}\right)=\frac{5n-8+18\frac{k}{n}}{n(n+2)}. \tag{1}$$

- **"Constant Per-Layer sparsity"**: Exactly kn connections are distributed at random in the layer connecting the n neurons in layer $\ell+1$ and the n neurons in layer ℓ, resulting in each neuron having k connections *on average*. The variance is:

$$\mathbf{Var}_{\text{Const-Per-Layer}}\left(\frac{||z^{\ell+1}||^2}{||z^\ell||^2}\right)=\frac{(n^2+7n-8)C_{n,k}+18\frac{k}{n}-n^2-2n}{n(n+2)}, \tag{2}$$

  where $C_{n,k}=\frac{n-1/k}{n-1/n}$. Note that when $n\gg 1$, $C_{n,k}\approx 1-\frac{n-k}{n^2k}$ is close to 1, and with $C_{n,k}=1$ we recover the formula for Bernoulli sparsity, meaning that this sparsity type and Bernoulli sparsity are very similar.
- **"Constant Fan-In sparsity"**: Each neuron in layer $\ell+1$ is connected to exactly k neurons from layer ℓ, chosen uniformly at random. In this case, the variance is:

$$\mathbf{Var}_{\text{Const-Fan-In}}\left(\frac{||z^{\ell+1}||^2}{||z^\ell||^2}\right)=\frac{5n-8+18\frac{k}{n}}{n(n+2)}-\frac{3(n-k)}{kn(n+2)}. \tag{3}$$

In deriving the above results we assumed that the direction of the layer output vector $\frac{z^\ell}{||z^\ell||}$ is uniformly distributed on the unit sphere. We compare our theoretical predictions with simulations in Fig. 1b and verify their accuracy. Bernoulli and constant-per-layer distribution result in unstructured sparsity, and most of the current DST approaches, including RigL, operate with constant-per-layer sparsity. In contrast, the constant-fan-in type imposes a strong structural constraint. Therefore we are somewhat surprised to find that, in fact, constant-fan-in sparsity always produces slightly smaller output-norm variance than the other types. The difference is larger when $k\ll n$, i.e., for very sparse networks. This indicates that, at the very least, the constant fan-in constraint should not impair SNN training dynamics and performance, motivating our method of maintaining the constant fan-in sparsity constraint within a DST approach.

B COMPUTING THE OUTPUT NORM VARIANCE

Definition B.1. Let $\xi \in \{0,1\}^N$ be a binary vector. Let $I \in \{0,1\}^{N \times N}$ be an $N \times N$ binary matrix. Let $u \in \mathbb{R}^N$ be any vector. Let $W \in \mathbb{R}^{N \times N}$ be a matrix of iid $\mathcal{N}(0,1)$ random variables.

Define the vector z by:

$$z = \sqrt{\frac{2}{k}}(W \odot I)(\xi \odot u) \tag{4}$$

i.e. the entries z_i are given by:

$$z_i = \sqrt{\frac{2}{k}} \sum_{j=1}^{n} W_{ij} I_{ij} \xi_j u_j \tag{5}$$

Proposition B.2. *The variance of each entry z_i is:*

$$\mathbf{Var}(z_i) = \frac{2}{k} \sum_{j=1}^{n} I_{ij} \xi_j u_j^2 \tag{6}$$

and therefore the distribution of each z_i can be written as

$$z_i \stackrel{d}{=} g_i \sqrt{\frac{2}{k} \sum_{j=1}^{n} I_{ij} \xi_j u_j^2} \tag{7}$$

where g_i are N iid $\mathcal{N}(0,1)$ random variables.

Proof. By the properties of variance:

$$\mathbf{Var}(z_i) = \frac{2}{k} \sum_{j,j'} I_{ij} I_{ij'} \xi_j \xi_{j'} u_j u_j' \mathbf{Cov}(W_{ij}, W_{ij'}) \tag{8}$$

$$= \frac{2}{k} \sum_{j,j'} I_{ij} I_{ij'} \xi_j \xi_{j'} u_j u_j' \delta_{j=j'} \tag{9}$$

$$= \frac{2}{k} \sum_{j} I_{ij}^2 \xi_j^2 u_j^2 \tag{10}$$

$$= \frac{2}{k} \sum_{j} I_{ij} \xi_j u_j^2 \tag{11}$$

since $I_{ij}^2 = I_{ij}$ and $\xi_j^2 = \xi_j$ because they are binary valued. Once the variance is established, notice that z_i is a linear combination of Gaussians with $z_i \perp z_{i'}$, because the row $W_{ij} \perp W_{i'j}$. Hence the z_i are independent Gaussians, so the form $z_i \stackrel{d}{=} g_i \sqrt{\frac{2}{k} \sum_{j=1}^{n} I_{ij} \xi_j u_j^2}$ follows. □

Corollary B.3. *The norm $\|z\|^2$ can be written as:*

$$\|z\|^2 \stackrel{d}{=} \frac{2}{k} \sum_{i,j=1}^{n} g_i^2 I_{ij} \xi_j u_j^2 \tag{12}$$

Proposition B.4 ("Bernoulli Sparsity")**.** *Suppose that $u \in \mathbb{R}^n$ is uniform from the unit sphere, the entries $I_{ij} \sim Ber\left(\frac{k}{n}\right)$, $\xi_j \sim Ber(\frac{1}{2})$ all independent of each other. Then:*

$$\mathbb{E}\left(\|z\|^2\right) = 1 \tag{13}$$

$$\mathbf{Var}\left(\|z\|^2\right) = \frac{5n - 8 + 18\frac{n}{k}}{n(n+2)} \tag{14}$$

Case	Num. Terms	$\mathbb{E}\big[g_i^2 g_{i'}^2\big]$	$\mathbb{E}[I_{i'j'}I_{ij}]$	$\mathbb{E}[\xi_j\xi_{j'}]$	$\mathbb{E}\big[u_j^2 u_{j'}^2\big]$
$i=i',j=j'$	n^2	3	$\frac{k}{n}$	$\frac{1}{2}$	$\frac{3}{n(n+2)}$
$i\neq i',j=j'$	$n^2(n-1)$	1	$\left(\frac{k}{n}\right)^2$	$\frac{1}{2}$	$\frac{3}{n(n+2)}$
$i=i',j\neq j'$	$n^2(n-1)$	3	$\left(\frac{k}{n}\right)^2$	$\left(\frac{1}{2}\right)^2$	$\frac{1}{n(n+2)}$
$i\neq i',j\neq j'$	$n^2(n-1)^2$	1	$\left(\frac{k}{n}\right)^2$	$\left(\frac{1}{2}\right)^2$	$\frac{1}{n(n+2)}$

Table 6: Overview of terms for Bernoulli type sparsity.

Proof. We have

$$\mathbb{E}\Big(\|z\|^2\Big) = \frac{2}{k}\sum_{i,j=1}^{n}\mathbb{E}\big[g_i^2 I_{ij}\xi_j u_j^2\big] \tag{15}$$

$$= \frac{2}{k}\sum_{i,j=1}^{n}\mathbb{E}\big[g_i^2\big]\mathbb{E}[I_{ij}]\mathbb{E}[\xi_j]\mathbb{E}\big[u_j^2\big] \tag{16}$$

$$= \frac{2}{k}\sum_{i,j=1}^{n} 1\cdot\frac{k}{n}\cdot\frac{1}{2}\cdot\frac{1}{n} \tag{17}$$

$$= 1 \tag{18}$$

Similarly, we compute the 4-th moment as follows:

$$\mathbb{E}\Big(\|z\|^4\Big) = \left(\frac{2}{k}\right)^2\sum_{i,j,i',j'}^{n}\mathbb{E}\big[g_i^2 g_{i'}^2\big]\mathbb{E}[I_{i'j'}I_{ij}]\mathbb{E}[\xi_j\xi_{j'}]\mathbb{E}\big[u_j^2 u_{j'}^2\big] \tag{19}$$

We split this into four cases and evaluate these based on whether or not $i=i'$ and $j=j'$ in the following table.

