Microsoft Research

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

AI2

Soft Threshold Weight Reparameterization for Learnable Sparsity

Aditya Kusupati

Vivek Ramanujan*, Raghav Somani*, Mitchell Wortsman*

Prateek Jain, Sham Kakade and Ali Farhadi

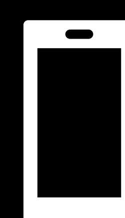
Motivation

- Deep Neural Networks
  - Highly accurate
  - Millions of parameters & Billions of FLOPs
  - Expensive to deploy

- Sparsity
  - Reduces model size & inference cost
  - Maintains accuracy
  - Deployment on CPUs & weak single-core devices

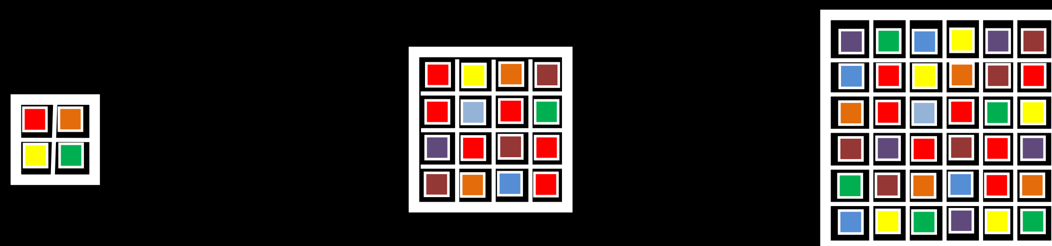
Privacy preserving smart glasses

Billions of mobile devices



Motivation

- Existing sparsification methods
  - Focus on model size vs accuracy – *very little on inference FLOPs*
  - Global, uniform or heuristic sparsity budget across layers



	Layer 1	Layer 2	Layer 3	Total
# Params	20	100	1000	1120
FLOPs	100K	100K	50K	250K
<u>Sparsity – Method 1</u>				
# Params	20	100	100	220
FLOPs	100K	100K	5K	205K
<u>Sparsity – Method 2</u>				
# Params	10	10	200	220
FLOPs	50K	10K	10K	70K

Motivation

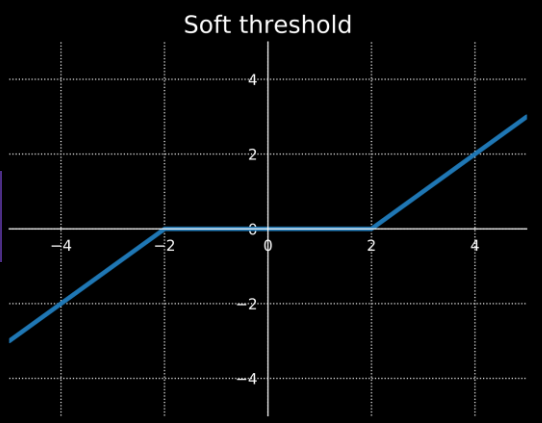
- Non-uniform sparsity budget – *Layer-wise*
  - Very hard to search in deep networks
  - Sweet spot – *Accuracy vs FLOPs vs Sparsity*
  - Existing techniques
    - Heuristics – *increase FLOPs*
    - Use RL – *expensive to train*

“Can we design a robust efficient method to learn non-uniform sparsity budget across layers?”

Overview

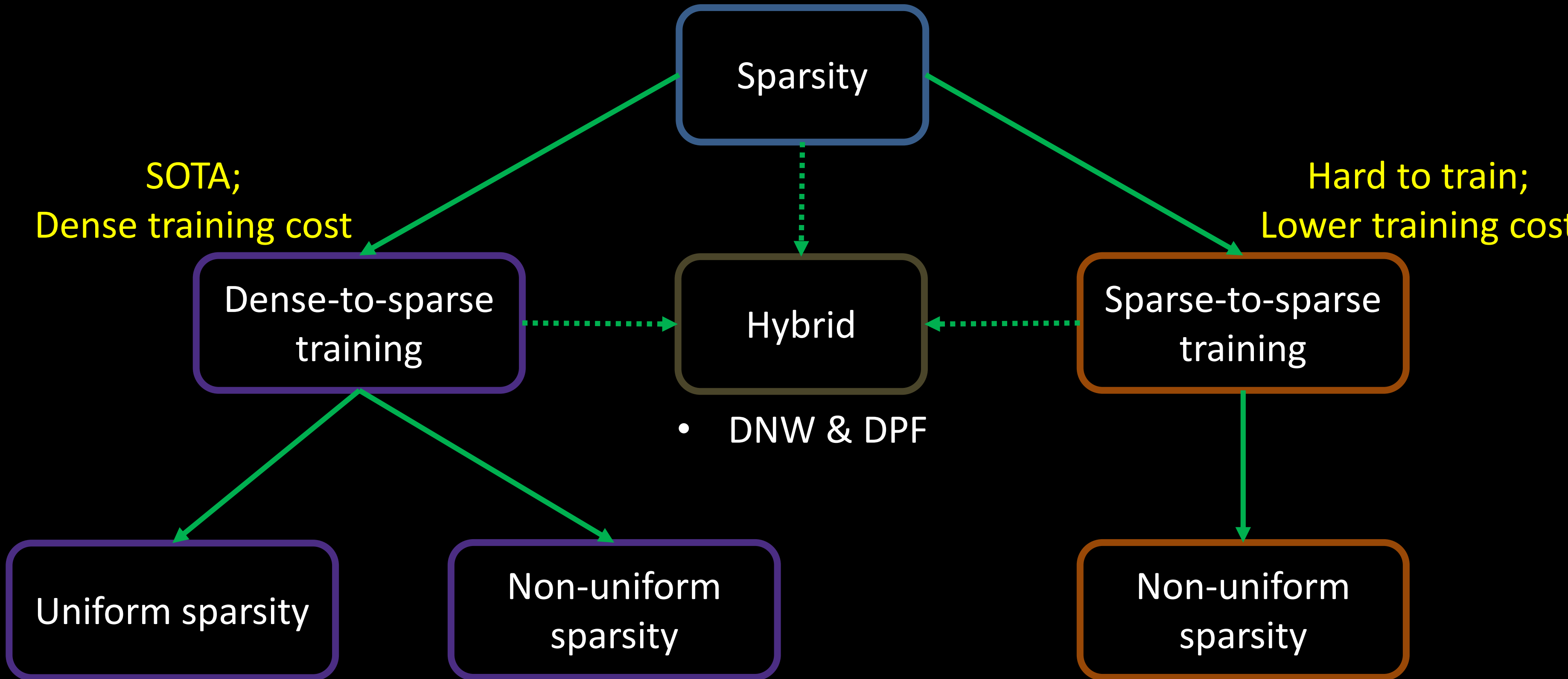
- **STR** – **S**oft **T**hreshold **R**eparameterization

