Randomized linear algebra for inversion

Mathias Louboutin

ML4Seismic Partners Meeting

SLIM
Georgia Institute of Technology

ML4Seismic

Georgia Tech College of Computing
School of Computational Science and Engineering

Georgia Tech College of Sciences
School of Earth and Atmospheric Sciences

Georgia Tech College of Engineering
School of Electrical and Computer Engineering

Motivation

High-memory footprint adjoint-state methods

Computationally expensive checkpointing

Case specific/internal solutions to manage memory

- Fourier (BP patent)
- Compression (No existing GPU porting)
- Serialization/Disk (High IO)
- Boundary methods (reversible only)
- …

Rajiv Kumar, Marie Graff-Kray, Ivan Vasconcelos, and Felix J. Herrmann, "Target-oriented imaging using extended image volumes—a low-rank factorization approach", Geophysical Prospecting, vol. 67, pp. 1312-1328, 2019
Mengmeng Yang, Marie Graff, Rajiv Kumar, and Felix J. Herrmann, "Low-rank representation of omnidirectional subsurface extended image volumes", Geophysics, vol. 86, pp. 1-41, 2021.
Mathias Louboutin, Ali Siahkoohi, Rongrong Wang, and Felix J. Herrmann, "Low-memory stochastic backpropagation with multi-channel randomized trace estimation". 2021.
Philipp A. Witte, Mathias Louboutin, Fabio Luporini, Gerard J. Gorman, and Felix J. Herrmann, "Compressive least-squares migration with on-the-fly Fourier transforms", Geophysics, vol. 84, pp. R655-R672, 2019.

Solutions

Take advantage of large scale randomized linear algebra

Leverage our work on full-subsurface offset Image Volumes

Build on lessons learned form machine learning (convolutional layers)

According to stochastic optimization

- inaccurate gradients can still lead to accurate inversion
- undergirds our compressive imaging

Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." *SIAM review* 53.2 (2011): 217-288.

SLIM ML4Seismic

Randomized linear algebra

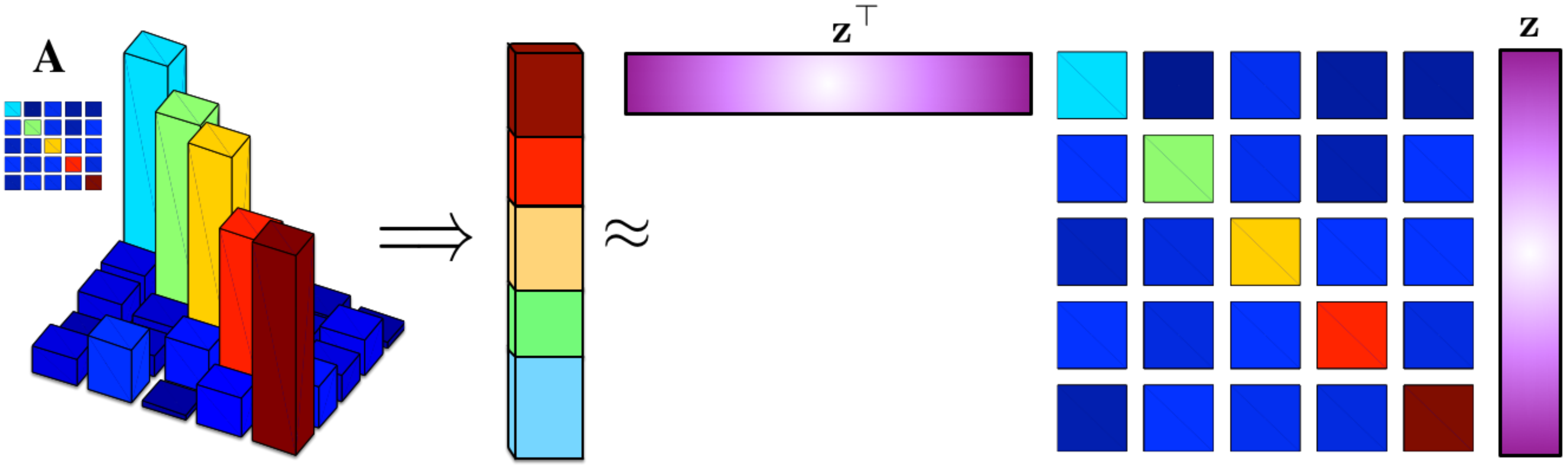
Randomized SVD:

$$\mathbf{A} \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top \quad \text{with} \quad \begin{cases} \left[\mathbf{Q}, \sim\right] & = \mathrm{qr}(\mathbf{A}\mathbf{Z}) \\ \left[\widetilde{\mathbf{U}}, \mathbf{S}, \mathbf{V}\right] & = \mathrm{svd}(\mathbf{Q}^\top \mathbf{A}) \\ \mathbf{U} = \mathbf{Q}\widetilde{\mathbf{U}} & \end{cases}$$

- information is reaped during probing $\mathbf{A}\mathbf{Z}$ w/ $\mathbf{Z} = [\mathbf{z}_1, \cdots, \mathbf{z}_r]$ random
- only need access to action of $\mathbf{A}$ (in parallel)
- memory friendly
- approximation w/ accuracy $\propto r \ll N$, # of sketches w/ random vectors $\mathbf{z}_i$

SLIM ML4Seismic

Random Trace Estimation



$$\mathrm{tr}(\mathbf{A}) \approx \frac{1}{r}\sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A}\mathbf{z}_j = \frac{1}{r}\mathrm{tr}(\mathbf{Z}^\top \mathbf{A}\mathbf{Z})$$

Hutchinson, Michael F. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines." *Communications in Statistics-Simulation and Computation* 18.3 (1989): 1059-1076.
Avron, Haim, and Sivan Toledo. "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix." Journal of the ACM (JACM) 58.2 (2011): 1-34.
Meyer, Raphael A., et al. "Hutch++: Optimal Stochastic Trace Estimation." Symposium on Simplicity in Algorithms (SOSA). Society for Industrial and Applied Mathematics, 2021.

SLIM ML4Seismic

Randomized linear algebra

Randomized Trace Estimation:

$$\mathrm{tr}(\mathbf{A}) \approx \frac{1}{r}\sum_{j=1}^{r}\mathbf{z}_j^\top \mathbf{A}\mathbf{z}_j = \frac{1}{r}\mathrm{tr}(\mathbf{Z}^\top \mathbf{A}\mathbf{Z})$$

- only needs matrix-free access to actions of $\mathbf{A}$
- unbiased estimator when $\mathbb{E}(\mathbf{z}\mathbf{z}^\top) = \mathbf{I}$ w/ accuracy $\propto r \ll N$, # of sketches w/ random vectors $\mathbf{z}_j$
- errors studied & understood

Why should we care?

Randomized trace estimation

Approximate FWI gradient calculation for $r \ll n_t$:

$$\delta\mathbf{m}[\mathbf{x}] = \mathrm{tr}\left(\ddot{\mathbf{u}}[t,\mathbf{x}]\mathbf{v}[t,\mathbf{x}]^\top\right) \approx \frac{1}{r}\,\mathrm{tr}\left((\mathbf{Z}^\top\ddot{\mathbf{u}}[\mathbf{x}])(\mathbf{v}[\mathbf{x}]^\top\mathbf{Z})\right)$$

- $\ddot{\mathbf{u}}$ second time derivative solution forward wave equation
- $\mathbf{v}$ solution adjoint wave equation
- $\sum \mathbf{x}_i\mathbf{y}_i = \mathbf{x}^\top\mathbf{y} = \mathrm{tr}(\mathbf{x}\mathbf{y}^\top)$
- probing vectors $\mathbf{Z} = \left[\mathbf{z}_1 \cdots \mathbf{z}_r\right]$ with $\mathbb{E}(\mathbf{z}_i^\top\mathbf{z}_i) = 1$

Choice of probing vectors

QR decomposition on the range of $\mathbf{u}$ is too expensive

Use the data as a proxy

$$\left[\mathbf{Q}, \sim\right] = \mathrm{qr}(\mathbf{AZ}) \quad \text{with} \quad \mathbf{A} = \mathbf{D}_{\text{obs}}\mathbf{D}_{\text{obs}}^\top$$

Data $\mathbf{D}_{\text{obs}}$ corresponds to

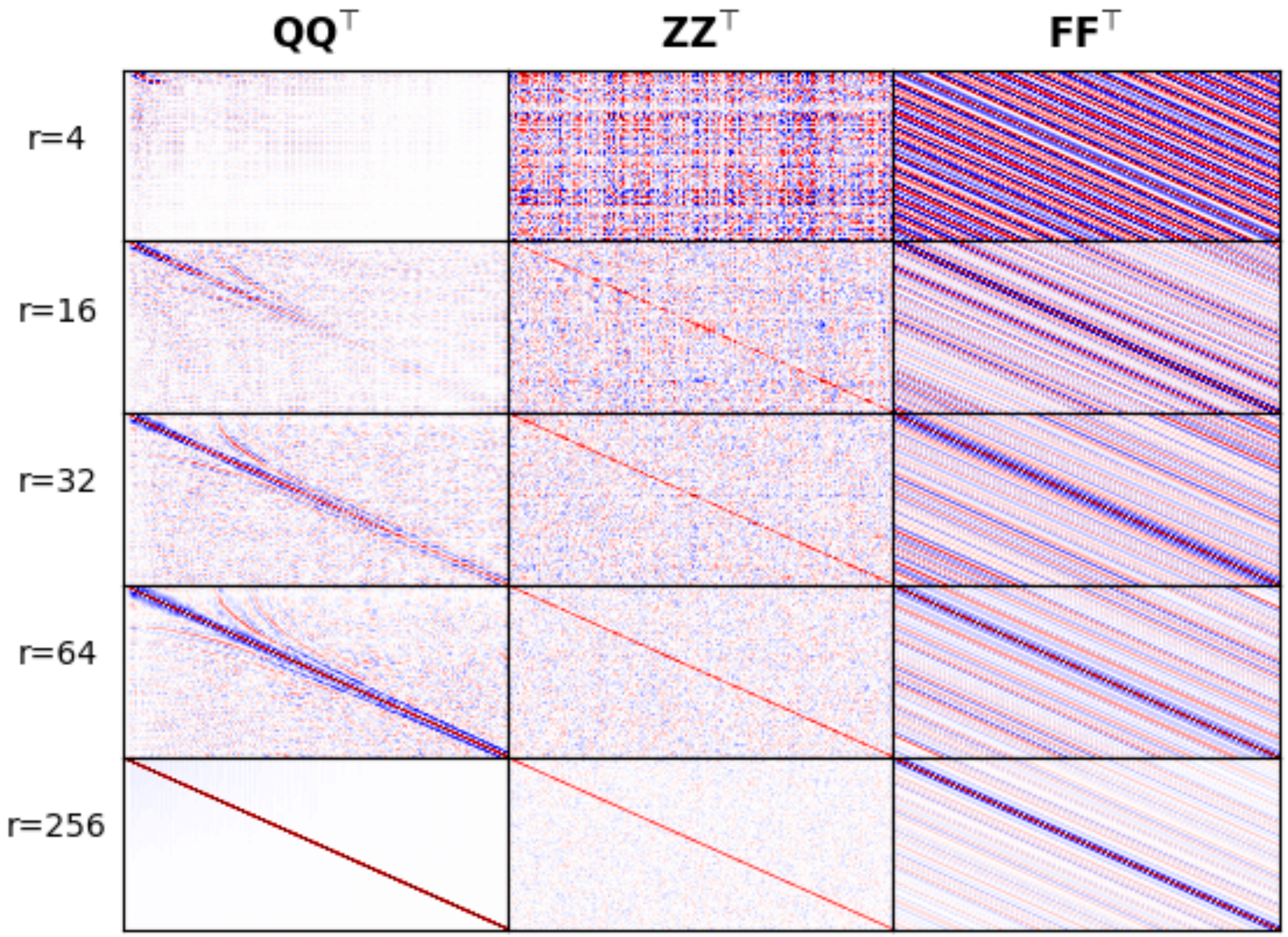
- restriction of the wavefield $\mathbf{u}$ to the receivers
- is representative of its range

Crosstalk

Stronger diagonal

Less crosstalk

Less coherent noise



Z : Random +-1
F: DFT

Approximate gradient FWI/RTM

Algorithm:

Accumulate over time

0. **for t=2:nt-1** # forward propagation
1. $\mathbf{u}[t+1] = f(\mathbf{u}[t], \mathbf{u}[t-1], \mathbf{m}, \mathbf{q}[t])$
2. $\overline{\ddot{\mathbf{u}}}[\mathbf{r}, \mathbf{x}] \mathrel{+}= \mathbf{Q}[\mathbf{r}, t]\ddot{\mathbf{u}}[t, \mathbf{x}] \quad \forall\ \mathbf{r}$
3. **end for**
4. **for t=nt:-1:1** # back propagation
5. $\mathbf{v}[t-1] = f^\top(\mathbf{v}[t], \mathbf{v}[t+1], \mathbf{m}, \delta\mathbf{d}[t])$
6. $\overline{\mathbf{v}}[\mathbf{r}, \mathbf{x}] \mathrel{+}= \mathbf{Q}[\mathbf{r}, t]\mathbf{v}[t, \mathbf{x}] \quad \forall\ \mathbf{r}$
7. **end for**
8. output: $\frac{1}{r}\,\mathrm{tr}(\overline{\ddot{\mathbf{u}}}\,\overline{\mathbf{v}}^\top)$

$$\ddot{\mathbf{u}}, \mathbf{v} \in \mathbb{R}^{n_t \times N} \implies \overline{\ddot{\mathbf{u}}}, \overline{\mathbf{v}} \in \mathbb{R}^{r \times N},\ r \ll n_t$$

Philipp A. Witte, Mathias Louboutin, Fabio Luporini, Gerard J. Gorman, and Felix J. Herrmann, “Compressive least-squares migration with on-the-fly Fourier transforms”, Geophysics, vol. 84, pp. R655-R672, 2019
McMechan, G. A., 1983, Migration by extrapolation of time-dependent boundary values: Geophysical Prospecting, 31, 413–420.