Combining the value of each term with the number of terms gives the desired result for the variance. □

Proposition B.5 ("Constant-per-layer sparsity"). *Suppose that $u\in\mathbb{R}^n$ is uniform from the unit sphere and $\xi_j\sim Ber(\frac{1}{2})$ are independent of each other. Suppose the entries of the matrix I_{ij} are chosen such that:*

There are exactly kn ones and exactly n^2-nk zeros in the matrix I, and their positions in the matrix are chosen uniformly from the $\binom{n^2}{nk}$ possible configurations. Then:

$$\mathbb{E}\Big(\|z\|^2\Big) = 1 \tag{20}$$

$$\mathbf{Var}\Big(\|z\|^2\Big) = \frac{(n^2+7n-8)C_{n,k}+18\frac{k}{n}-n^2-2n}{n(n+2)} \tag{21}$$

Proof. Note that $\mathbb{E}(I_{ij})=k/n$ still holds, since there are kn ones distributed over n^2 locations. Thus the computation for $\mathbb{E}(\|z\|^2)$ is identical to the previous proposition. Note also that when there are two entries, we have:

$$\mathbb{E}[I_{ij}I_{i'j'}] = \begin{cases}\frac{k}{n} & \text{if } i=i' \text{ and } j=j'\\ \frac{k}{n}\cdot\frac{nk-1}{n^2-1} & \text{otherwise}\end{cases} \tag{22}$$

$$= \begin{cases}\frac{k}{n} & \text{if } i=i' \text{ and } j=j'\\ \left(\frac{k}{n}\right)^2\cdot C_{n,k} & \text{otherwise}\end{cases} \tag{23}$$

where $C_{n,k}=\frac{n-1/k}{n-1/n}$. The table with terms for computing $\mathbb{E}(\|z\|^4)$ becomes: The extra factor of $C_{n,k}$ in the entries leads to the stated result. □

Case	Num. Terms	$\mathbb{E}\left[g_i^2 g_{i'}^2\right]$	$\mathbb{E}[I_{i'j'}I_{ij}]$	$\mathbb{E}[\xi_j \xi_{j'}]$	$\mathbb{E}\left[u_j^2 u_{j'}^2\right]$
$i=i', j=j'$	n^2	3	$\frac{k}{n}$	$\frac{1}{2}$	$\frac{3}{n(n+2)}$
$i\neq i', j=j'$	$n^2(n-1)$	1	$\left(\frac{k}{n}\right)^2 C_{n,k}$	$\frac{1}{2}$	$\frac{3}{n(n+2)}$
$i=i', j\neq j'$	$n^2(n-1)$	3	$\left(\frac{k}{n}\right)^2 C_{n,k}$	$\left(\frac{1}{2}\right)^2$	$\frac{1}{n(n+2)}$
$i\neq i', j\neq j'$	$n^2(n-1)^2$	1	$\left(\frac{k}{n}\right)^2 C_{n,k}$	$\left(\frac{1}{2}\right)^2$	$\frac{1}{n(n+2)}$

Table 7: Overview of terms for Constant-per-layer type sparsity.

Proposition B.6 ("Constant Fan-In sparsity"). *Suppose that $u \in \mathbb{R}^n$ is uniform from the unit sphere, and $\xi_j \sim Ber(\frac{1}{2})$ all independent of each other. Suppose the entries of the matrix I_{ij} are chosen so that:*

1. *There are exactly k ones **in each row** of the matrix I and exactly $n-k$ zeros in the matrix I, chosen uniformly from the $\binom{n}{k}$ possible ways this can happen.*
2. *Different rows of I are independent.*

Then:

$$\mathbb{E}\left(\|z\|^2\right) = 1 \tag{24}$$

$$\mathbf{Var}\left(\|z\|^2\right) = \frac{5n-8+18\frac{n}{k}}{n(n+2)} - \frac{3(n-k)}{kn(n+2)} \tag{25}$$

Proof. Same arguments as before apply, but now we have

$$\mathbb{E}[I_{ij}I_{i'j'}] = \begin{cases} \frac{k}{n} & \text{if } i=i' \text{ and } j=j' \\ \frac{k}{n}\frac{k-1}{n-1} & \text{if } i=i' \text{ and } j\neq j' \\ \left(\frac{n}{n}\right)^2 & \text{otherwise} \end{cases} \tag{26}$$

(27)

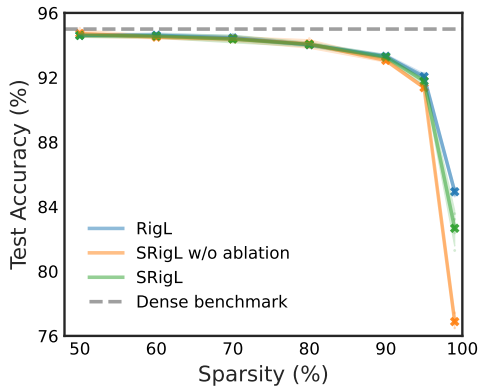
and the table for the variance computation becomes:

Case	Num. Terms	$\mathbb{E}\left[g_i^2 g_{i'}^2\right]$	$\mathbb{E}[I_{i'j'}I_{ij}]$	$\mathbb{E}[\xi_j \xi_{j'}]$	$\mathbb{E}\left[u_j^2 u_{j'}^2\right]$
$i=i', j=j'$	n^2	3	$\frac{k}{n}$	$\frac{1}{2}$	$\frac{3}{n(n+2)}$
$i\neq i', j=j'$	$n^2(n-1)$	1	$\left(\frac{k}{n}\right)^2$	$\frac{1}{2}$	$\frac{3}{n(n+2)}$
$i=i', j\neq j'$	$n^2(n-1)$	3	$\frac{k}{n}\cdot\frac{k-1}{n-1}$	$\left(\frac{1}{2}\right)^2$	$\frac{1}{n(n+2)}$
$i\neq i', j\neq j'$	$n^2(n-1)^2$	1	$\left(\frac{k}{n}\right)^2$	$\left(\frac{1}{2}\right)^2$	$\frac{1}{n(n+2)}$

Table 8: Overview of terms for Constant-fan-in type sparsity.

Which leads to the stated result. □

(28)



sparsity (%)	RigL	SRigL w/o	SRigL w/ ablation
50	94.6±0.1	94.7±0.1	94.6±0.1
60	94.6±0.1	94.5±0.1	94.6±0.1
70	94.5±0.1	94.4±0.1	94.4±0.1
80	94.0±0.1	94.1±0.2	94.0±0.1
90	93.3±0.1	93.1±0.1	93.3±0.1
95	92.1±0.1	91.4±0.1	91.8±0.2
99	**84.9±0.2**	76.9±0.3	82.7±0.8
0	*dense Wide ResNet-22*: 95.0		

Figure 5 & Table 9: Test accuracy of Wide ResNet-22 trained on CIFAR-10. Mean and 95% confidence intervals are reported over five runs.