$$STR(\mathbf{W}_l, \alpha_l) = \text{sign}(\mathbf{W}_l) \cdot \text{ReLU}(|\mathbf{W}_l| - \alpha_l)$$



- Learns layer-wise non-uniform sparsity budgets
  - Same model size; Better accuracy; Lower inference FLOPs
  - SOTA on ResNet50 & MobileNetV1 for ImageNet-1K
  - Boosts accuracy by up to 10% in ultra-sparse (98-99%) regime
- Extensions to structured, global & per-weight (mask-learning) sparsity

Existing Methods

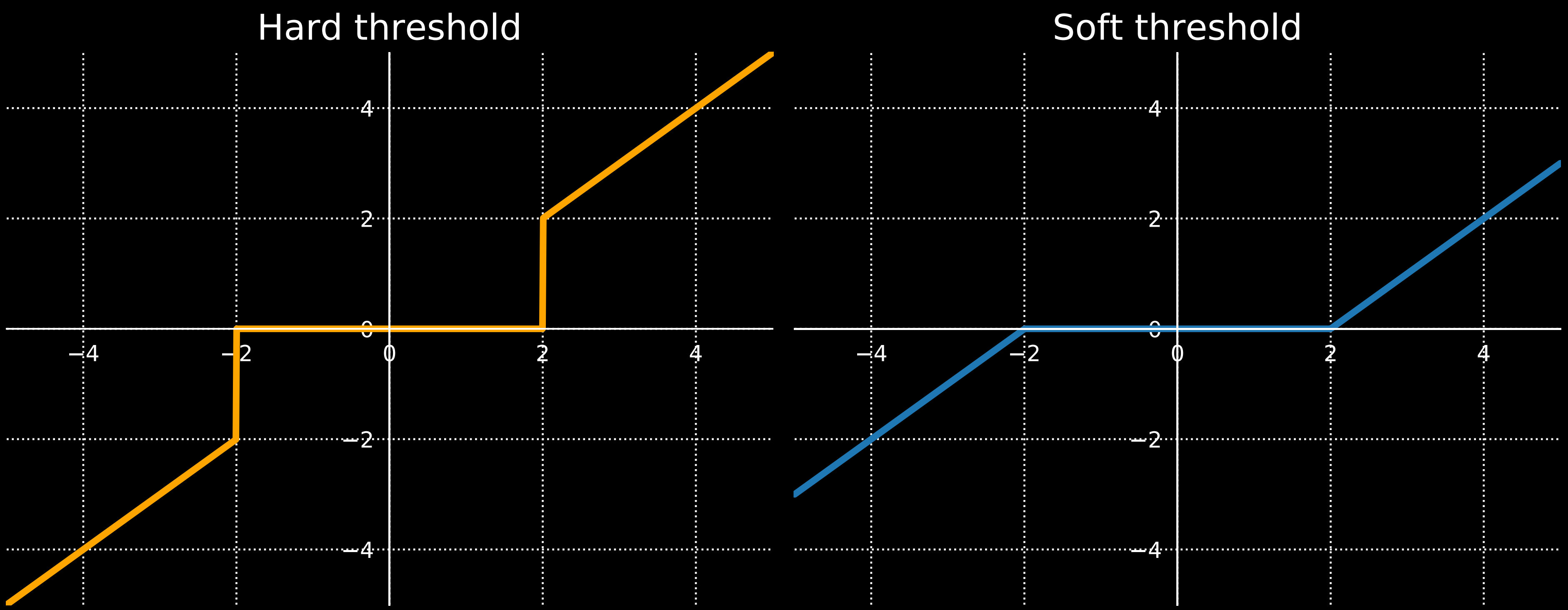


- Gradual Magnitude Pruning (GMP)