SLIM
ML4Seismic

Randomized trace estimation

Ultra-low memory use:

	FWI	DFT	Probing	Optimal checkpointing	Boundary reconstruction
Compute	0	$\mathcal{O}(2r) \times n_t \times N$	$\mathcal{O}(r) \times n_t \times N$	$\mathcal{O}(log(n_t)) \times N \times n_t$	$n_t \times N$
Memory	$N \times n_t$	$2r \times N$	$r \times N$	$\mathcal{O}(10) \times N$	$n_t \times N^{\frac{2}{3}}$

For fixed r:

- half memory cost of DFT
- half compute cost of DFT
- simple real-valued algorithm

Needs much smaller r than with DFT in practice

Op't Root, T. J., C. C. Stolk, and M. V. de Hoop, 2012, Linearized inverse scattering based on seismic reverse time migration: Journal de Mathematiques Pures et Appliquees, 98, 211–238.

Randomized trace estimation

Ultra-cheap imaging conditions:

$$\mathrm{Q}^\top \left(\mathbf{D}_x \mathbf{u}[\cdot,\mathbf{x}]\right) = \mathbf{D}_x \left(\mathbf{Q}^\top \mathbf{u}[\cdot,\mathbf{x}]\right)$$

Apply space-only imaging condition to time-compressed wavefields:

- K-space filter
- inverse-scattering imaging condition (ISIC)

Imaging condition usually costs extra(s) PDEs (ISIC = 1 PDE)

FWI example

Hands-on tutorial:

Breakout 2. Scalable Software in the Cloud

2D overthrust model

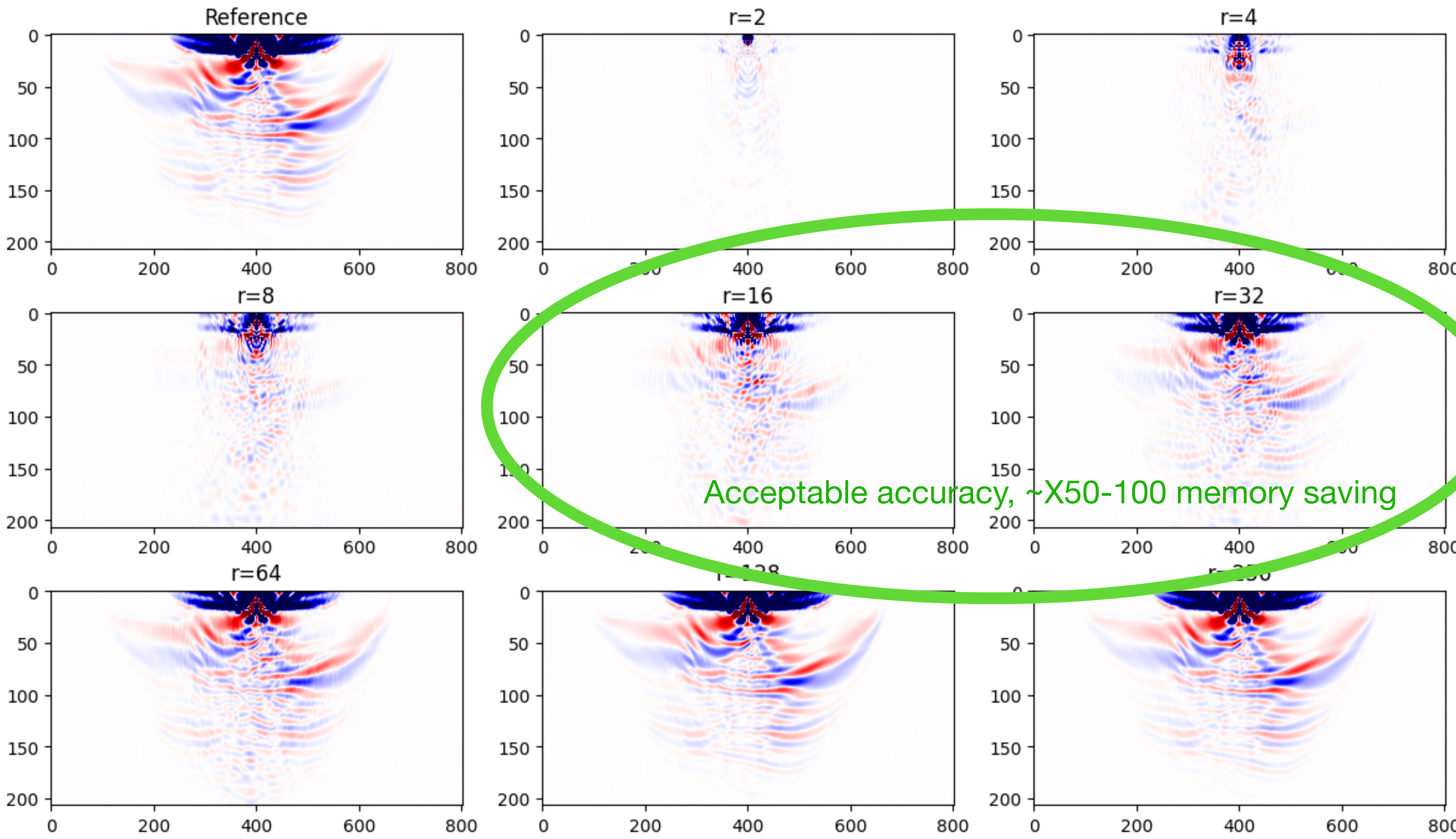
OBN acquisition

Comparisons:

- standard FWI
- on-the-fly DFT
- randomized trace estimation

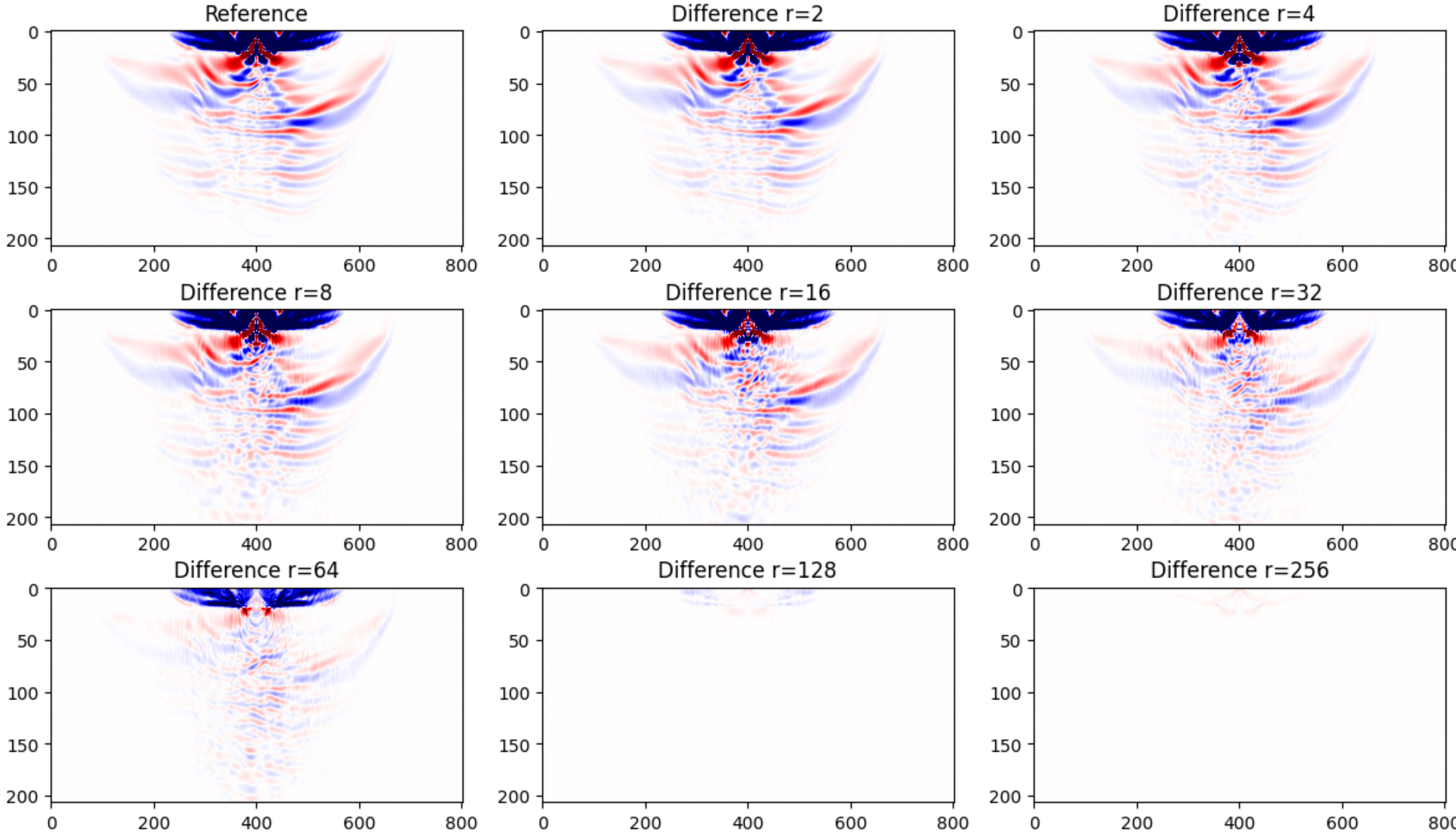
Accuracy – gradients

- Converges to true gradient as $r \rightarrow n_t$
- Less accurate near source



Accuracy – gradients

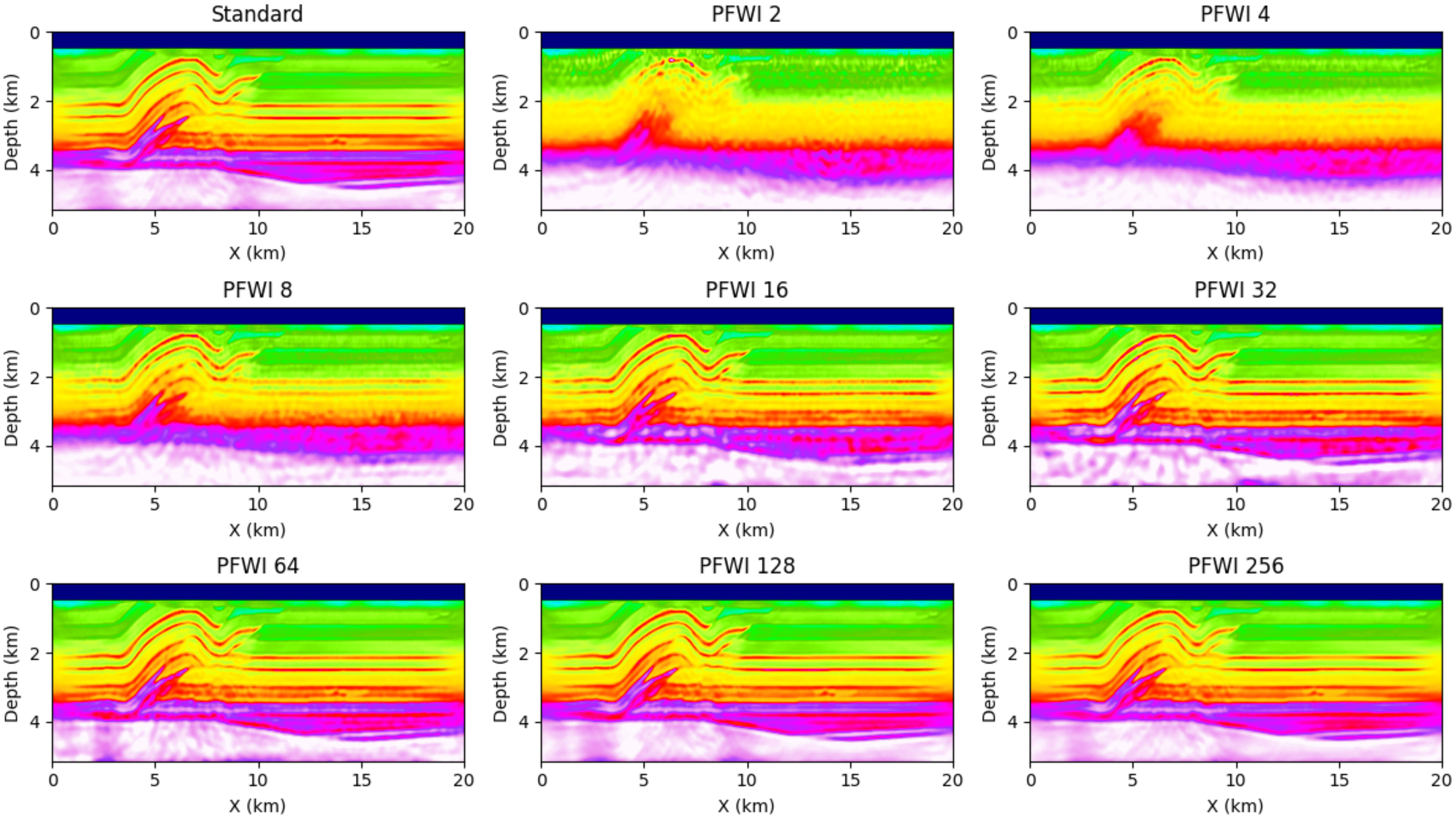
- Exact for high r
- Noisy error



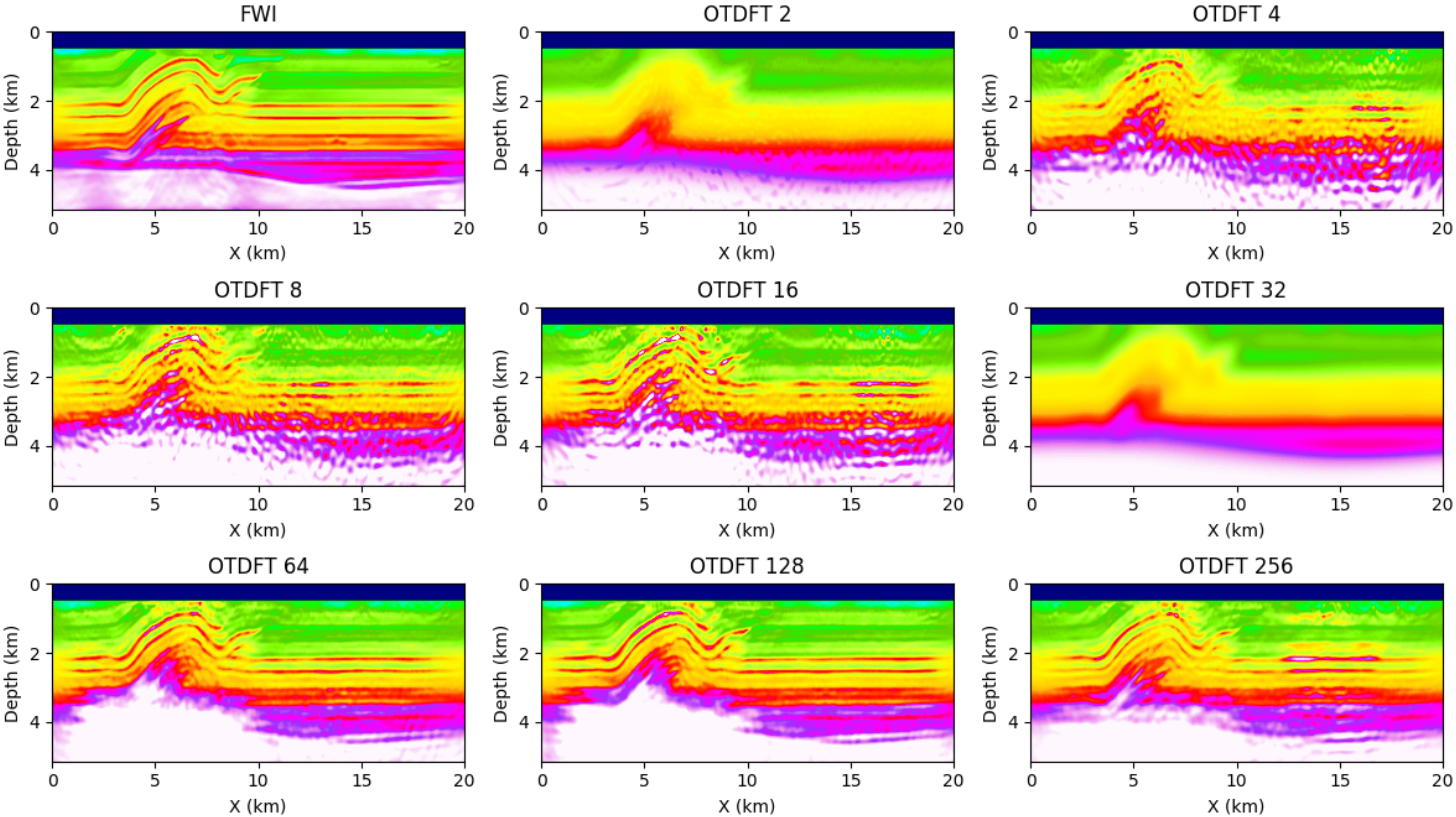
Very accurate but gets expensive

Trace estimation

SLIM
ML4Seismic



DFT



SLIM
ML4Seismic

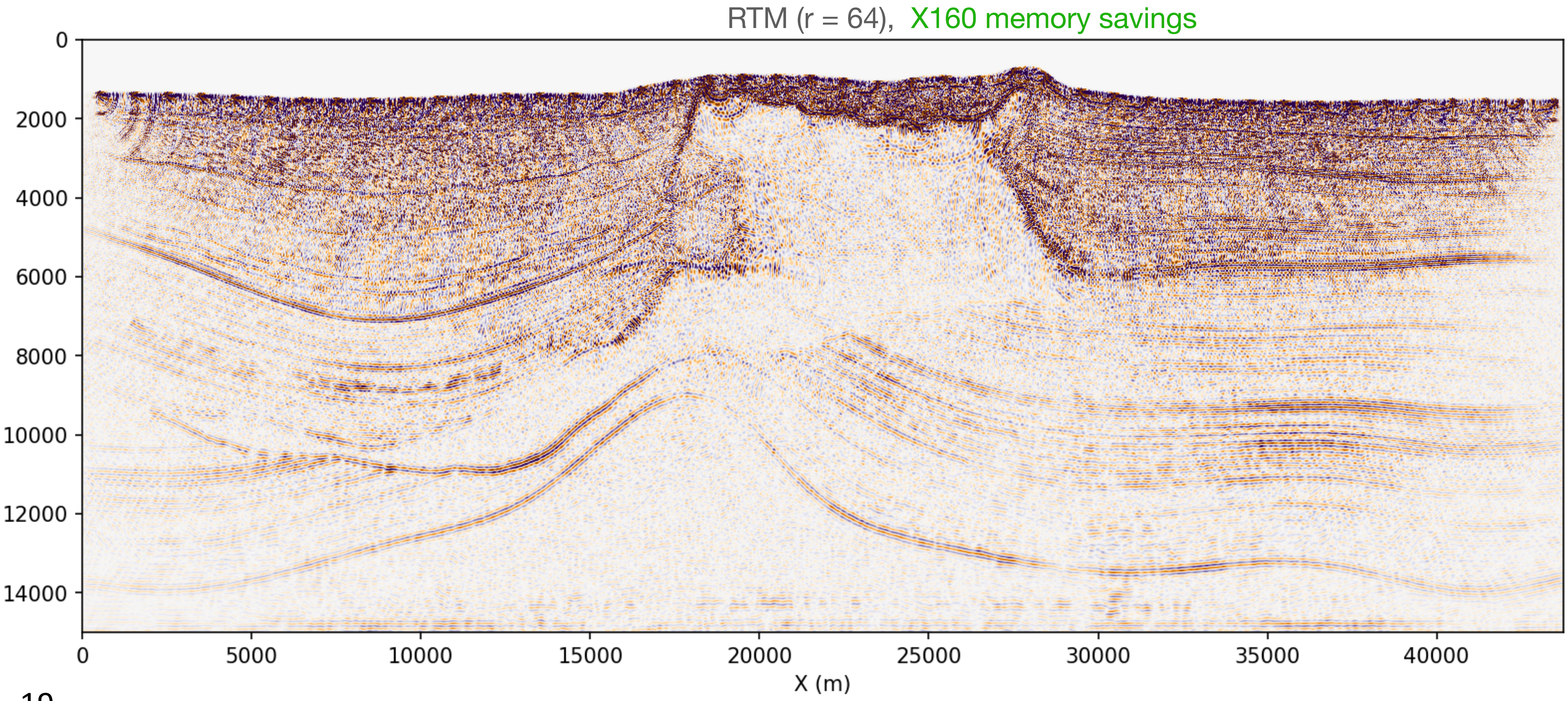
RTM

SEAM 2D:

- 44 OBN 1km apart
- 3521 sources 12.5m apart
- 14.5Hz Ricker wavelet
- 64 probing vectors (160 X memory savings, 84Gb vs .5Gb)

Makes RTM conducive to acceleration w/ GPUs

Noisy but accurate

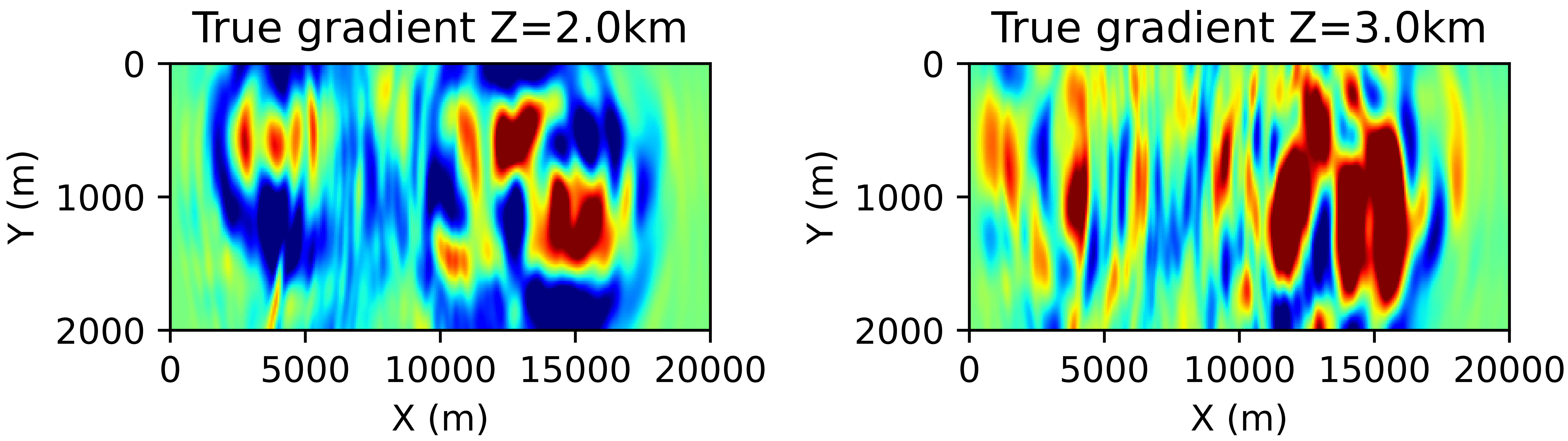


3D, frist gradient

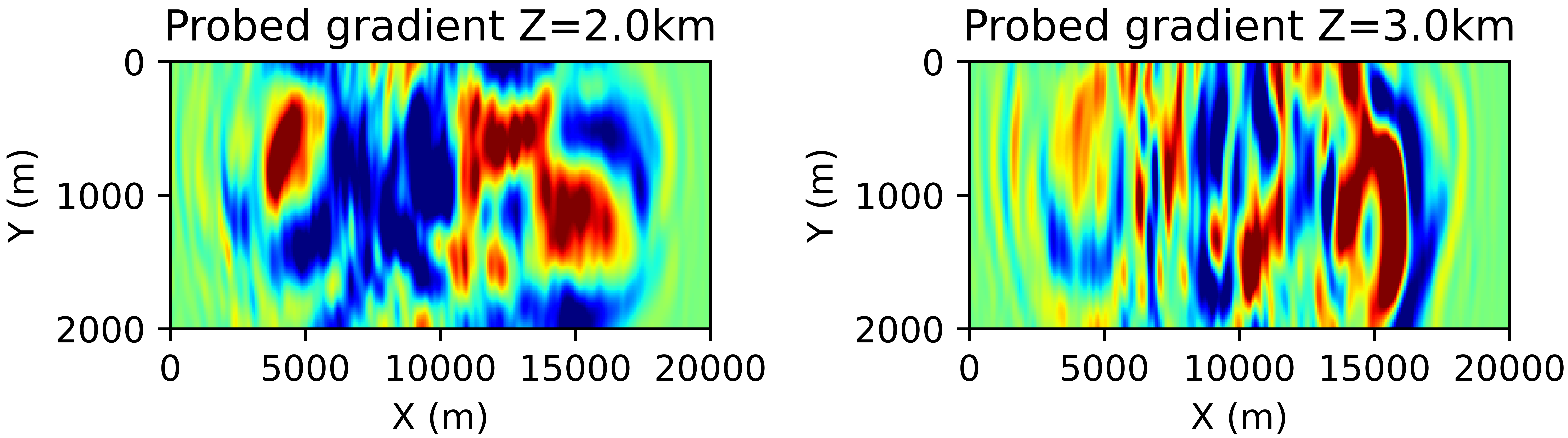
Overthrust 3D:

- Marine acqusition
- 12.5Hz Ricker wavelet filter at 3-15Hz
- 32 probing vectors => X40 memory reduction
- Probing on GPU (M60, 45$/hr)
- True gradient on CPU (Intel Skylake, 65$/hour)

- Truncated QR => Lower frequency
- Matches the true gradient well (only 32 vectors)



Some artifacts but accurate at depth



Conclusions

Leverage Randomized Linear Algebra

Low memory foot print and low algorithmic complexity

Controllable error

Allows for accelerators (GPUs)

Drop-in extension for existing open source framework JUDI/Devito

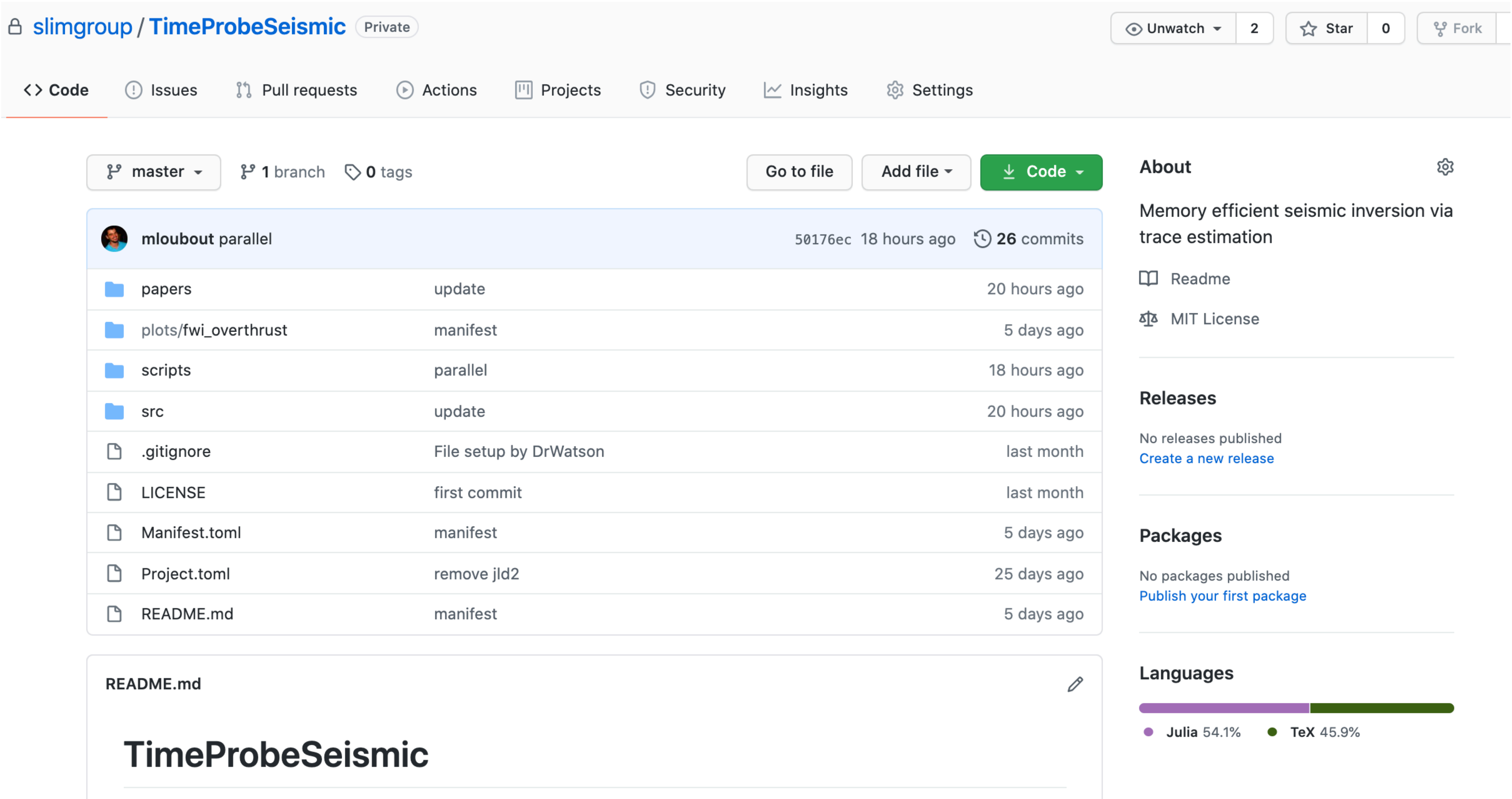
SLIM ML4Seismic

Open source software

```
# Standard JUDI
function objective_function(x)
  model0.m .= x
   f, g = fwi_objective(model0, q[idx], d_obs[idx]; options=opt)
end
options = spg_options(verbose = 3, maxIter = fevals, memory = 3,
iniStep = 1f0)
g_const = 0
sol = spg(x->objective_function(x), vec(m0), ProjBound, options)

# Probing extension
function objective_function(x, ps)
  model0.m .= x
   f, g = fwi_objective(model0, q[idx], d_obs[idx], ps; options=opt)
end
ps = 32
global g_const = 0
sol = spg(x->objective_function(x, ps), vec(m0), ProjBound, options)
```

TimeProbeSeismic.jl:

- Open-source MIT license
- Built on top of JUDI.jl
- Leverages Devito
- https://github.com/slimgroup/TimeProbeSeismic.jl

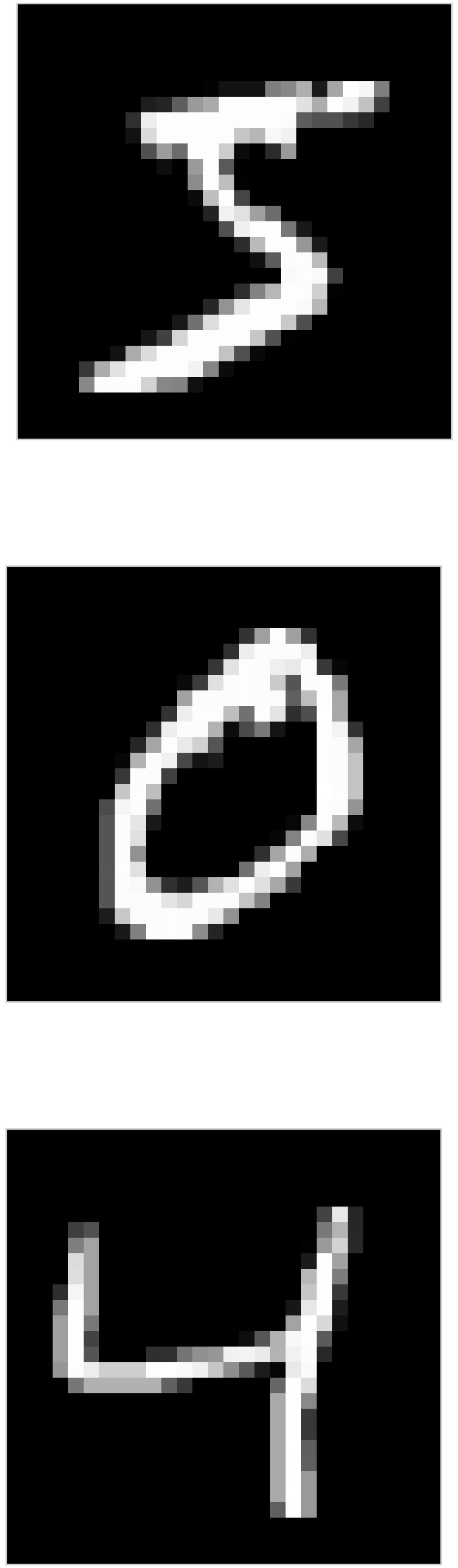


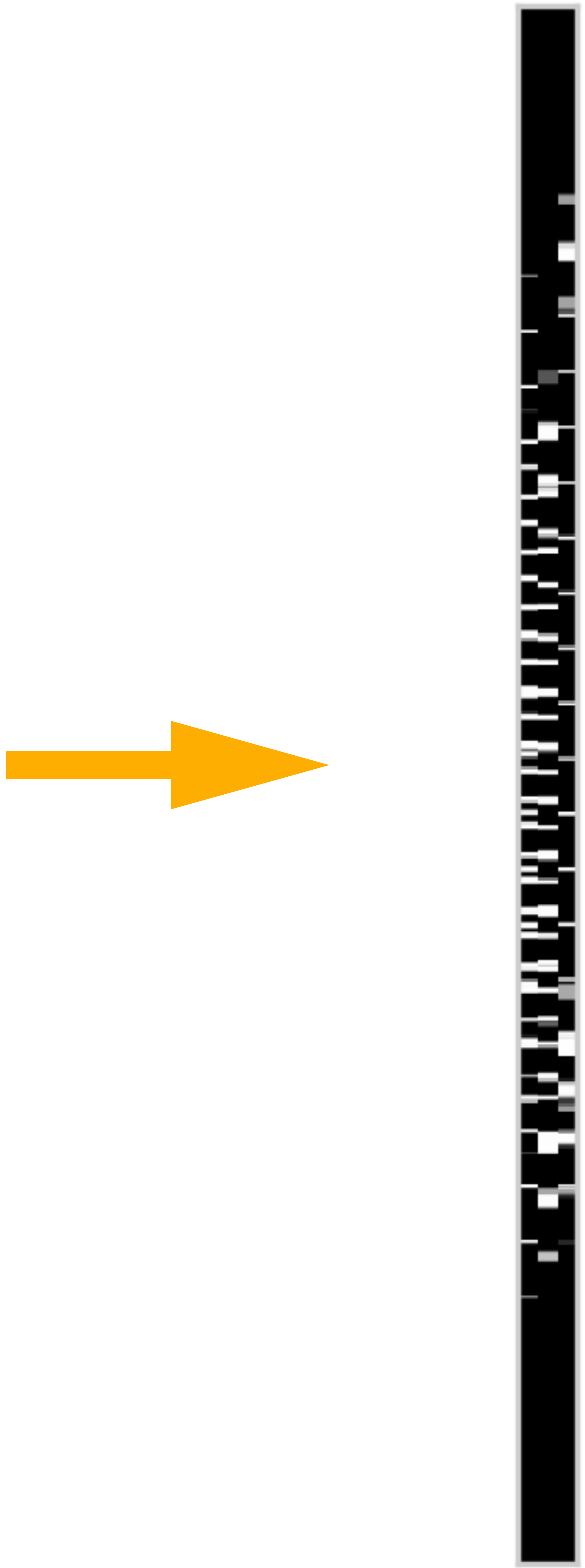
Convolutions in ML

- Equivalent to adjoint state
  - Network is the wave-equation
  - Backpropagation is the adjoint wave-equation
- Similar gradient structure
  - Correlation of input and back propagated residual
- Bottleneck of CNNs
  - High memory (i.e saving wavefield)
  - High computed cost (i.e laplacian)