C WIDE RESNET-22 TRAINED ON CIFAR-10

In Fig. 5 we present results of training Wide ResNet-22 (Zagoruyko & Komodakis, 2017) with RigL or SRigL on the CIFAR-10 dataset. The training details for this experiment are identical to those reported in Section 4.1. SRigL without ablation performs poorly at very high sparsities. With ablation, SRigL achieves generalization performance comparable to RigL.

D HYPERPARAMETER AND TRAINING DETAILS

D.1 RESNET-18 TRAINED ON CIFAR-10

As per Liu (2017), we modify the original ResNet-18 network by changing the kernel dimensions of the first convolutional layer to 3×3 instead of 7×7. Further, we reduce the stride in the first two convolutional layers to one to avoid excessive reduction of the feature map's spatial dimensions.

We train each network for 250 epochs (97,656 steps) using a batch size of 128. An initial learning rate of 0.1 is reduced by a factor of 5 every 77 epochs (about 30,000 steps). We use stochastic gradient descent (SGD) with momentum, with an L2 weight decay coefficient of 5e-4 and momentum coefficient of 0.9. We train each model using a single Nvidia V100 GPU.

We achieve the desired overall sparsity by distributing the per-layer sparsity according to the ERK (Evci et al., 2021; Mocanu et al., 2018) distribution, which scales the per-layer sparsity based on the number of neurons and the dimensions of the convolutional kernel, if present. We set the number of mini-batch steps between connectivity updates, ΔT, to 100. γ_{sal} is set at 30% based on the results of a small grid search performed on CIFAR-10 with ResNet-18 and Wide ResNet-22. See Fig. 8 for details.

For each trial, we select a desired sparsity in the range from 0.5 to 0.99. At each connectivity update, the portion of weights to be pruned or regrown is based on a cosine annealing schedule (Dettmers & Zettlemoyer, 2019) with an initial value $\alpha = 0.3$. The portion of weights to be updated decays from the initial value to zero once 75% of the total training steps have been completed, after which the weight mask remains constant.

D.2 RESNET-50 TRAINED ON IMAGENET

We use a mini-batch size of 512 instead of 4096, We linearly scale the learning rate and ΔT to account for our smaller batch size. Linearly scaling the learning rate in this manner was included in the original RigL source code and is further motivated by Goyal et al. (2018). We increase ΔT to 800 and average the dense gradients over eight mini-batch steps to ensure that SRigL has the same quality of parameter saliency information available as RigL at each network connectivity update. We set γ_{sal} to 30% based on our grid search presented in Fig. 8.

Our learning rate uses a linear warm-up to reach a maximum value of 0.2 at epoch five and is reduced by a factor of 10 at epochs 30, 70, and 90. Using a mini-batch of 512, we train the networks for 256,000 steps to match RigL's training duration. We use a cosine connectivity update schedule with $\alpha = 0.3$. We initialize the sparse model weights per Evci et al. (2022). We train the networks using SGD with momentum, L2 weight decay, and label smoothing (Szegedy et al., 2016) coefficients of 0.9, 1e-4 and 0.1, respectively.

We use the same standard data augmentation in our data preprocessing as RigL, including randomly resizing to 256×256 or 480×480 pixels, random crops to 224×224 pixels, random horizontal flips, and per-image normalization to zero mean and unit variance using identical per RGB channel mean and standard deviation values as RigL. We train each model using either four Nvidia V100 or A100 GPUs.

D.3 Vision Transformer trained on ImageNet

For our ViT-B/16 experiments, we used sparsity on the convolutional projection (input projection to patches), the fully connected layers in the feed forward (MLP) blocks and the output projections of the multi-headed attention (MHA) modules. We performed a lightweight ablation study on four ViT-B/16 networks trained on ImageNet to determine the affect of sparsifying the first convolutional projection layer as well as the input projection layers in the multi-headed attention modules. Based on the results of our ablation study, *we did not use sparsity on the MHA input projection layers or the scaled-dot products*. See Fig. 9b for more details. This setup is similar to the "Sparse FF" models investigated by Jaszczur et al. (2021). The global model sparsity level reported in Table 4 is calculated based on the sparse modules only. If we also consider the parameters in the MHA input projections as part of our parameter budget, the global model sparsities tabulated in Table 4 correspond to 60.35% and 67.90% for the rows labelled 80% and 90% sparsity, respectively.

We add additional data augmentations following the standard TorchVision (Maintainers & Contributors, 2016) ViT-B/16 training procedure for ImageNet. These data augmentations applied include: random cropping, resizing the cropped image to 224 by 224 pixels, randomly horizontal flips, randomly augmenting with RandAugment algorithm (Cubuk et al., 2020), and normalizing with the typical RGB channel mean and standard deviation values. We also randomly choose one of random mixup (Zhang et al., 2023) or random cutmix (Yun et al., 2019) and add it to the above-noted augmentations. We use 0.2 and 1.0 for the alpha parameter values for mixup and cutmix, respectively. We omit Dropout (Srivastava et al., 2014) from the model entirely to avoid potential layer collapse in the case where all non-zero weights are dropped from a layer and to avoid any other unintended interference with SRigL's sparse training procedure.

We sample eight mini-batch steps with 512 samples per mini-batch and accumulate gradients before applying the optimizer, resulting in an effective mini-batch size of 4096. We train the model for 150 epochs using an AdamW (Loshchilov & Hutter, 2018) optimizer with weight decay, label smoothing, β_1, and β_2 coefficients of 0.3, 0.11, 0.9, and 0.999, respectively. We use cosine annealing with linear warm-up for our learning rate scheduler with an initial learning rate of 9.9e-5 that warms-up to a maximum value of 0.003 at epoch 16. We clip all parameter gradients to a max L2 norm of 1.0. We apply uniformly distributed sparsity across all layers in the model. ΔT is set to 100 to update network connectivity every 100 mini-batch steps. We train each model using either four Nvidia V100 or A100 GPUs.

D.4 MobileNet-V3 trained on ImageNet

We follow the TorchVision (Maintainers & Contributors, 2016) training recipe for MobileNet-V3 Large and Small for ImageNet. We set ΔT to 100 and γ_{sal} to 30% similar to our other CNN experiments. We train the models from scratch for 600 epochs using an RMSProp (Tieleman et al., 2012) optimizer with a momentum, L2 weight decay, and smoothing constant coefficients of 0.9, 1e-5, and 0.9, respectively. The networks are trained with a step learning rate decay schedule with initial learning rate of 0.064, multiplicative factor of 0.973, and we decay the learning rate every two epochs.

The input data is augmented with random cropping to 224 by 224 pixels, random horizontal flips, AutoAugmentation using the ImageNet policy (Cubuk et al., 2019), normalizing to standard RGB mean and standard deviation values, and random erasing with a probability of 0.2 (Zhong et al., 2017). Similar to the above, we omit Dropout (Srivastava et al., 2014) to avoid potential layer collapse. Unlike the TorchVision recipe, we *do not* average the trained parameters across the last three checkpoints that improved the top-1 accuracy. We train with a batch size of 512 and accumulate gradients across

two mini-batches, resulting in an effective mini-batch size of 1024. We train each model using four Nvidia A100 GPUs.