- Heuristics – *ERK*
- Global Pruning/Sparsity
- STR - some gains from sparse-to-sparse

- DSR, SNFS, RigL etc.,
- Heuristics – *ERK*
- Re-allocation using magnitude/gradient

STR - Method

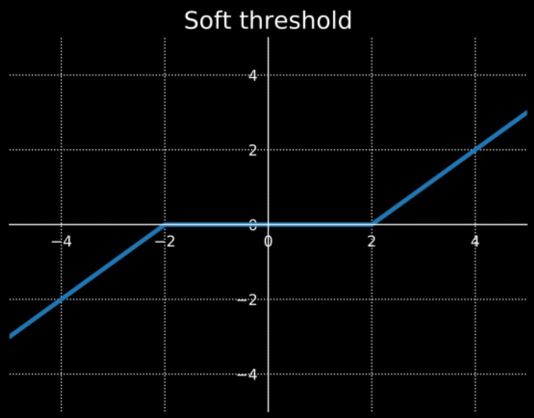


$$\alpha = 2$$

$$HT(x,\alpha) = \begin{cases} x; & |x| > \alpha \\ 0; & |x| \le \alpha \end{cases}$$

$$ST(x,\alpha) = \begin{cases} x - \alpha; & x > \alpha \\ 0; & |x| \le \alpha \\ x + \alpha; & x < -\alpha \end{cases}$$

STR - Method



$$ST(x, \alpha) = \text{sign}(x) \cdot \text{ReLU}(|x| - \alpha)$$
$$= \text{sign}(x) \cdot \text{ReLU}(|x| - g(s))$$

L-layer DNN, $\mathcal{W} = [\mathbf{W}_l]_{l=1}^{L}, \mathbf{s} = [s_l]_{l=1}^{L}$ and a function $g(.)$

$$\boxed{\mathcal{S}_g(\mathbf{W}_l, s_l) = \text{sign}(\mathbf{W}_l) \cdot \text{ReLU}(|\mathbf{W}_l| - g(s_l))}$$

$$\mathcal{W} \leftarrow \mathcal{S}_g(\mathcal{W}, \mathbf{s})$$

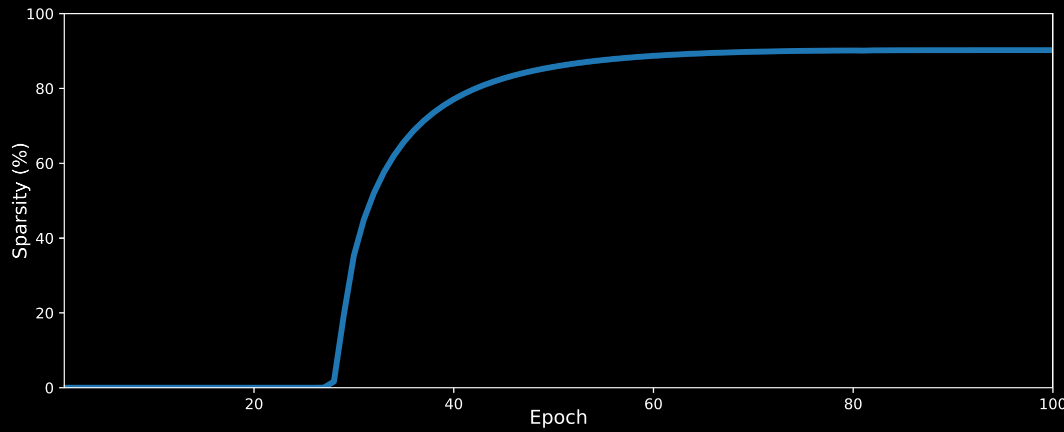
STR - Training

$$\min_{\mathcal{W},\mathbf{s}} \mathcal{L}\left(\boxed{\mathcal{S}_g(\mathcal{W},\mathbf{s})}, \mathcal{D}\right) + \lambda \sum_{l=1}^{L} \left(|\mathbf{W}_l|_2^2 + |s_l|_2^2\right)$$

- Regular training with reparameterized weights $\mathcal{S}_g(\mathcal{W},\mathbf{s})$
- Same weight-decay parameter (λ) for both $(\mathcal{W},\mathbf{s})$
  - Controls the overall sparsity
- Initialize s; $g(s) \approx 0$
  - Finer sparsity and dense training control
- Choice of $g(.)$
  - *Unstructured sparsity*: Sigmoid
  - *Structured sparsity*: Exponential