Vectorize

$\mathbb{R}^{n_x \times n_y}$





$\mathbf{X} \in \mathbb{R}^{N \times b}$

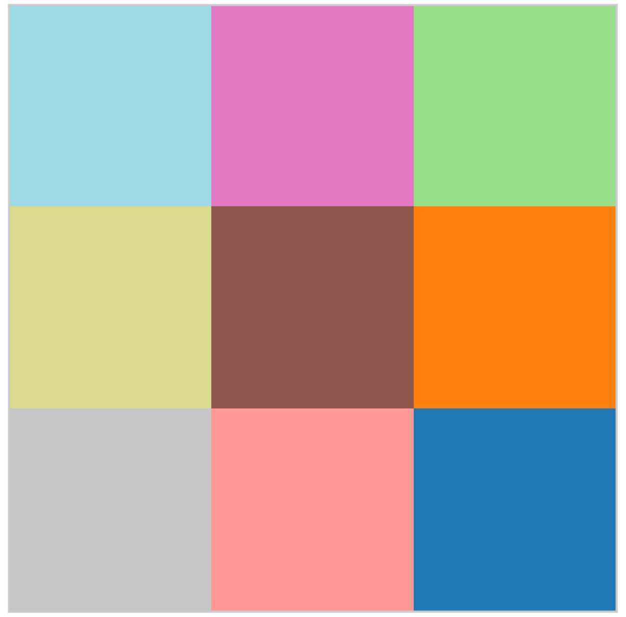
b batch size

N image size

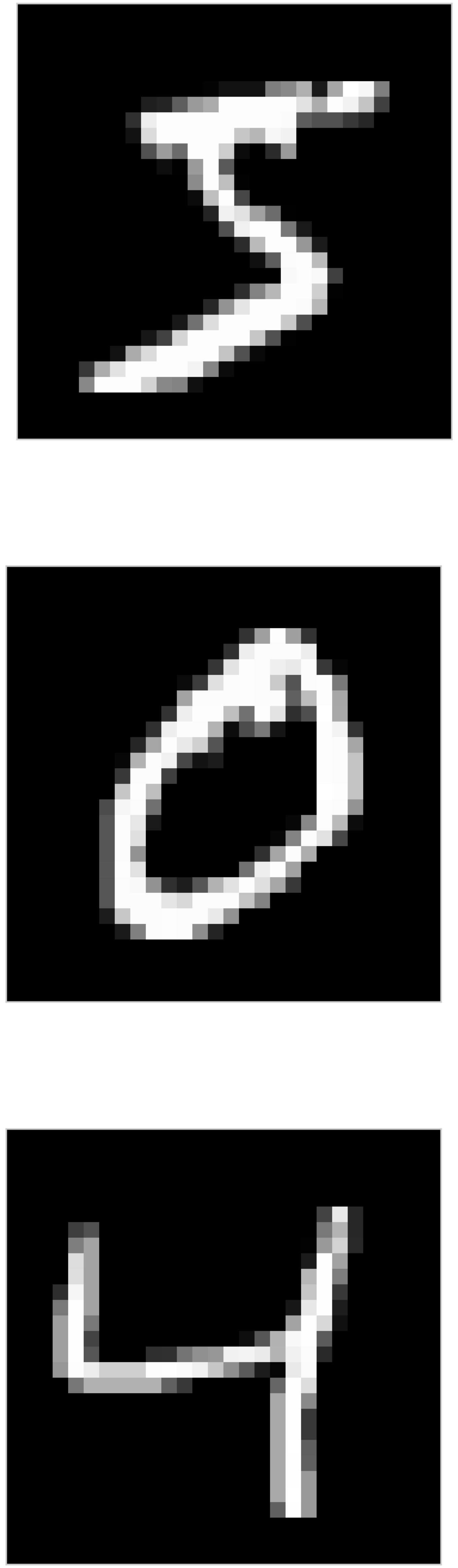
$N = n_x \times n_y$

CNN

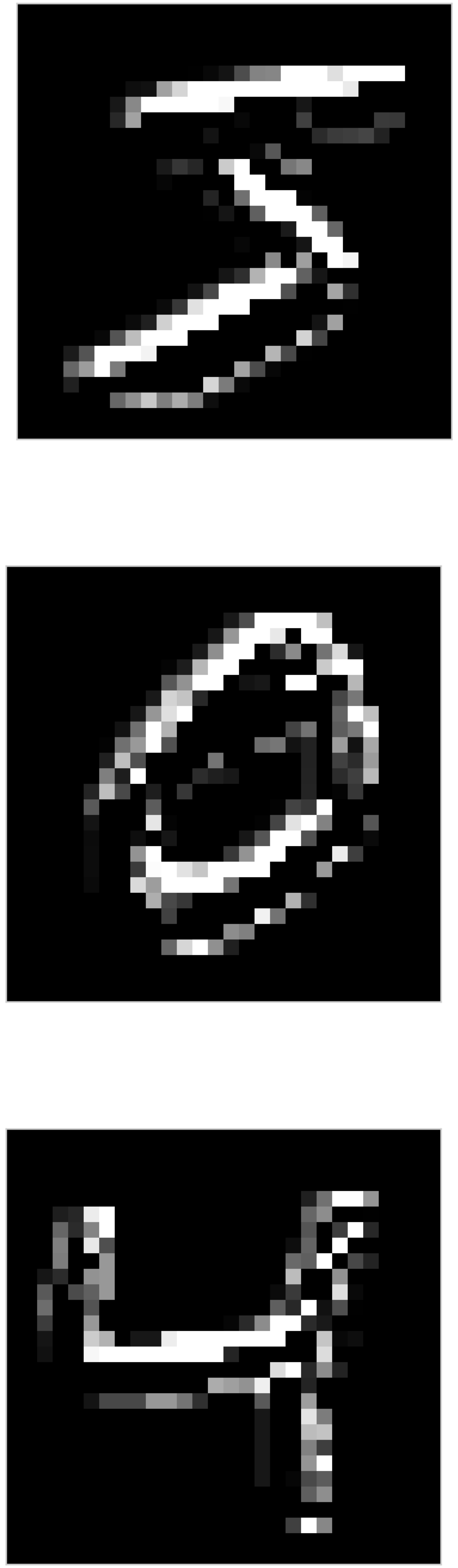
$\mathbf{w} \in \mathbb{R}^{n_w},\ n_w = 9$