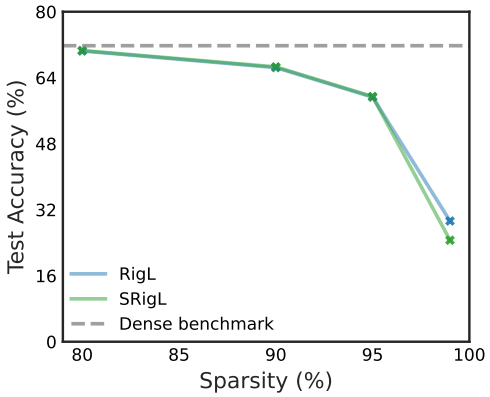


Figure 6: MobileNet-V3 Large / ImageNet top-1 accuracy. SRigL compares well against RigL both both models perform poorly compared to the denes baseline at 99% sparsity.

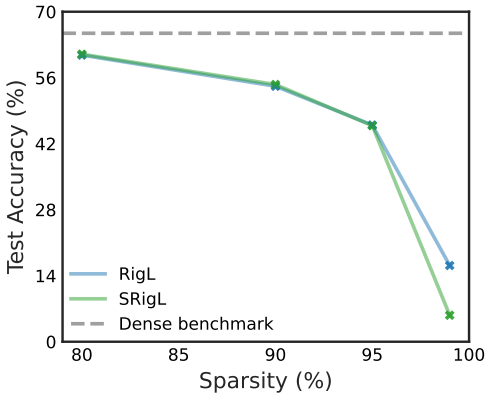
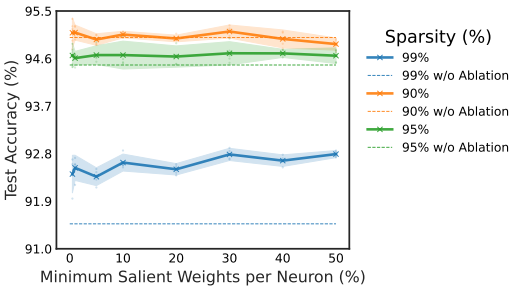


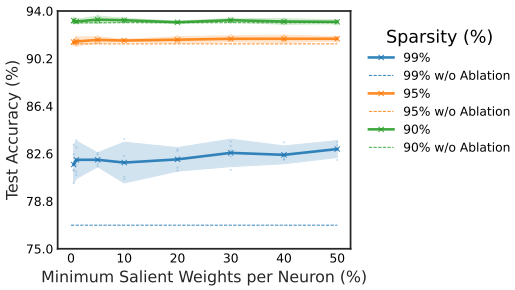
Figure 7: MobileNet-V3 Small / ImageNet top-1 accuracy. SRigL compares well against RigL both both models perform poorly compared to the denes baseline at 99% sparsity.

E TUNING γ_{sal}, MINIMUM PERCENTAGE SALIENT WEIGHTS PER NEURON

Fig. 8 depicts the generalization performance of highly sparse ResNet-18 and Wide ResNet-22 models trained on the CIFAR-10 dataset. SRigL's generalization performance at high sparsities is improved with neuron ablation; however, the specific value selected for γ_{sal} does not have a significant effect on performance. Our experiments demonstrate that SRigL performs well with a variety of γ_{sal} values. In Section 4 we report the results of SRigL models trained with γ_{sal} set to 30%. With dynamic ablation enabled, we set the minimum salient weights per neuron to one if the user-defined threshold results in a value less than one. In Fig. 10, many layers in ResNet-50 are set to the minimum threshold of one when we apply a γ_{sal} of 30% for all model types other than ViT-B/16. This minimum threshold explains the invariance of the model's performance when comparing against multiple values for γ_{sal}.



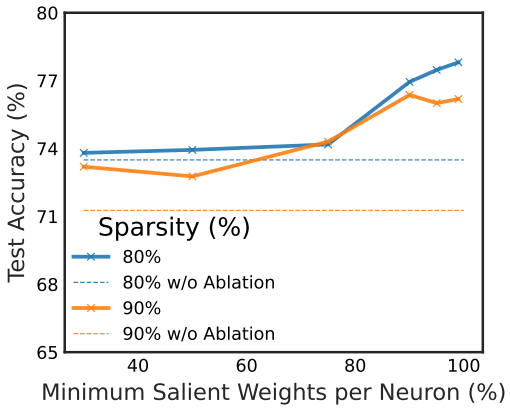
(a) ResNet-18/CIFAR-10



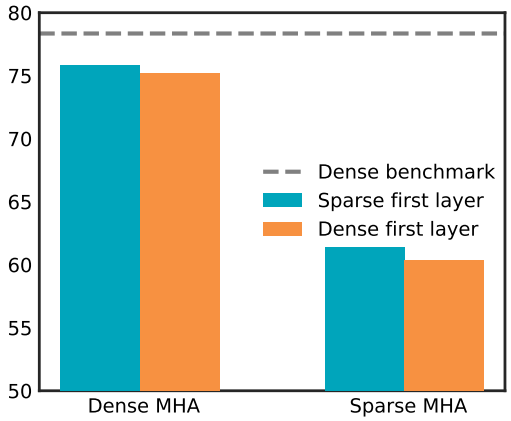
(b) Wide ResNet-22/CIFAR-10

Figure 8: (a) ResNet18/CIFAR-10 Test Accuracy vs. γ_{sal} when trained with SRigL with and without ablation for a range of sparsities. The mean and 95% confidence intervals are shown for five different random seeds for the runs with ablation. For the runs without ablation, we report the mean of five different random seeds. **(b) Wide ResNet-22 Test Accuracy vs.** γ_{sal}. The mean and 95% confidence intervals are shown for five different random seeds.

Fig. 9a demonstrates how ViT-B/16's generalization performance is much more sensitive to γ_{sal}. We find that RigL learns a sparse connectivity pattern with a large variance in sparse fan-in between neurons within a given layer, with some neurons having an order of magnitude more fan-in connection than the mean fan-in.



(a) ViT-B/16/ImageNet Test Accuracy vs. γ_{sal} when trained with SRigL with and without ablation enabled for 80% and 90% sparsity. ViT-B/16's performance is much more sensitive to γ_{sal} and generally performs best with high ablation thresholds. Based on this data we set γ_{sal} to 95% for our results reported in Section 4.3.



(b) ViT-B/16 ablation study. The best performing variant used a sparse first layer and dense input projections in the MHA modules.

F CONDENSED MATRIX MULTIPLICATION

Using a constant fan-in sparse representation presents an advantage compared to the general N:M sparse representation in that we can represent our weight matrices in a compact form, since every neuron/convolutional filter has the same number of non-zero weights. Here we demonstrate how this can be used to accelerate a fully-connected layer.