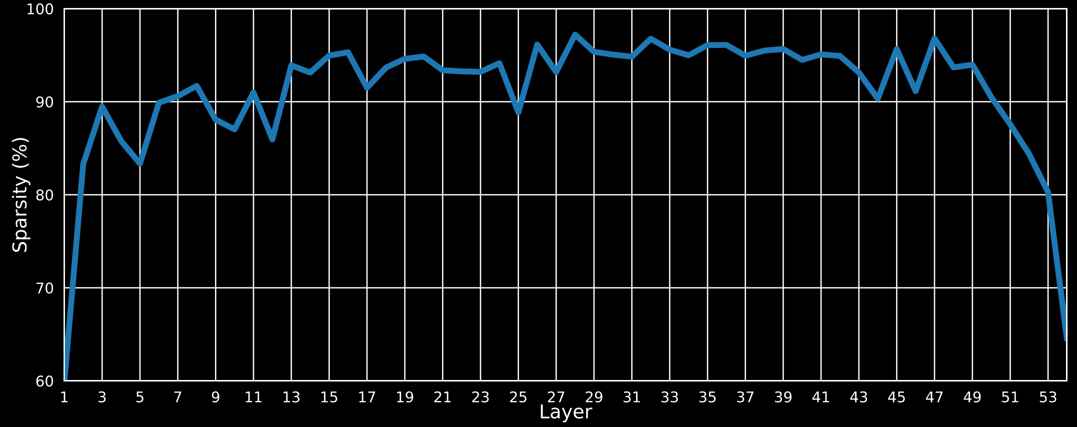
STR - Training

- STR learns the SOTA hand-crafted heuristic of GMP



Overall sparsity vs Epochs – 90% sparse ResNet50 on ImageNet-1K

- STR learns diverse non-uniform layer-wise sparsities



Layer-wise sparsity – 90% sparse ResNet50 on ImageNet-1K

STR - Experiments

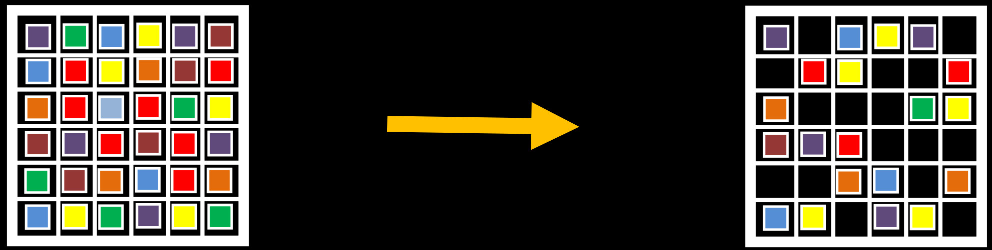
- Unstructured sparsity - CNNs
  - *Dataset*: ImageNet-1K
  - *Models*: ResNet50 & MobileNetV1
  - *Sparsity range*: 80 - 99%
    - Ultra-sparse regime: 98 - 99%

- Structured sparsity – Low rank in RNNs
  - *Datasets:* Google-12 (keyword spotting), HAR-2 (activity recognition)
  - *Model*: FastGRNN

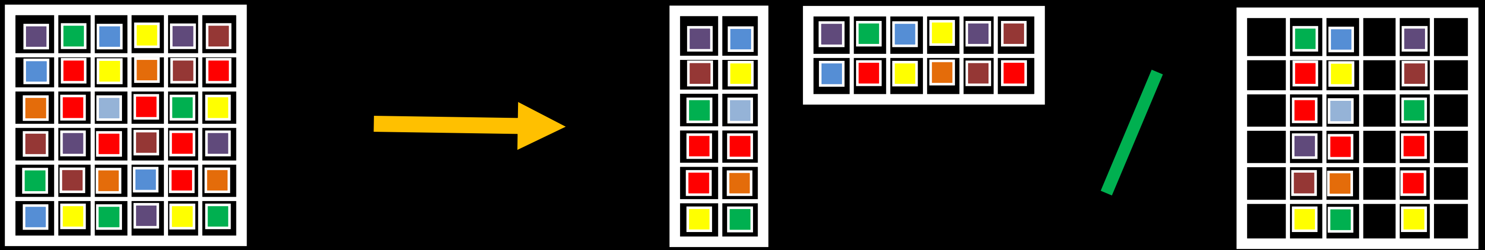
- Additional
  - Transfer of learnt budgets to other sparsification techniques
  - STR for global, per-weight sparsity & filter/kernel pruning