$\star$

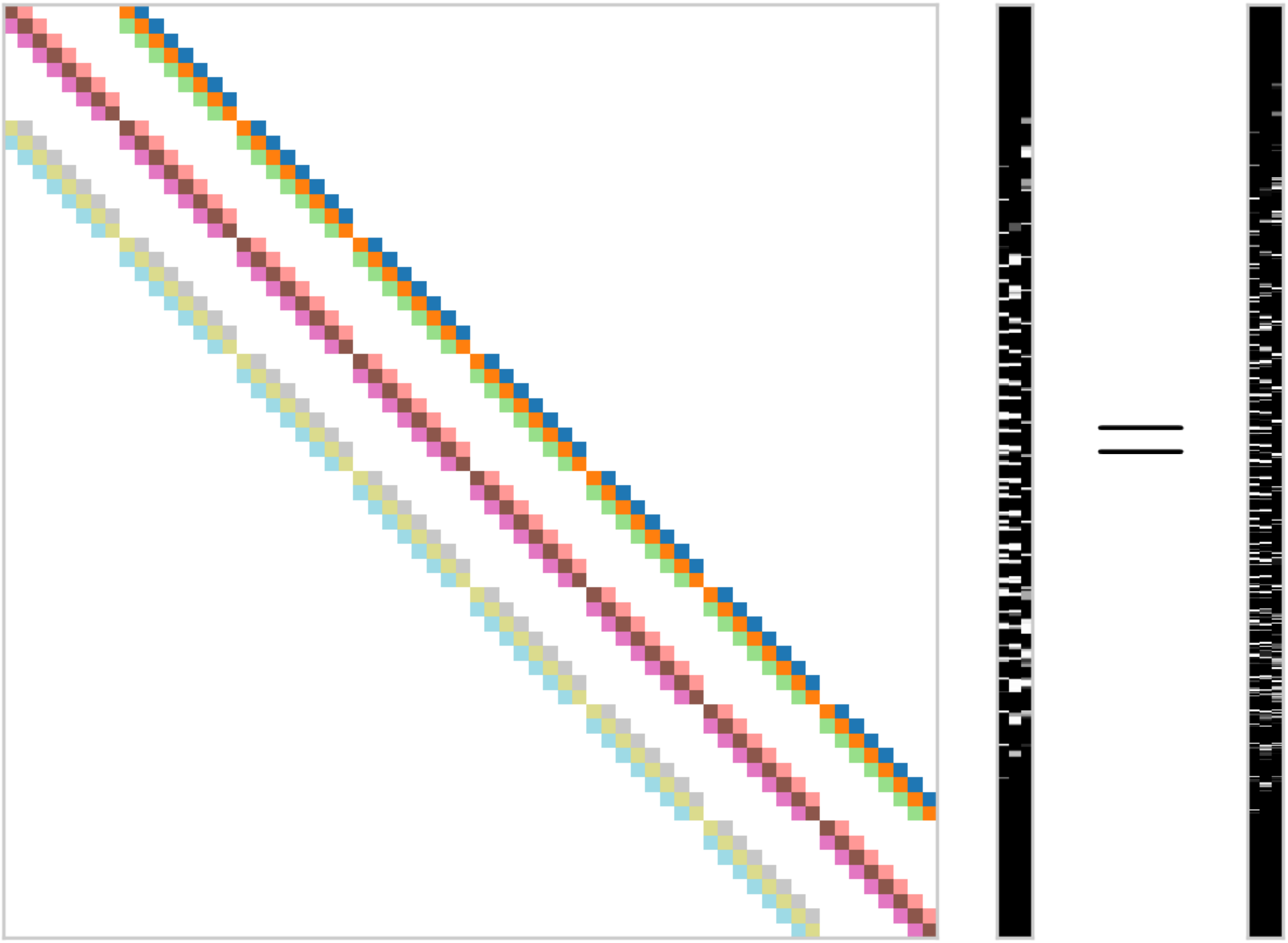


$=$

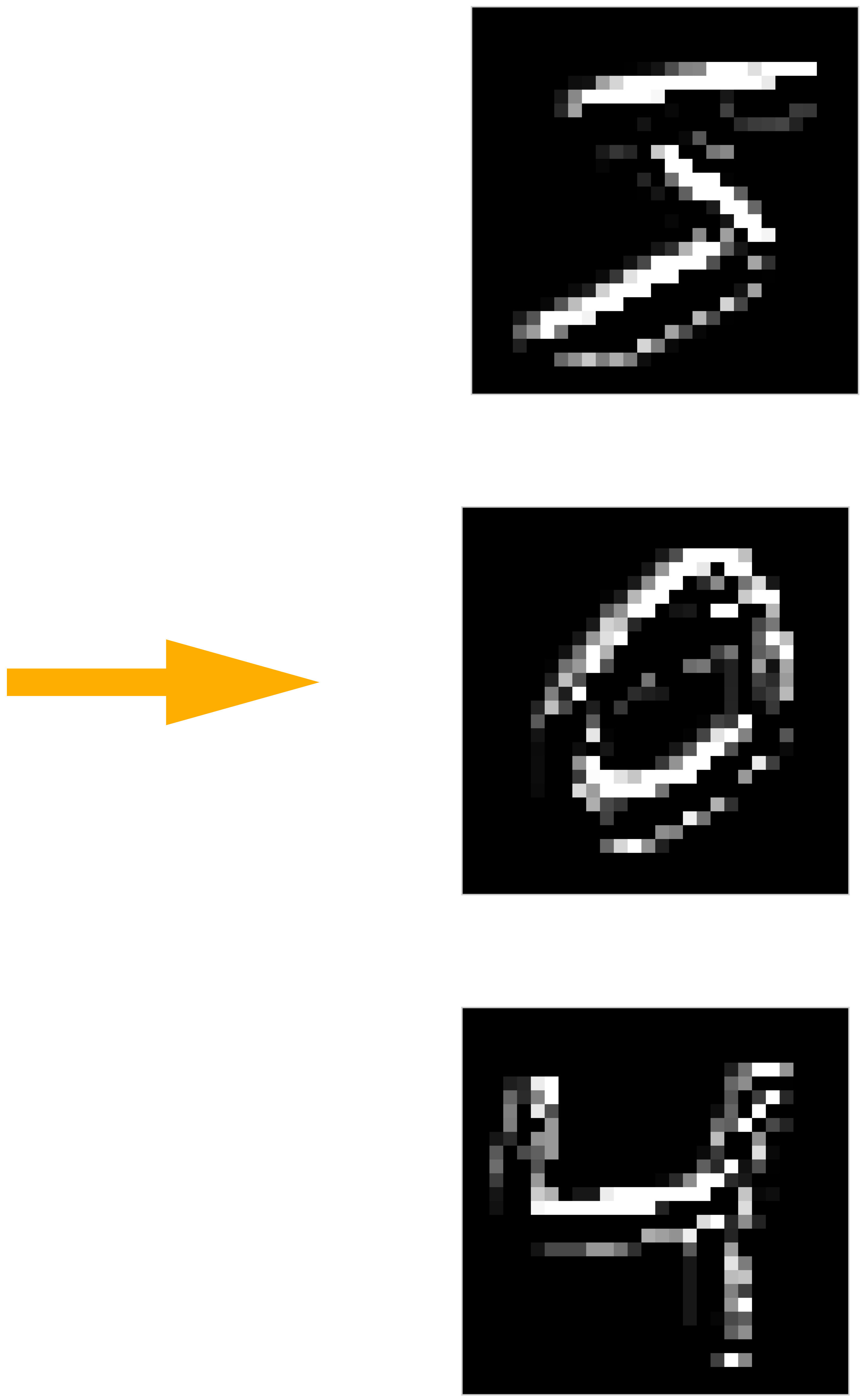


$\mathrm{conv}(\mathbf{x}; \mathbf{w})$

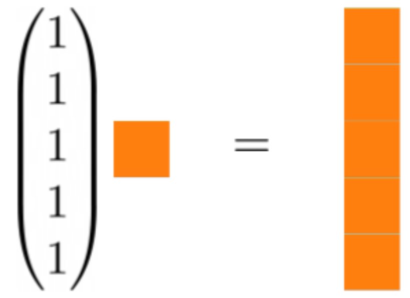
CNN as matvec



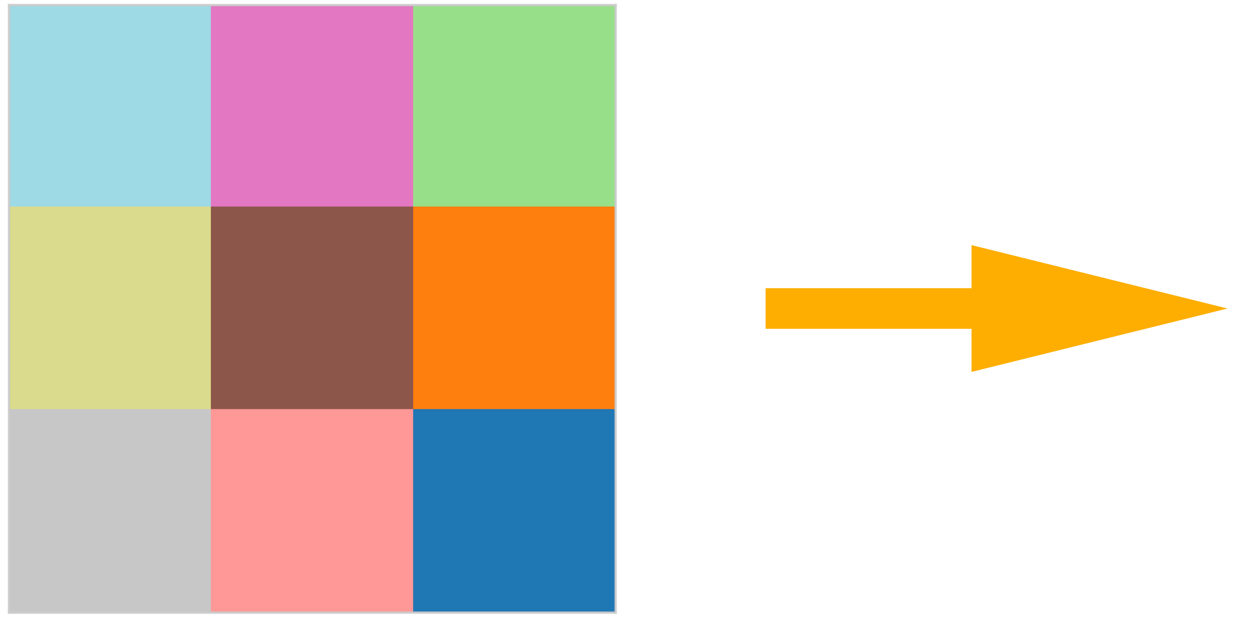
$\mathbf{Wx}$

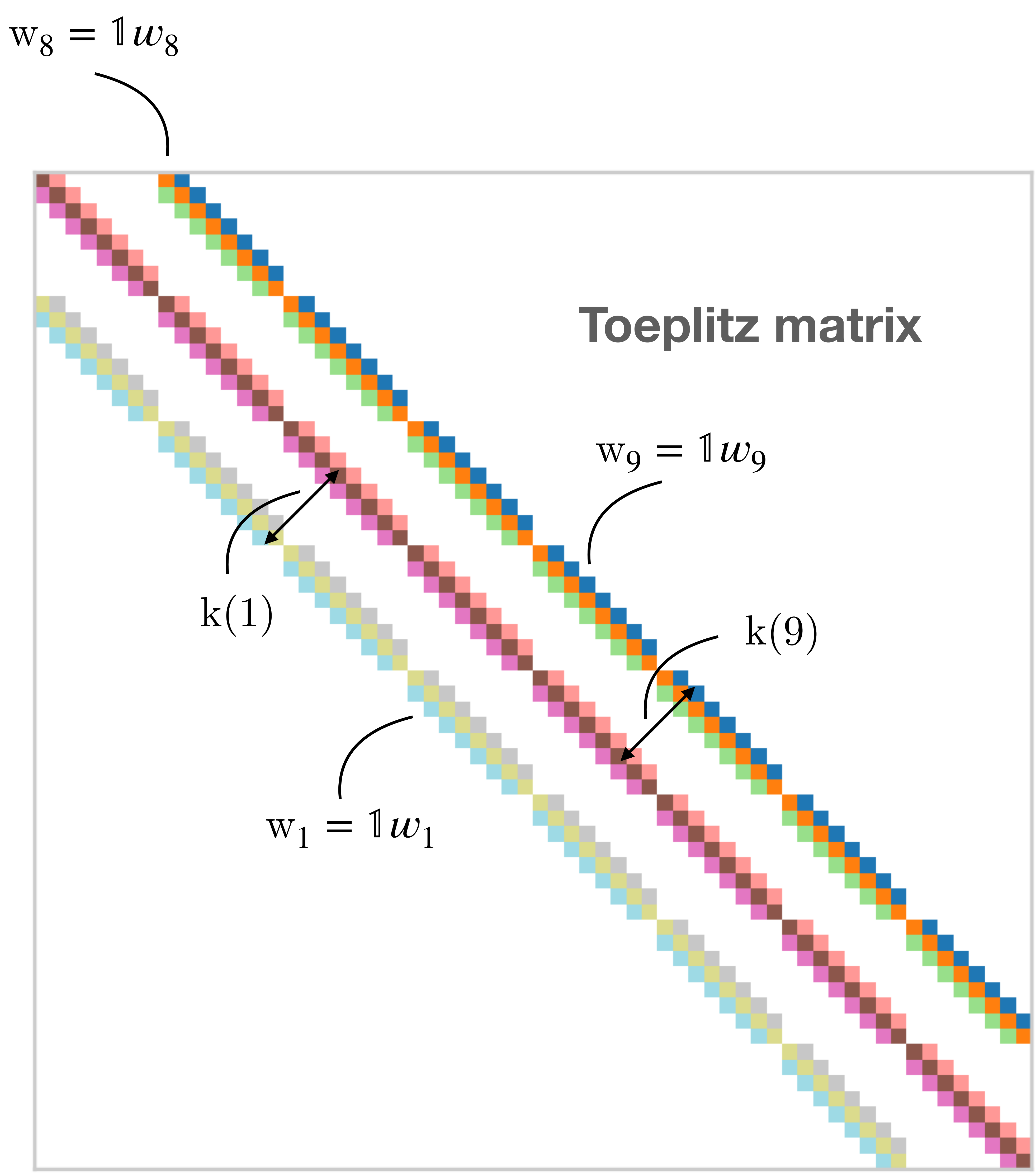


Matricize



$\mathbf{w} \in \mathbb{R}^{n_w}, \; n_w = 9$





$$\mathbf{W} = \left(\sum_{i=1}^{n_w} \overrightarrow{\mathrm{diag}}(\mathbf{w}_i) \mathbf{T}_{k(i)} \right)$$

Trace in CNNs?

Gradient w.r.t all weights

$$\frac{\partial}{\partial \mathbf{W}} f(\mathbf{W}\mathbf{X}) = \delta \mathbf{Y}\mathbf{X}^\top$$

"Toeplitz" structure

$$\mathbf{W} = \sum_{i=1}^{n_w} \overrightarrow{\mathrm{diag}}(\mathbf{w}_i) T_{k(i)} = \sum_{i=1}^{n_w} \overrightarrow{\mathrm{diag}}(w_i \mathbf{1}) T_{k(i)}$$

Gradient w.r.t. i^{th} weight conv layer

$$\begin{aligned}\frac{\partial}{\partial w_i} f(\mathbf{W}\mathbf{X}) &= \mathrm{tr}\left(\left(\frac{\partial f(\mathbf{W}\mathbf{X})}{\partial \mathbf{W}}\right)^\top \frac{\partial \mathbf{W}}{\partial w_i}\right) \\ &= \mathrm{tr}\left(\left(\delta \mathbf{Y}\mathbf{X}^\top\right)^\top \mathbf{T}_{k(i)}^\top\right) \\ &= \mathrm{tr}\left(\mathbf{X}\delta\mathbf{Y}^\top \mathbf{T}_{-k(i)}\right)\end{aligned}$$

Hutchinson, Michael F. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines." *Communications in Statistics-Simulation and Computation* 18.3 (1989): 1059-1076.
Avron, Haim, and Sivan Toledo. "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix." Journal of the ACM (JACM) 58.2 (2011): 1-34.
Meyer, Raphael A., et al. "Hutch++: Optimal Stochastic Trace Estimation." Symposium on Simplicity in Algorithms (SOSA). Society for Industrial and Applied Mathematics, 2021.

Randomized trace estimation

Based on stochastic approximation of the identity $\mathbf{I}$

$$
\begin{aligned}
\operatorname{tr}(\mathbf{A}) = \operatorname{tr}(\mathbf{A}\mathbf{I}) &= \operatorname{tr}\left(\mathbf{A}\mathbb{E}\left[\mathbf{z}\mathbf{z}^\top\right]\right) \\
&= \mathbb{E}\left[\operatorname{tr}\left(\mathbf{A}\mathbf{z}\mathbf{z}^\top\right)\right] \\
&= \mathbb{E}\left[\mathbf{z}^\top\mathbf{A}\mathbf{z}\right] \\
&\approx \frac{1}{r}\sum_{i=1}^{r}\left[\mathbf{z}_i^\top\mathbf{A}\mathbf{z}_i\right] \\
&= \frac{1}{r}\operatorname{tr}\left(\mathbf{Z}^\top\mathbf{A}\mathbf{Z}\right)
\end{aligned}
$$

http://blog.shakirm.com/2015/09/machine-learning-trick-of-the-day-3-hutchinsons-trick/

Randomized trace estimation

Stochastic approximation of the shift operator $\mathbf{T}_{k(i)}$

$$
\begin{aligned}
\mathrm{tr}(\mathbf{A}\mathbf{T}_{k(i)}) &= \mathrm{tr}\left(\mathbf{A}\mathbb{E}\left[\mathbf{T}_{-k(i)}\mathbf{z}\mathbf{z}^\top\right]\right) \\
&\approx \frac{1}{r}\sum_{i=1}^{r}\left[\mathbf{z}_i^\top \mathbf{A}\mathbf{T}_{-k(i)}\mathbf{z}_i\right]
\end{aligned}
$$

So that

$$
\begin{aligned}
\delta w_i &= \frac{1}{r}\sum_{j=1}^{r}\left(\mathbf{z}_j^\top \mathbf{X}\right)\left(\delta \mathbf{Y}^\top \mathbf{T}_{-k(i)}\mathbf{z}_j\right) \\
&= \frac{1}{r}\mathrm{tr}(\underbrace{(\mathbf{Z}^\top \mathbf{X})}_{\overline{\mathbf{X}}\in\mathbb{R}^{r\times b}}\underbrace{(\delta\mathbf{Y}^\top \mathbf{T}_{-k(i)}\mathbf{Z})}_{\overline{\mathbf{Y}}^\top\in\mathbb{R}^{b\times r}})
\end{aligned}
$$

Algorithm

Forward pass

Data: Convolution input $\mathbf{X}$ and weights $\mathbf{w}$
Result: Convolution
begin
  Draw random seed s
  $\overline{\mathbf{X}} = \mathbf{Z}(s)^\top \mathbf{X}$
  $\mathbf{Y} = \mathrm{conv}(\mathbf{X}; \mathbf{w})$
  Store $\overline{\mathbf{X}}, s$
end

Backward pass

Data: Back-propagated residual $\delta\mathbf{Y}$
Result: Gradient w.r.t to weights
begin
  Load random seed s and probed forward $\overline{\mathbf{X}}$
  $\overline{\mathbf{Y}}^\top = \delta\mathbf{Y}^\top \mathbf{T}_{-k(i)} \mathbf{Z}(s)$
  $\delta\mathbf{w} = \mathrm{tr}(\overline{\mathbf{X}}\,\overline{\mathbf{Y}}^\top)$
end

$$\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N \times b} \Longrightarrow \overline{\mathbf{X}}, \overline{\mathbf{Y}} \in \mathbb{R}^{r \times b},\ r \ll N = n_x \times n_y$$

https://github.com/slimgroup/XConv/blob/master/benchmark/mem_prof.py

SLIM ML4Seismic

Convolutions only memory

```
scripts @ eas-coda-fherr07 [mlouboutin3](master)$ PYTORCH_NO_CUDA_MEMORY_CACHING=1 python3 mem_prof.py 1 &&
PYTORCH_NO_CUDA_MEMORY_CACHING=1 python3 mem_prof.py 2;
1 True
Network Sequential(
  (0): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (3): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
Input size torch.Size([128, 16, 256, 256])
GPU usage probe before forward: mem: 26.951%, abs-mem: 1.0595703125 (GiB) # nothing done
GPU usage probe after forward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y = N(X).mean()
GPU usage probe after backward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y.backward
Network Sequential(
  (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
Input size torch.Size([128, 16, 256, 256])
GPU usage true before forward: mem: 26.951%, abs-mem: 1.0595703125 (GiB) # nothing done
GPU usage true after forward: mem: 65.154%, abs-mem: 2.5615234375 (GiB) # after Y = N(X).mean()
GPU usage true after backward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y.backward
```