Consider the standard matrix-vector product:

$$Wv = \begin{pmatrix} W_{11} & W_{12} & \dots & W_{1d} \\ W_{21} & W_{22} & \dots & W_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ W_{n1} & W_{n2} & \dots & W_{nd} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{d} W_{1j} v_j \\ \sum_{j=1}^{d} W_{2j} v_j \\ \vdots \\ \sum_{j=1}^{d} W_{nj} v_j \end{pmatrix} = v^{\text{out}} \tag{29}$$

When $W \in \mathbb{R}^{n \times d}$ is sparse and has only k non-zero elements per row, the sums representing each element of v^{out} will be limited to k terms, i.e.:

$$v_i^{\text{out}} = \sum_{\alpha=1}^{k} W_{ij_\alpha} v_{j_\alpha} \quad \text{with } j_\alpha \in \{1,\dots,d\}, \quad j_\alpha \neq j_{\alpha'} \tag{30}$$

Note that the expression on the right-hand side of Eq. (29) can be represented as an operation between a dense matrix $W^c \in \mathbb{R}^{n \times k}$ (we call it "condensed W") and k vectors $v^{\pi_1},\dots,v^{\pi_k}$, $v^{\pi_i} \in \mathbb{R}^n$, whose elements are drawn from v with replacement (we call them "recombinations of v"). The operation is a sum over element-wise products between the i-th column of W^c and the i-th column vector v^{π_i}:

$$Wv = \sum_{i=1}^{k} W^c_{:,i} \odot v^{\pi_i} \tag{31}$$

Mathematically, these methods are equivalent for any matrices. Computationally, the condensed method can be more efficient, in particular for sparse matrices with constant small fan-in k. By construction, this method requires the sparse matrix W to be stored in dense representation which involves two 2D arrays of shape $n \times k$: One holds the *values* of the non-zero elements of W and the other one their respective *column indices*, which are used to generate input vector re-combinations. An efficient computational implementation of this method is subject of ongoing work on this project. Based on our results, the constant fan-in constraint does not appear to have a limiting effect on SNNs.

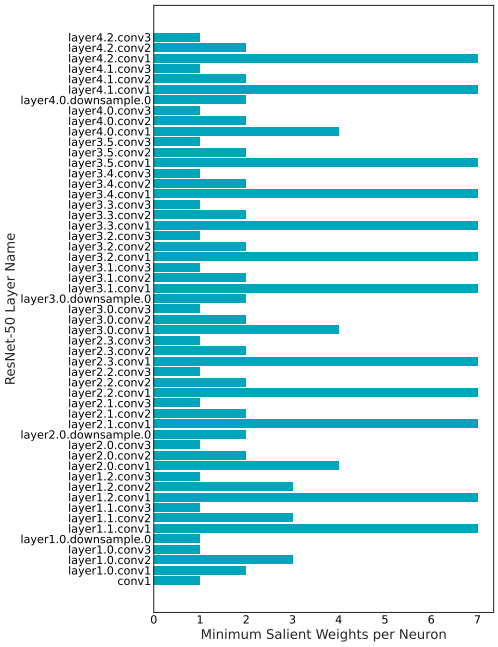


Figure 10: ResNet-50 Layer vs. Minimum salient weights per neuron. SRigL sets the minimum salient weight per neuron to 1 if the product between γ_{sal} and the sparse fan-in per neuron is less than 1. Therefore, even in a relatively large network such as ResNet50 many of the layers only require that a single weight be active to keep the neuron active. We believe this is why SRigL's performance is relatively invariant to various ablation thresholds when applied to CNNs

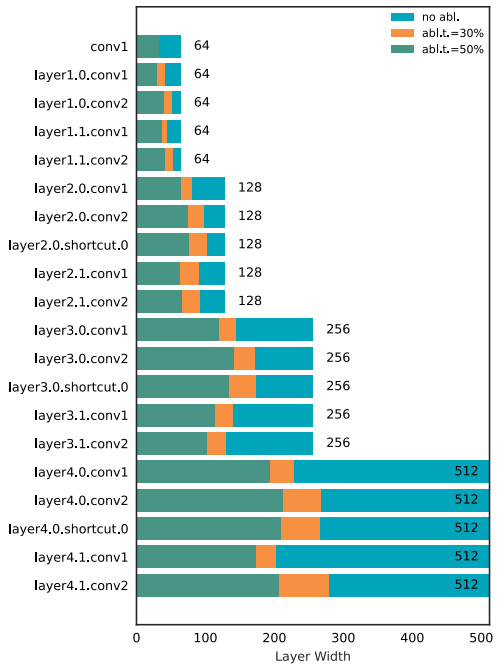


Figure 11: ResNet-18/CIFAR-10 layer widths at the end of training at 99% sparsity. Without ablation, constant fan-in constraint enforces that sparse layers retain their original width. When ablation is enabled, the γ_{sal} threshold (minimum percentage salient weights per neuron) is used to control the amount of ablation.

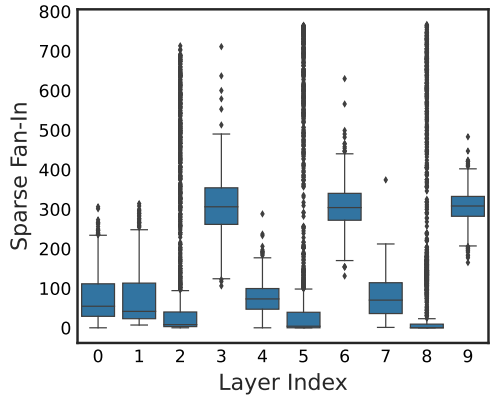


Figure 12: Sparse Fan-In vs. ViT-B/16 layer index at the end of training with RigL at 90% sparsity. Only the first 10 layers are shown for clarity. We find that RigL learns a sparse connectivity with large variance in fan-in between neurons within the same layer with some neurons receiving up to $\times 10$ the number of active connections than the mean for the same layer.

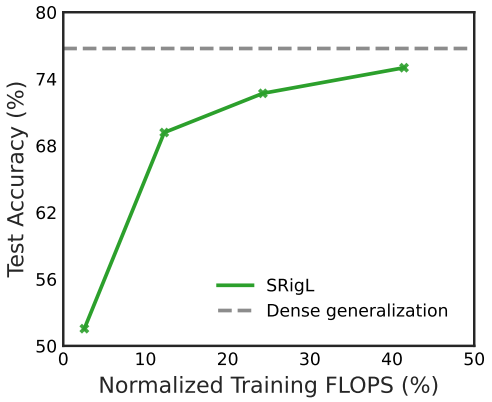


Figure 13: Training FLOPs for SRigL on ResNet-50/ImageNet at a variety of sparsities compared with dense generalization. FLOPs are normalized by dense training FLOPs.

G FLOPs ANALYSIS

In Fig. 13, we present an analysis of the FLOPs required during training and inference for SRigL and compare with SR-STE. We calculate FLOPs using the same methodology as Evci et al. (2021) by considering only operations induced by convolutional and linear layers and their activations. FLOPs for add and pooling operations are ignored. For training FLOPs, we also disregard FLOPs required for mask updates, as this step is amortized over ΔT steps and is negligible compared to the FLOPs required otherwise for training. The open-source code for counting operations is from the NeurIPS 2019 MicroNet Challenge and is available on GitHub[2].

Similar to other DST methods, SRigL obtains generalization performance comparable to a dense network benchmark at a fraction of the FLOPs required for both training and inference.

H IN TIME OVERPARAMETERIZATION RATES

In Figs. 14 to 17 we present the In Time Overparameterization Rate (ITOP) (Liu et al., 2021c) for various models and datasets. In this same work, Liu et al. (2021c) proposed modified hyperparameters for RigL that may yield higher generalization performance; however, a detailed investigation of these hyperparameters for SRigL is left to future work.