Unstructured vs Structured Sparsity

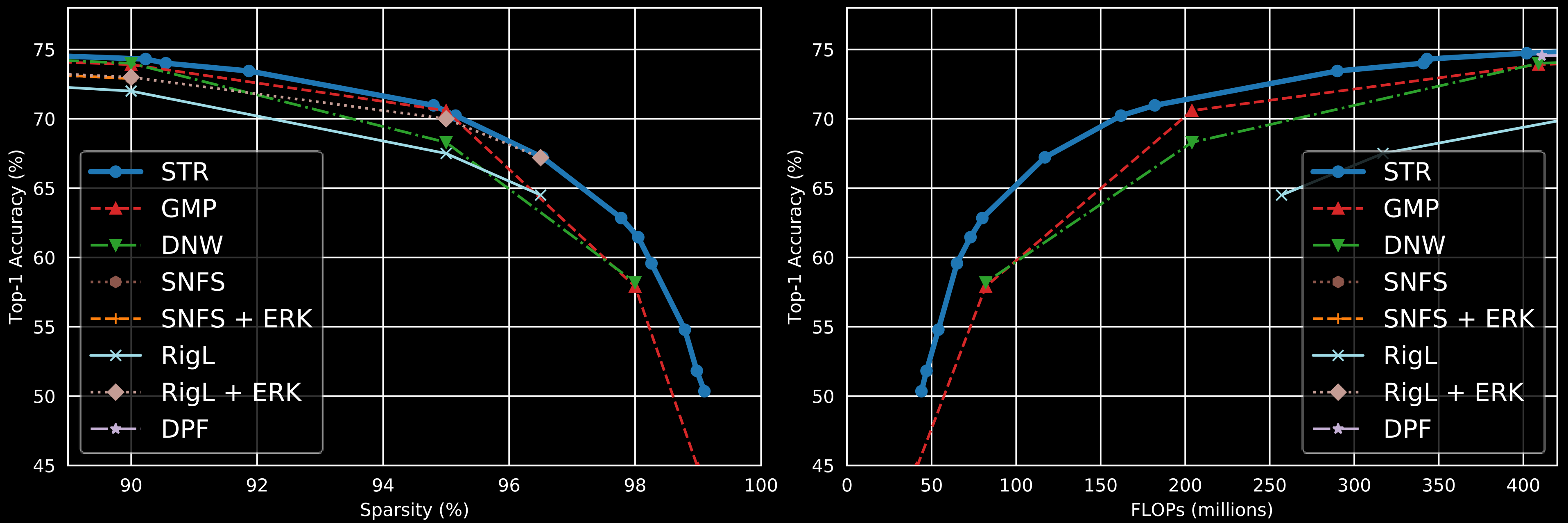
- Unstructured sparsity
  - Typically magnitude based pruning with global or layer-wise thresholds



- Structured sparsity
  - Low-rank & neuron/filter/kernel pruning

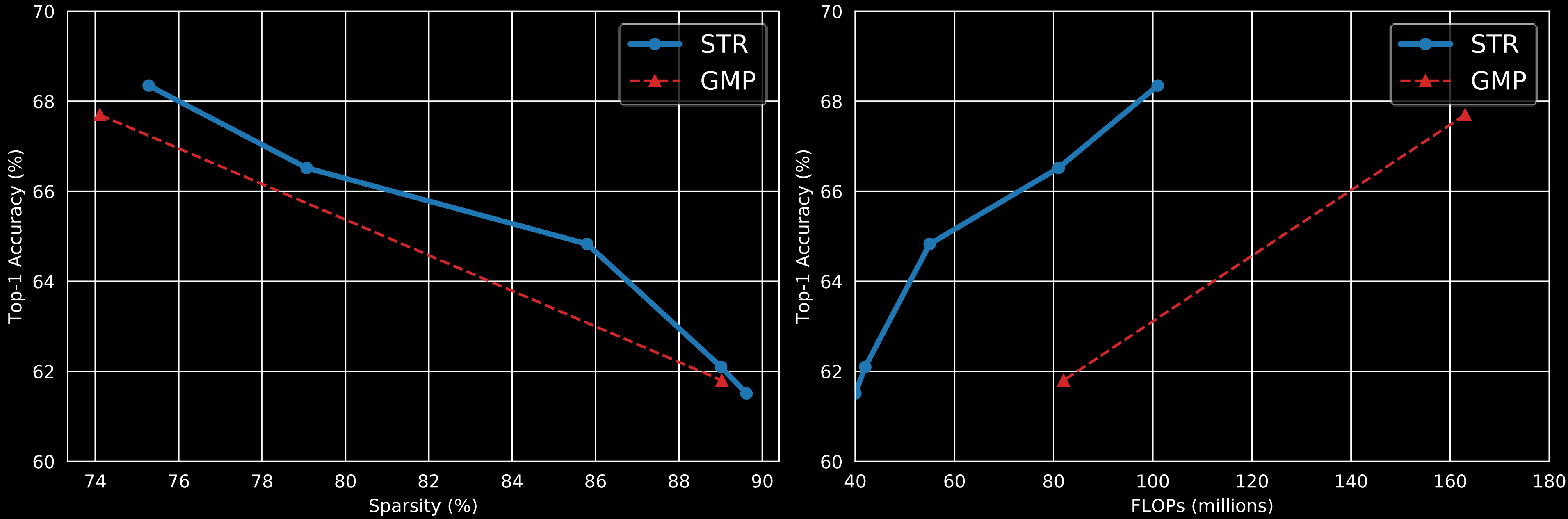


STR Unstructured Sparsity: ResNet50



- STR requires 20% lesser FLOPs with same accuracy for 80-95% sparsity
- STR achieves 10% higher accuracy than baselines in 98-99% regime