2Mb memory

1.5Gb memory

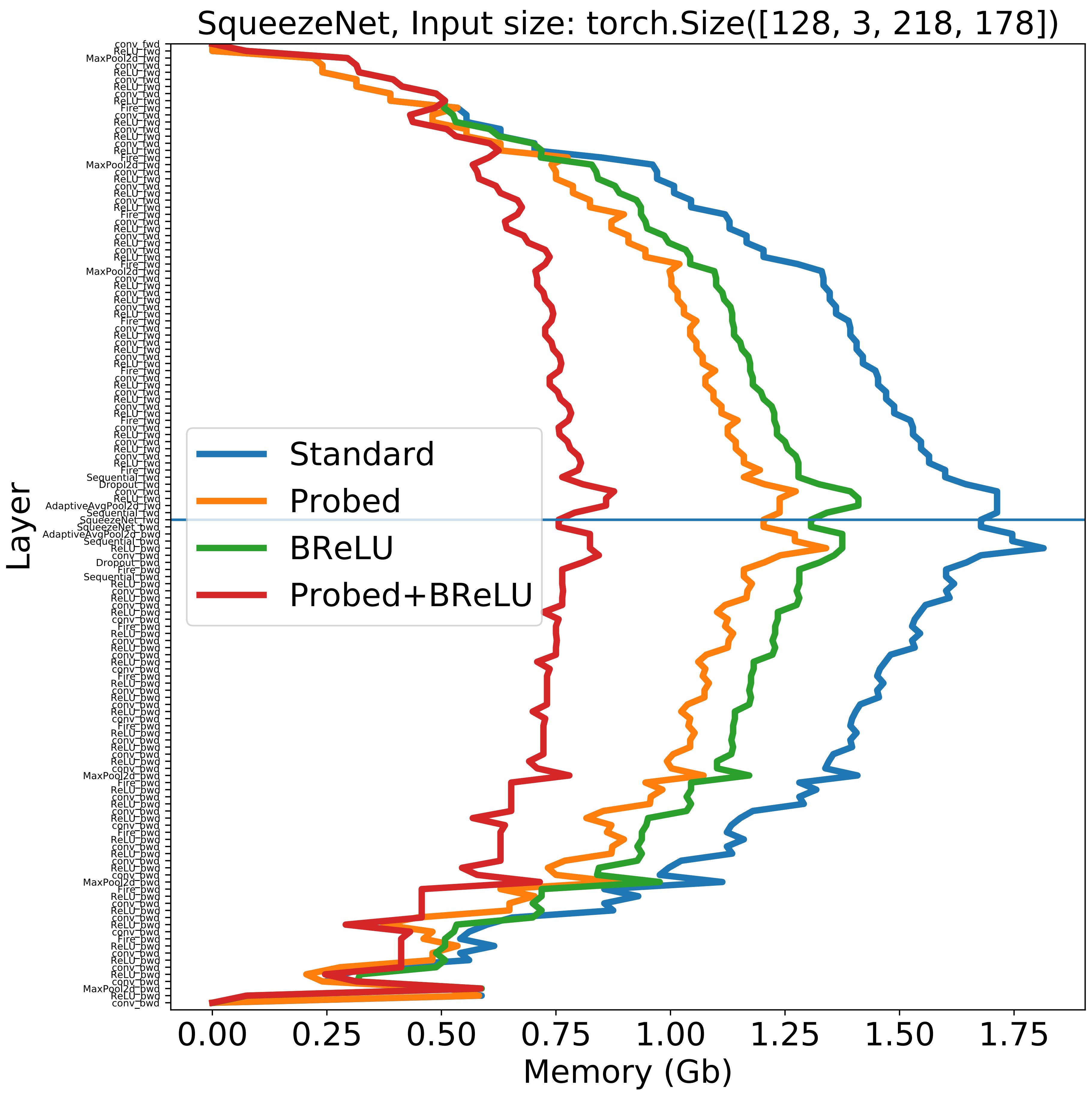
Orders of magnitude ($\mathcal{O}(10^3)$) reduction in memory usage…

Deep Learning Memory Usage and Pytorch Optimization Tricks, Quentin Fevbre. https://github.com/quentinf00/article-memory-log

https://github.com/slimgroup/XConv/blob/master/benchmark/network_mem.py

Network memory – squeezenet

- effective 100
- memory use
- opportunitie



Forward

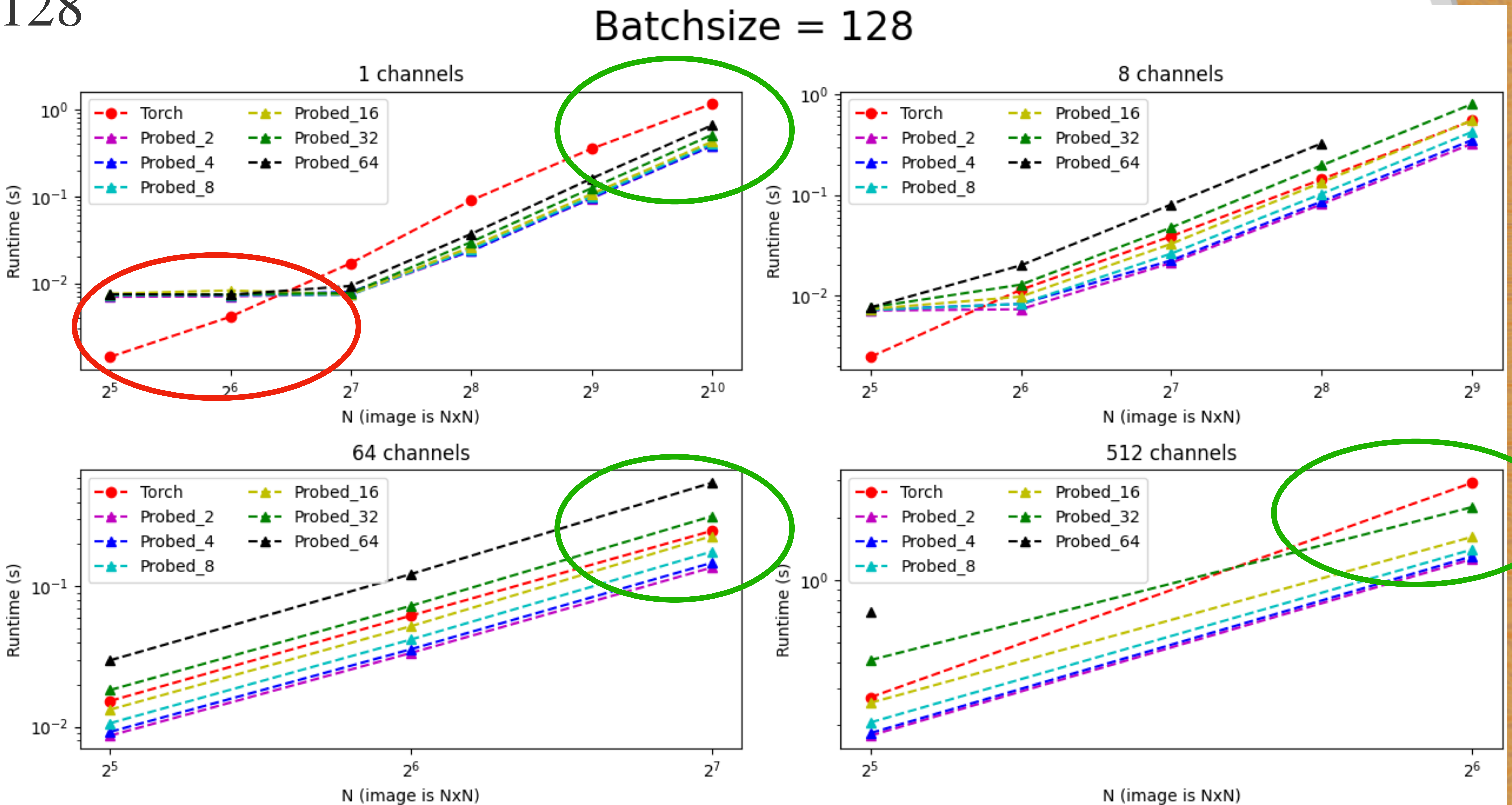
Backward

https://github.com/slimgroup/XConv/blob/master/benchmark/perf_pyxconv.py

SLIM ML4Seismic

GPU

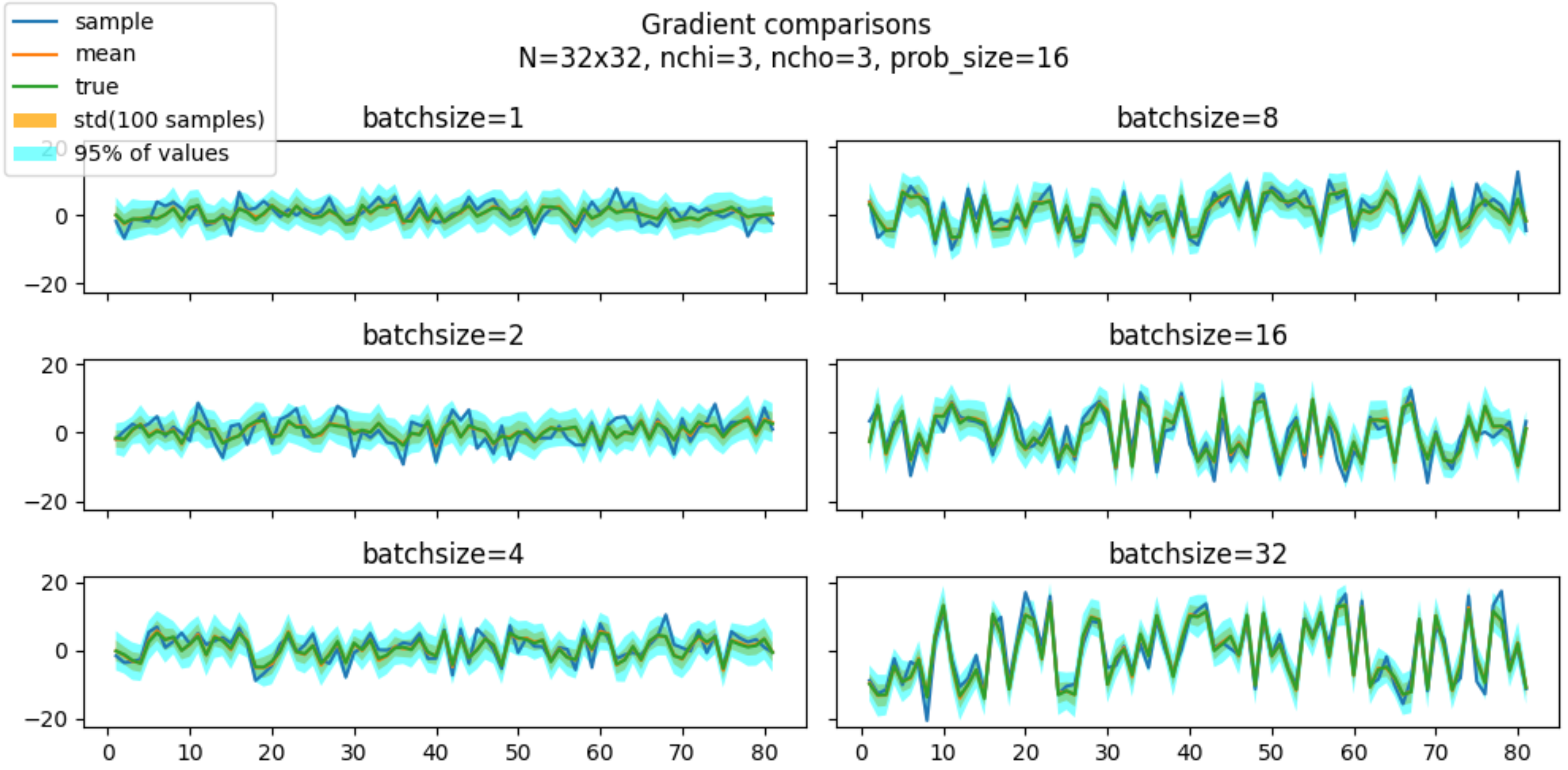
- sizes
  $N \times N \times (1 - 512) \times 128$
  for $N = 2^5 - 2^{10}$
- linear scaling in PyTorch
- up to $3\times$ speedup on GPUs
- not good for small sizes & small # of channels