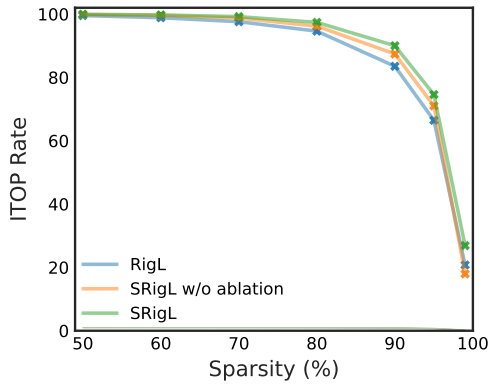


Figure 14: ResNet-18/CIFAR-10 ITOP rate

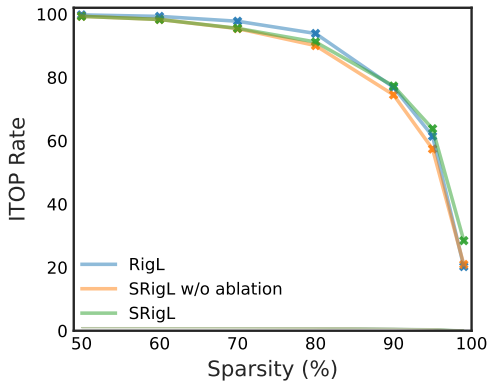


Figure 15: ResNet-18/CIFAR-10 ITOP rate

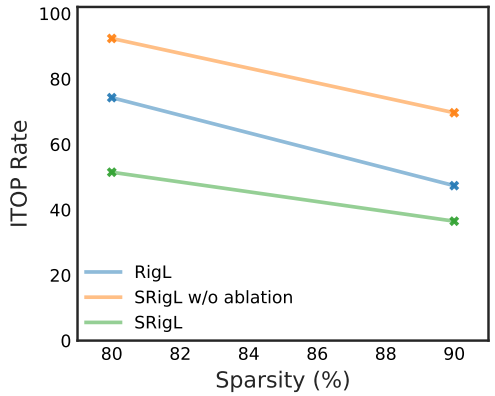


Figure 16: ViT-B/16/ImageNet ITOP Rate

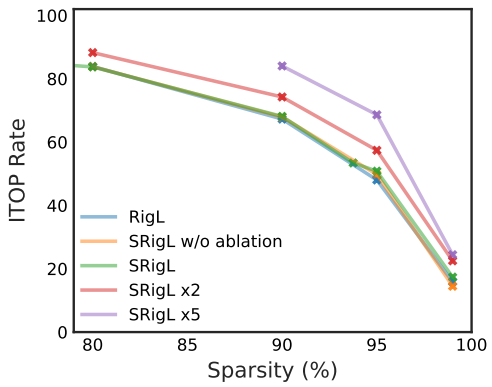


Figure 17: ResNet-50/ImageNet ITOP Rate

[2]MicroNet Challenge Github Repository

I CONDENSED LINEAR CPU BENCHMARK DETAILS

For each sparsity level, we used the trained weights from the last linear layer in the final multi-layer perception block from the ViT-B/16 transformer encoder. This layer has a width of 768 neurons and an input of 3072 features. The input and layer parameters are all set to a 32 bit floating point type. Across all sparsities, batch sizes, and number of threads investigated, our condensed representation utilizing both structured and fine-grained sparsity yields the fastest online inference speed. However, at higher batch sizes and modest sparsities, structured sparsity is often faster than our condensed representation. See Figs. 18 to 20 for benchmark results from 1-8 threads and batch sizes 1-64. We note that SRigL with either a condensed or a structured sparse representation yields the fastest benchmark times.

We used `torch.compile` with the inductor backend. For compiler options, we used the max-autotune mode and full graph output. However, full graph output is not compatible with CSR formats so we omit this parameter for the unstructured benchmarks. The benchmark script was run with a `niceness` value of -15 to ensure as accurate results as possible. The apparent slow down in 99% structured sparse benchmarks compared to other sparsities is due to the fact that SRigL ablates fewer neurons at 99% sparsity. At extreme sparsities, each neuron has very few active weights resulting in more neurons being considered as *salient* by SRigL.

J GPU BENCHMARKS

Using GPU CUDA kernels developed by (Schultheis & Babbar, 2023), we accelerate our sparse networks and demonstrate a significant acceleration for batched inference and a modest acceleration for online inference at high sparsities (>90%), see Fig. 21. All runs conducted on an NVIDIA Titan V. Note y-axis scale is logarithmic.