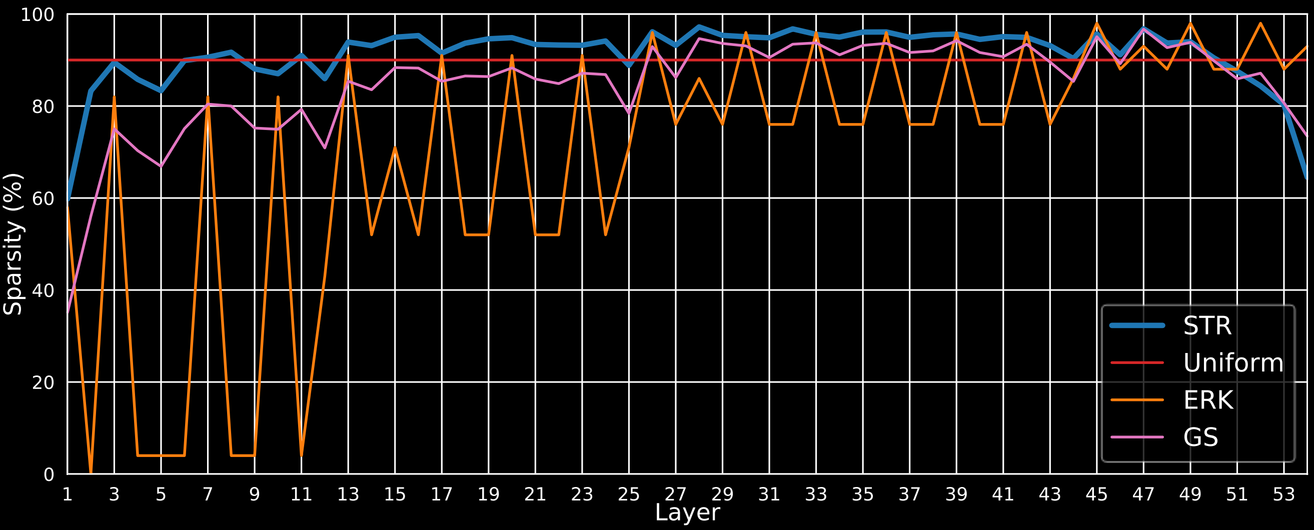
STR Unstructured Sparsity: MobileNetV1

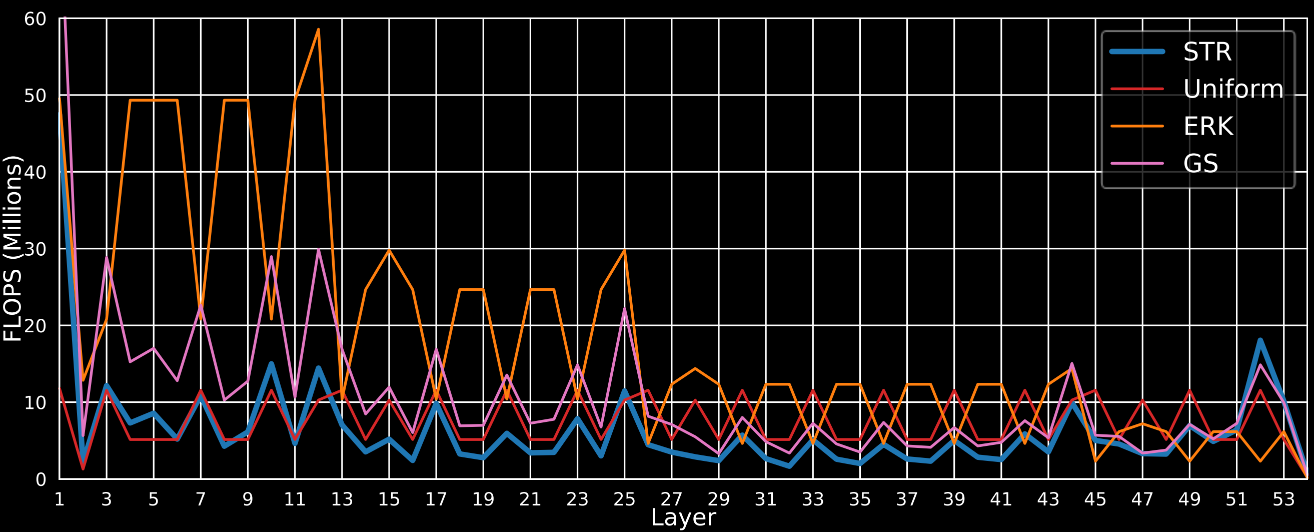


- STR maintains accuracy for 75% sparsity with 62M lesser FLOPs
- STR has ∼50% lesser FLOPs for 90% sparsity with same accuracy