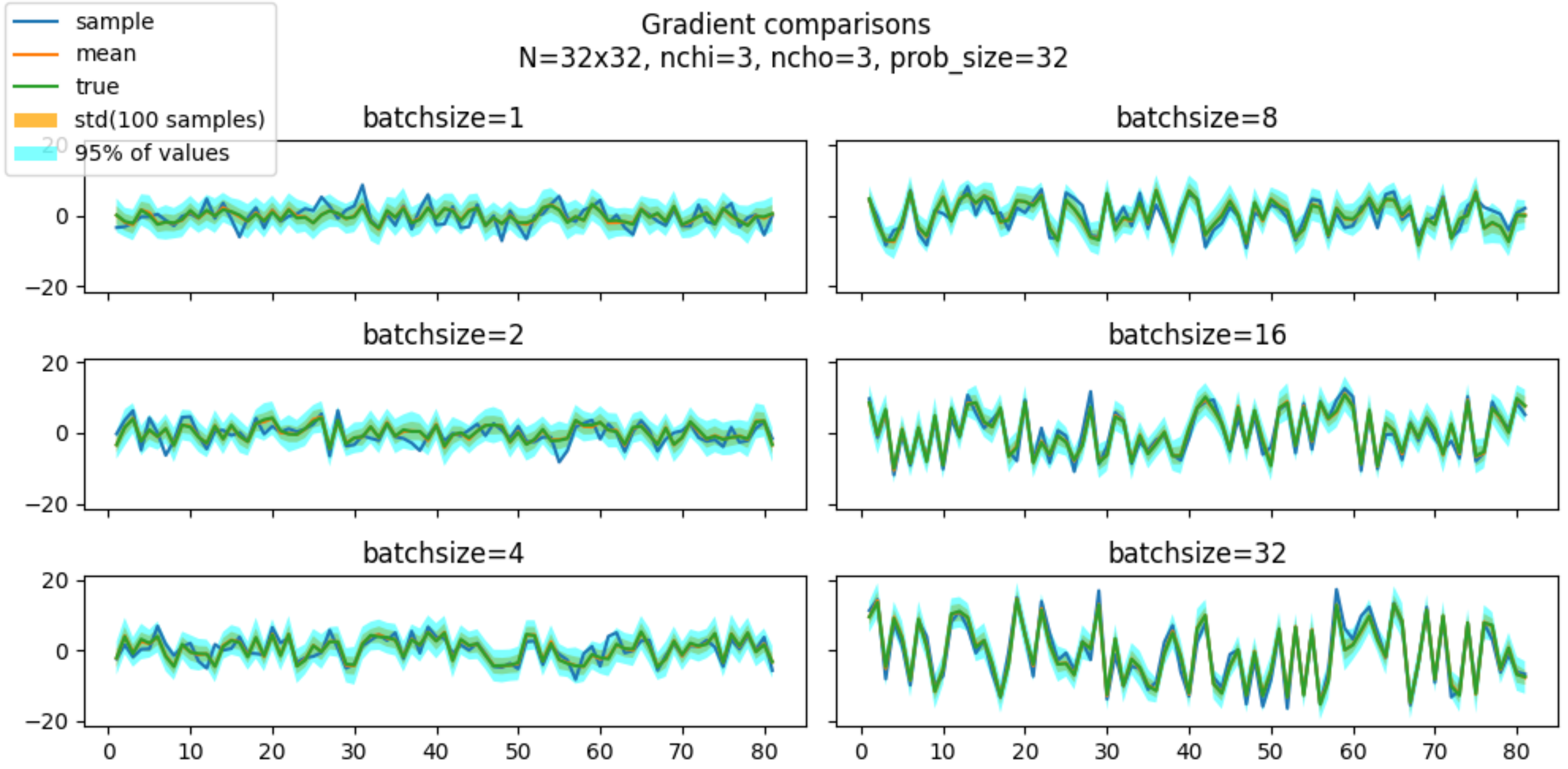


https://github.com/slimgroup/XConv/blob/master/benchmark/var_grad.jl

SLIM ML4Seismic

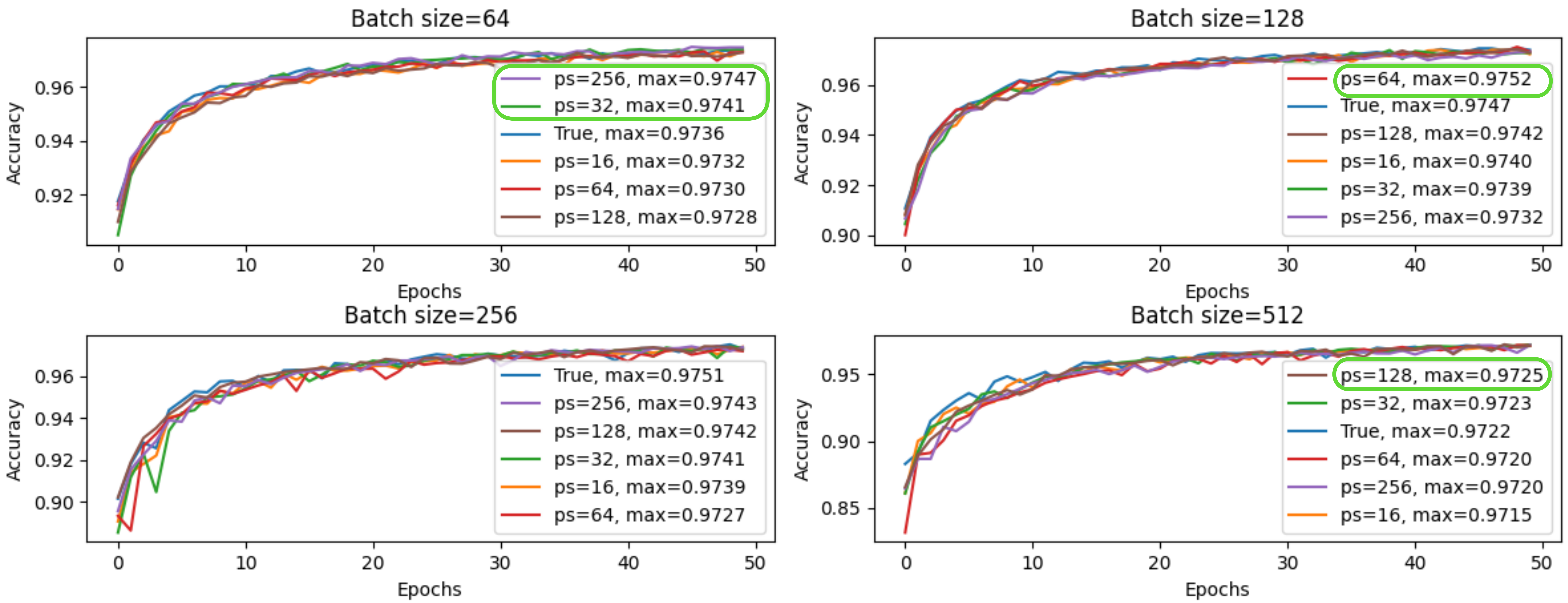
Accuracy





- theoretical control on the error of random-trace estimation
- for Gaussian probing vectors & $\mathbb{E}(\mathbf{z}\mathbf{z}^\top) = \mathbf{I}$ error decreases w/
  - batch size b
  - number of probing vectors r
- flexibility to strike balance between memory gain, compute & error

Accuracy MNIST training



- Validation classification accuracy vs epochs
- **No loss in classification accuracy**
- **Ability to work w/ larger batch sizes**

Observations

Gradient calculations w/ random trace estimation

- approximate gradient w/ controllable error
- theoretical memory reduction ($\mathcal{O}(n_x \times n_y \times n_c \times b)$ to $\mathcal{O}(r \times b)$) training CNNs
- effective memory improvement of $2\times$ for actual DNN
- computational performance improvement for larger images/channels
- speedups of $2-3\times$ for GPUs and $10\times$ CPUs
- comparable NN performance after training
- option to increase batch sizes (offset inaccuracies gradient)

We leveraged ideas known from randomized linear algebra.

Bottom line

NN training w/ randomized trace estimation

- more efficient use of hardware & less CO2 production
- facilitates ML@scale (e.g. video encoding, 3D seismic segmentation, etc.)
- allows for training next-generation memory-efficient & larger NNs

Use of randomized algorithms

- adaptive accuracy control during training
- use of optical devices (LightOn) for random projections

Technology is as good as the weakest link…

- reliance on dense linear algebra impedes ML@scale
- optimization landscape remains a challenge

Code

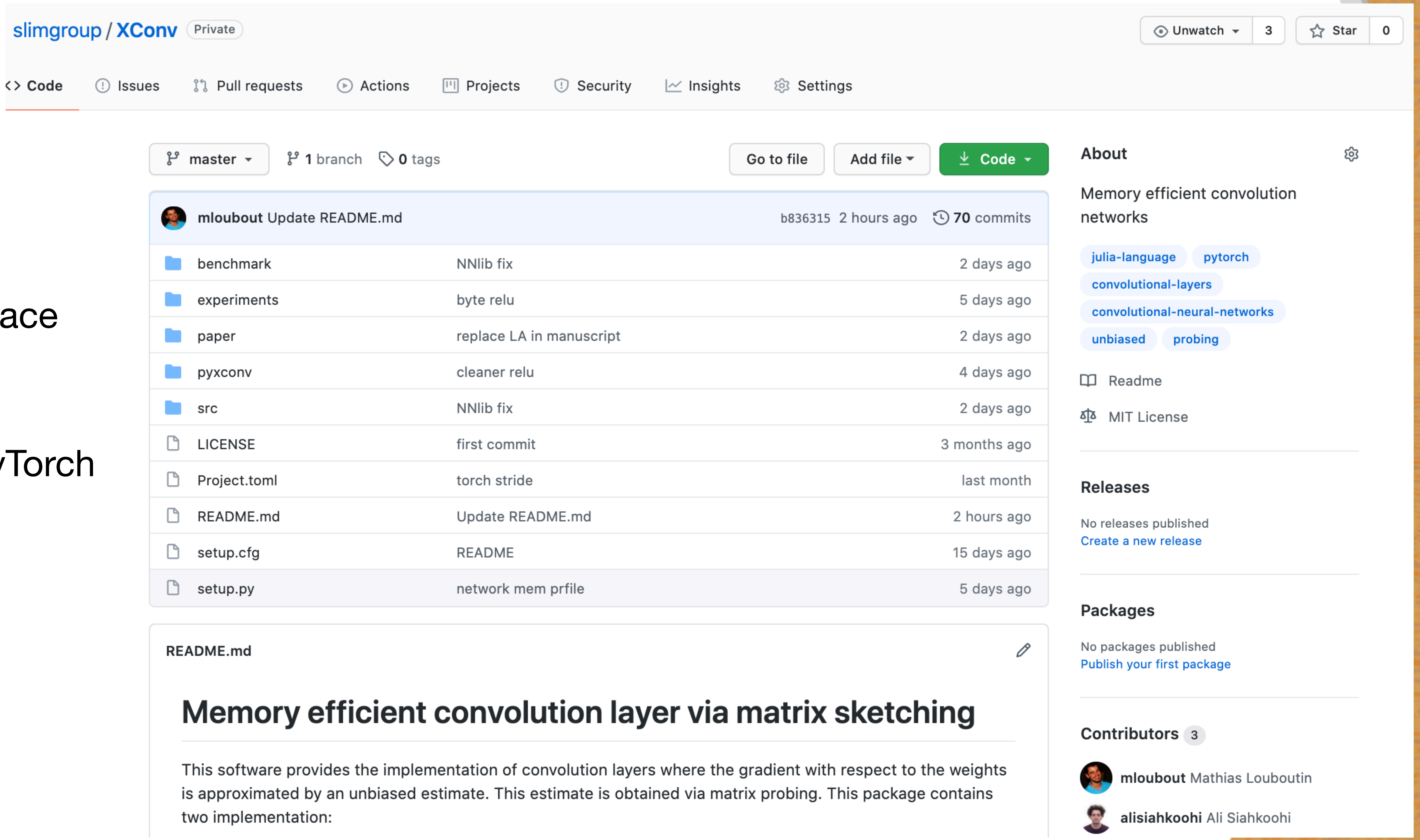
```
# Pytorch
Xconv2D(cc, cc, (3, 3), bias=False, ps=ps, stride=1, padding=1)
torch.nn.Conv2d(cc, cc, (3, 3), bias=False, padding=1, stride=1)

# Converts all pytorch convolutions to XConv in a network
convert_net(model, 'net', mode=‘conv’)

# Julia, Flux.jl overload
XConv.initXConv(ps, “EVGrad”)
```

SLIM ML4Seismic

XConv:

CPU/GPU codes for training w/ randomized trace estimation

- open-source MIT license
- optimized Python implementation for PyTorch
- Julia implementation for Flux
- easily integrated in existing networks
- https://github.com/slimgroup/XConv



This research was carried out with the support of Georgia Research Alliance and partners of the ML4Seismic consortium.