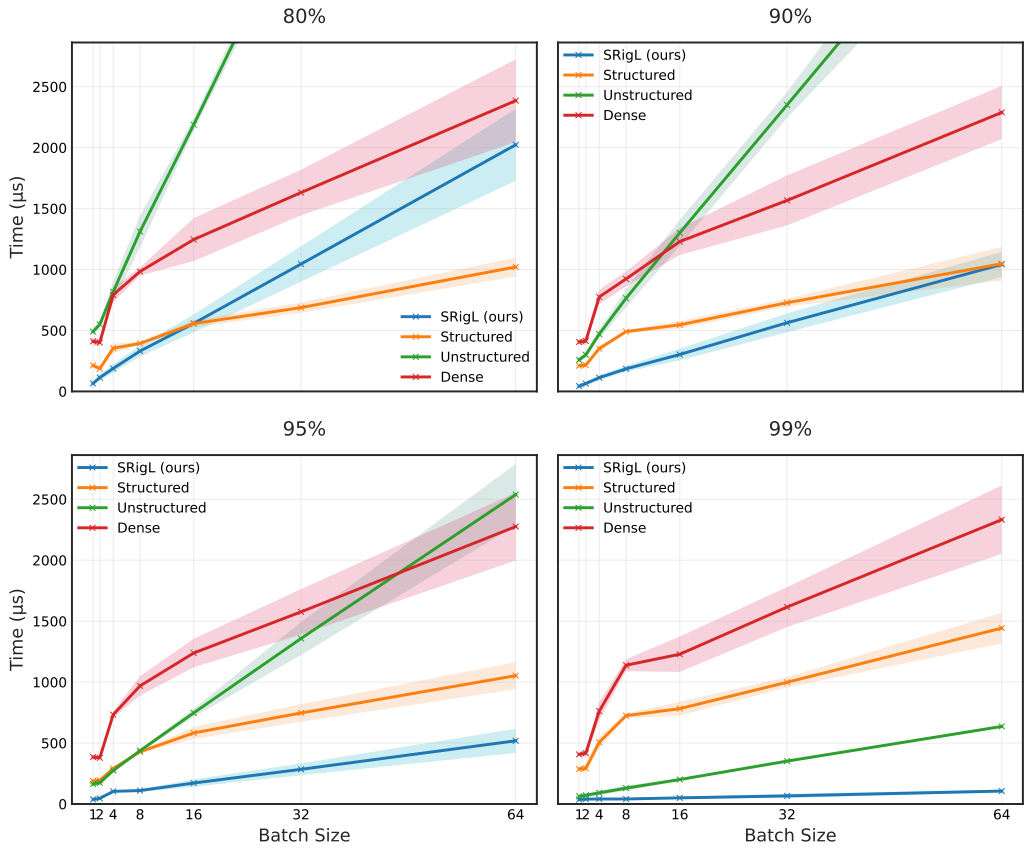


Figure 18: CPU benchmarks with 1 thread up to batch size 64

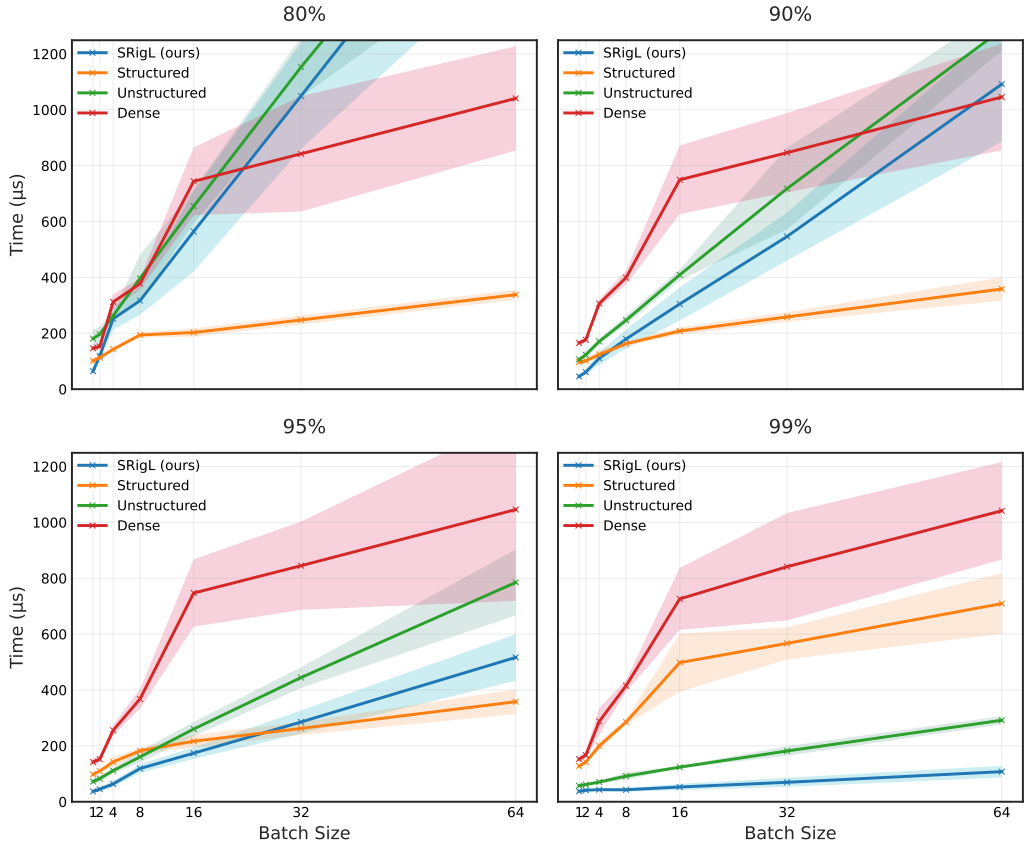


Figure 19: CPU benchmarks with 4 threads up to batch size 64

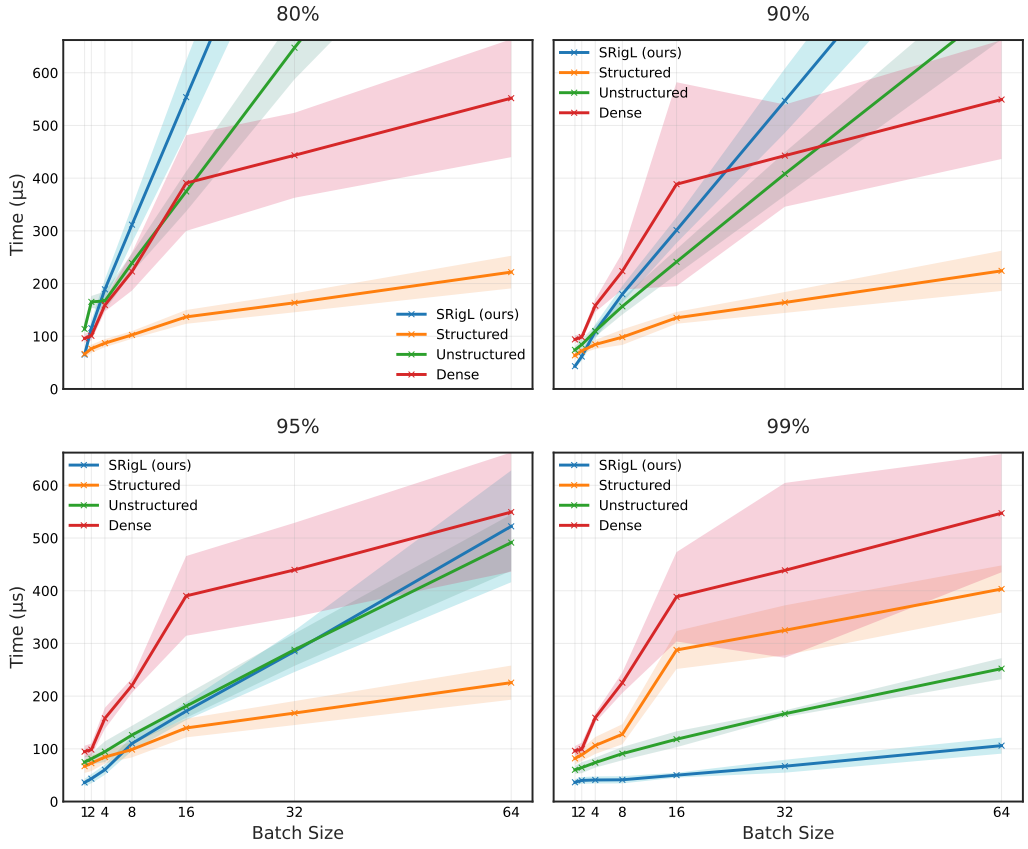
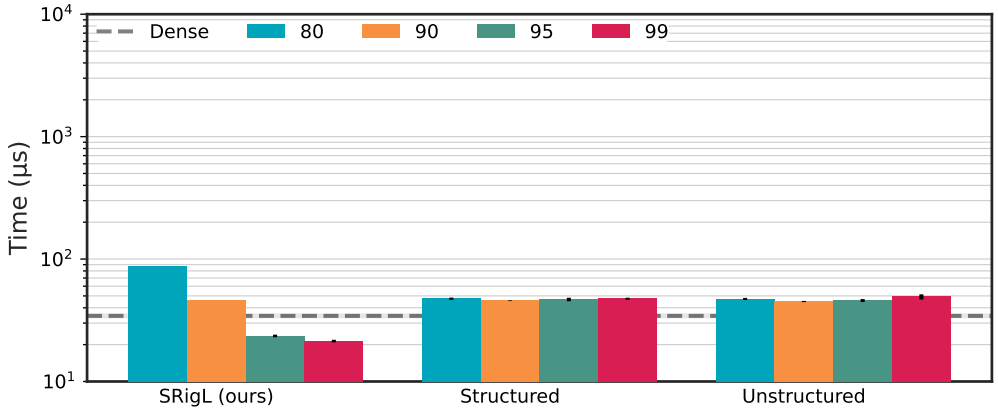
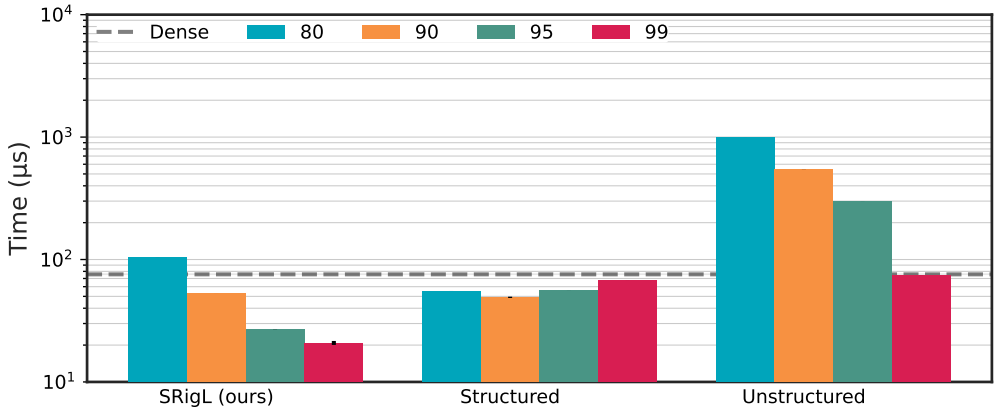