STR Sparsity Budget: ResNet50

Layer-wise sparsity and FLOPs budgets for 90% sparse ResNet50 on ImageNet-1K

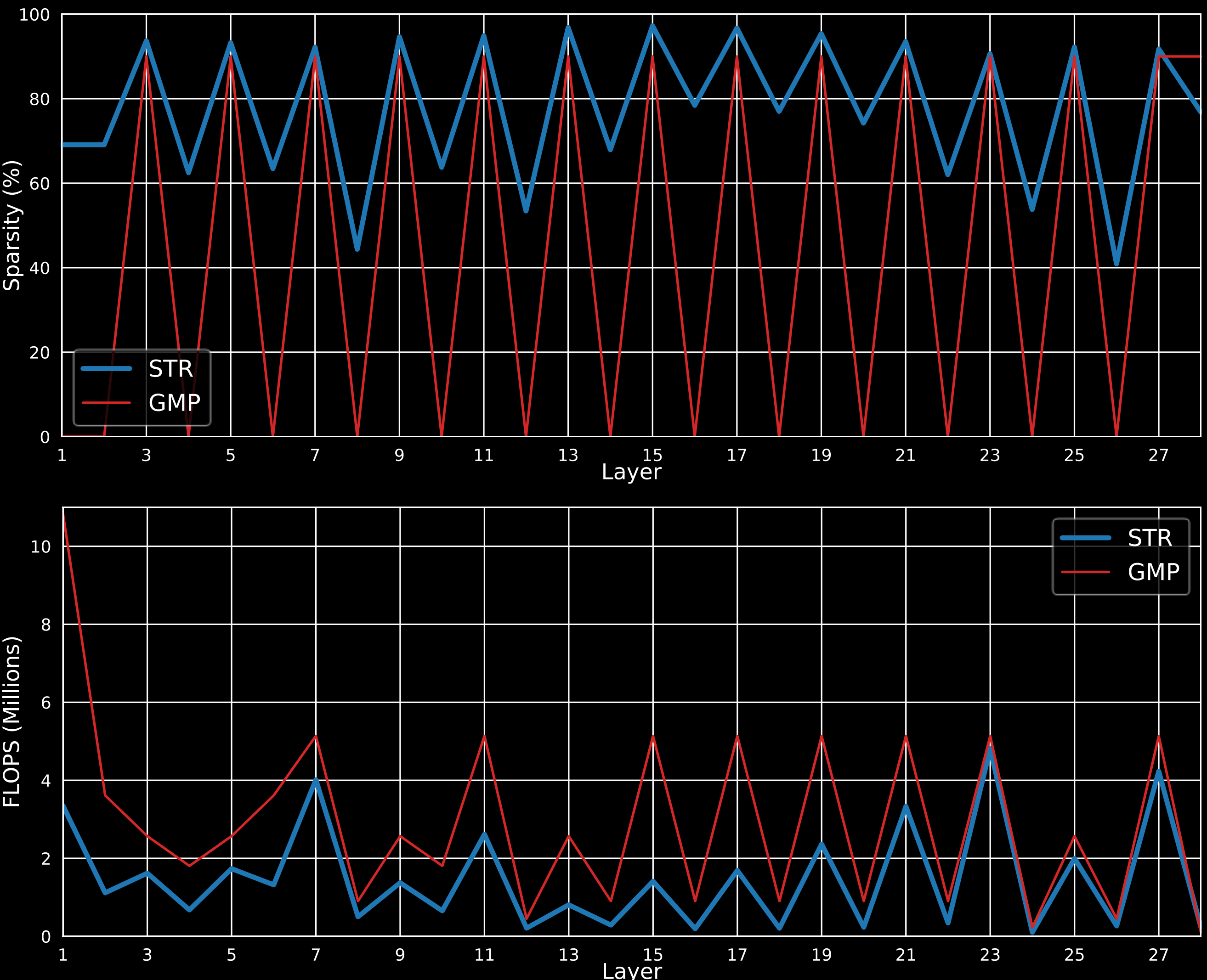




- STR learns sparser initial layers than the non-uniform sparsity baselines
- STR makes last layers denser than all baselines
- STR produces sparser backbones for transfer learning
- STR adjusts the FLOPs across layers such that it has lower total inference cost than the baselines