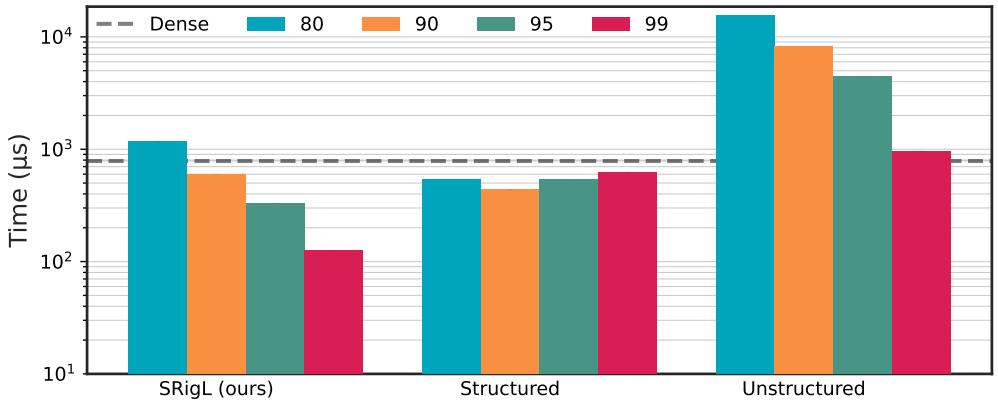
Figure 20: CPU benchmarks with 8 threads up to batch size 64



(a) GPU online inference (batch size of 1)



(b) GPU batched inference with batch size of 128



(c) GPU batched inference with batch size of 2048

Figure 21: Real-world GPU wall-clock timings for inference on an NVIDIA Titan V. We compare timings for a fully-connected layer extracted from the ViT-B/16 model trained with SRigL when compressed using the condensed representation learned by SRigL, structured (i.e. SRigL with only neuron ablation) and unstructured (i.e. CSR) representations. Batch sizes are 1, 256, and 2048 for sub-figures 21a, 21b, 21c, respectively. The median over a minimum of 5 runs is shown, while the error bars show the std. dev. Note: y-axis scale is logarithmic

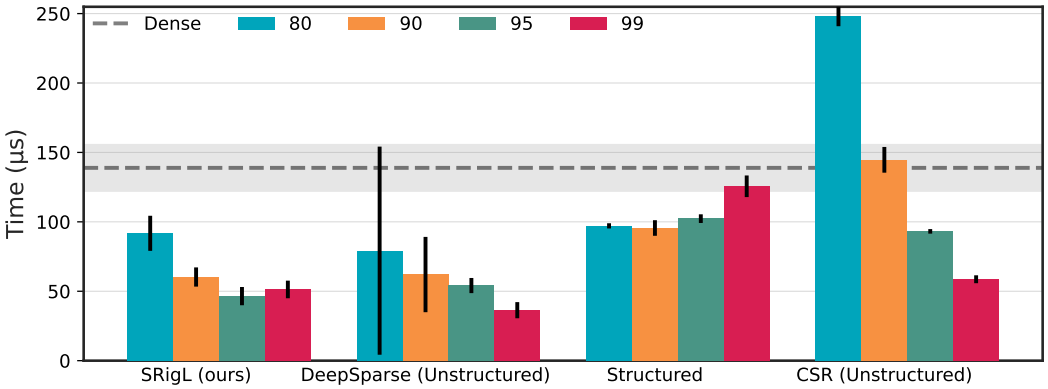


Figure 22: Online inference with DeepSparse compared to SRigL on an Intel Xeon W-2145 with 4 threads. The median over a minimum of 5 runs is shown, while the error bars show the std. dev.

K DeepSparse CPU Benchmarks

Here we present online inference benchmarks for CPU using the DeepSparse Engine library (Iofinova et al., 2021). DeepSparse library includes several engineering innovations to accelerate unstructured sparsity on CPU. For instance, a *depth-wise asynchronous* execution algorithm is used that takes advantage of the relatively large cache size for CPUs compared to hardware accelerators such as GPUs. Other additional innovations used include pre-loading the input data to hide latency via CPU *pipelining*, compressing sparse activations into a CSR format on-the-fly, and keeping convolutional kernels in L2 cache. For more details see Kurtz et al. (2020).

We compare our CPU timings for SRigL to DeepSparse in Fig. 22 and find similar latency; however, we note that DeepSparse is subject to a higher variability as evidenced by a larger standard deviation. Further, many of the innovations used to accelerate unstructured sparse networks with DeepSparse could equally be applied to networks trained with SRigL.

L Comparison with Structured Pruning Methods

In the following table we compare several structured pruning methods to SRigL. The tabulated structured pruning methods typically prune and fine-tune a pretrained model, resulting in extended training duration compared to typical dense training. We report the inference FLOPs, top-1 accuracy, and number of epochs for each method in Table 10.

Table 10: Top-1 ImageNet test accuracy of ResNet-50 for various structured pruning methods compared with SRigL and Chase (Yin et al., 2023). All values, except for SRigL, are obtained from Yin et al. (2023).

Methods	Inference FLOPs	Top-1 Accuracy	Epochs
Uniform	2.0G	75.1%	300
Random	2.0G	74.6%	300
GBN (You et al., 2019)	2.4G	76.2%	350
LEGR (Chin et al., 2020)	2.4G	75.7%	-
FPGM (He et al., 2019)	2.4G	75.6%	200
TAS (Dong & Yang, 2019)	2.3G	76.2%	240
Hrank (Lin et al., 2020)	2.3G	75.0%	570
SCOP (Tang et al., 2020)	2.2G	76.0%	230
CHIP (Sui et al., 2021)	2.2G	76.3%	-
Group Fisher (Liu et al., 2021a)	2.0G	76.4%	-
AutoSlim (Yu & Huang, 2019)	2.0G	75.6%	-
CafeNet-R (Su et al., 2021)	2.0G	76.5%	300
Chase-1†(Yin et al., 2023)	1.5G	76.6%	250
SRigL†	2.0G	74.7%	205
SRigL†	2.0G	76.2%	515
Uniform	1.0G	73.1%	300
Random	1.0G	72.2%	300
Group Fisher (Liu et al., 2021a)	1.0G	73.9%	-
CafeNet-R (Su et al., 2021)	1.0G	74.9%	300
CafeNet-E (Su et al., 2021)	1.0G	75.3%	300
Chase-2†(Yin et al., 2023)	0.9G	75.7%	250
SRigL†	1.0G	71.5%	205
SRigL†	1.0G	73.6%	515

† DST methods. All other methods tabulated are structured pruning methods.