STR Sparsity Budget: MobileNetV1

Layer-wise sparsity and FLOPs budgets for 90% sparse MobileNetV1 on ImageNet-1K



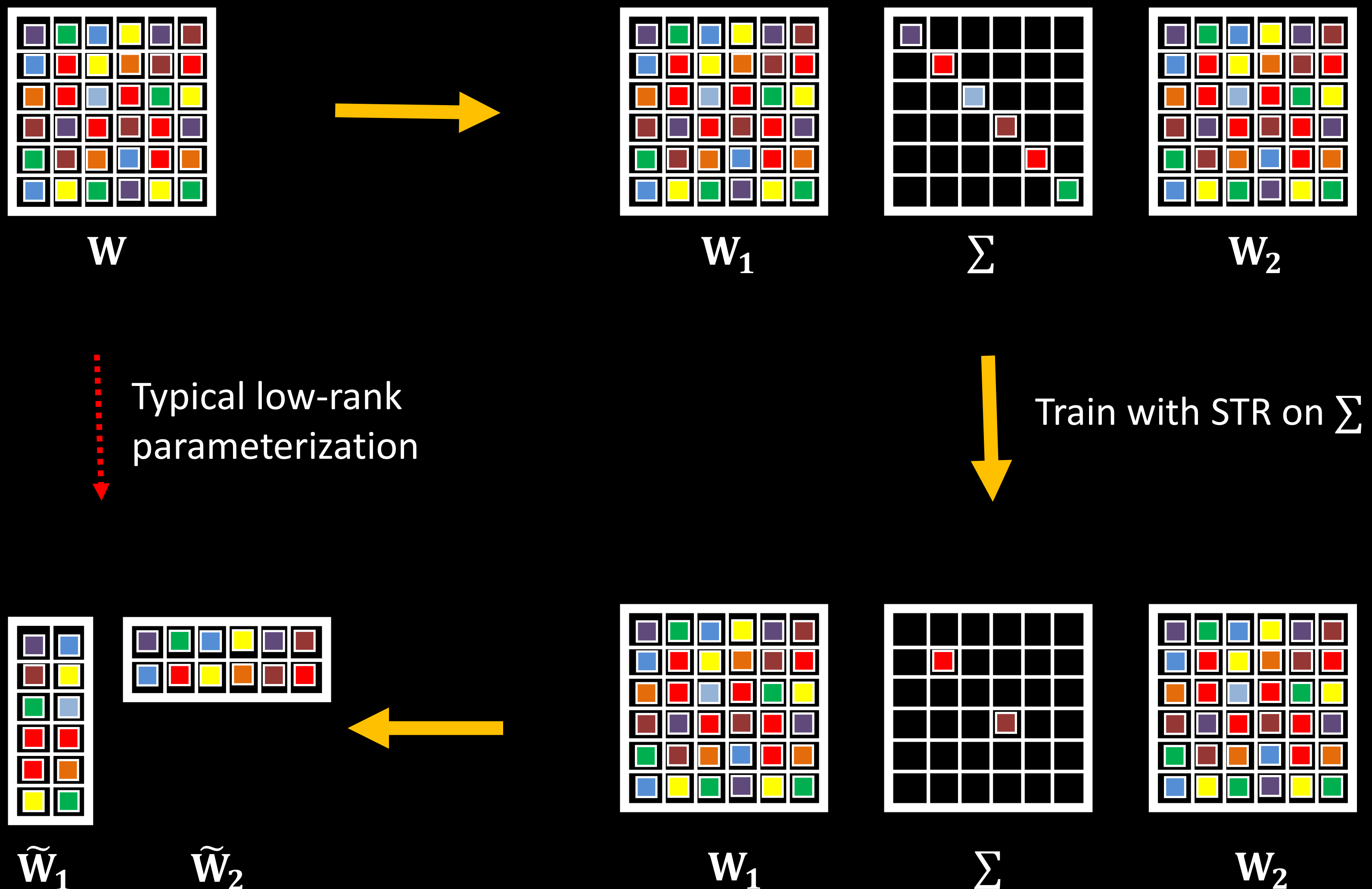
- STR automatically keeps depth-wise separable conv layers denser than rest of the layers
- STR's budget results in 50% lesser FLOPs than GMP

STRConv

Algorithm 1 PyTorch code for STRConv with per-layer threshold.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

from args import args as parser_args

def softThreshold(x, s, g=torch.sigmoid):
   # STR on a weight x (can be a tensor) with "s" (typically a scalar, but can be a tensor) with function "g".
   return torch.sign(x)*torch.relu(torch.abs(x)-g(s))

class STRConv(nn.Conv2d): # Overloaded Conv2d which can replace nn.Conv2d
   def __init__(self, *args, **kwargs):
      super().__init__(*args, **kwargs)
      # "g" can be chosen appropriately, but torch.sigmoid works fine.
      self.g = torch.sigmoid
      # parser_args gets arguments from command line. sInitValue is the initialization of "s" for all layers. It
          can take in different values per-layer as well.
      self.s = nn.Parameter(parser_args.sInitValue*torch.ones([1, 1]))
      # "s" can be per-layer (a scalar), global (a shared scalar across layers), per-channel/filter (a vector)
         or per individual weight (a tensor of the size self.weight). All the experiments use per-layer "s" (a
         scalar) in the paper.

   def forward(self, x):

      self.sparseWeight = softThreshold(self.weight, self.s, self.g)
      # Parameters except "x" and "self.sparseWeight" can be chosen appropriately. All the experiments use
         default PyTorch arguments.
      x = F.conv2d(x, self.sparseWeight, self.bias, self.stride, self.padding, self.dilation, self.groups)

      return x

# FC layer is implemented as a 1x1 Conv2d and STRConv is used for FC layer as well.
```

STR Structured Sparsity: Low rank



STR – Critical Design Choices

- Weight-decay λ
  - Controls overall sparsity
  - Larger $\lambda \rightarrow$ higher sparsity at the cost of some instability

- Initialization of s_l
  - Controls finer sparsity exploration
  - Controls duration of dense training

- Careful choice of $g(.)$
  - Drives the training dynamics
  - Better functions which consistently revive dead weights

STR - Conclusions

- STR enables stable end-to-end training (with no additional cost) to obtain sparse & accurate DNNs
- STR efficiently learns per-layer sparsity budgets
  - Reduces FLOPs by up to 50% for 80-95% sparsity
  - Up to 10% more accurate than baselines for 98-99% sparsity
  - Transferable to other sparsification techniques
- Future work
  - Formulation to explicitly minimize FLOPs
  - Stronger guarantees in standard sparse regression setting
- Code, pretrained models and sparsity budgets available at

https://github.com/RAIVNLab/STR

Aditya

Vivek*

Raghav*

Mitchell*

Thank You

Prateek

Sham

Ali