RÉPUBLIQUE FRANÇAISE

Liberté
Égalité
Fraternité

FRANCE 2030 PROGRAMME DE RECHERCHE NUMÉRIQUE POUR L'EXASCALE

Scaling deep solvers for imaging inverse problems on HPC systems

Joint work: Baptiste Besnard, Valentin Debarnot, Tobias Liaudat, Thomas Moreau, Benoît Malézieux, Triem Phan, Julian Tachella, Romain Vo

Scaling Deep Solvers for Imaging Inverse Problems on HPC Systems

Inverse problems: recover the signal behind an observation

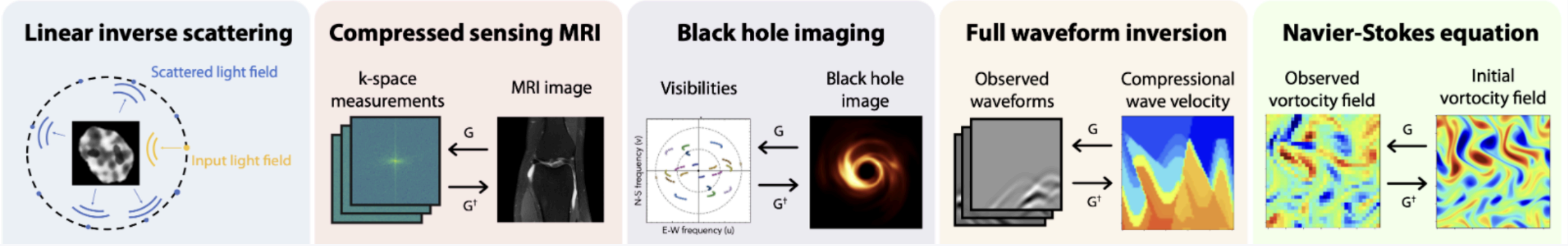


Figure adapted from Zheng et al., InverseBench, ICLR 2025.

SOTA solvers combine physics and learning

Measurements

Physics-based update
forward · adjoint · data-fidelity gradient

ITERATE

Learned prior / neural net
denoiser · score model · learned regularizer

Reconstruction

Plug-and-Play · PnP

Physics update + learned denoiser.

x_k → Physics → Denoiser → x_{k+1}

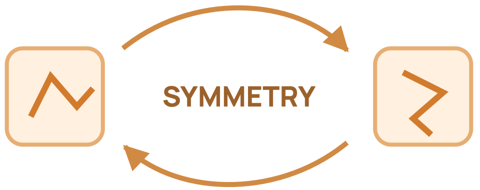
Unrolled optimization

A fixed number of trainable iterations.

P N → P N → P N

Learned priors · equivariant imaging

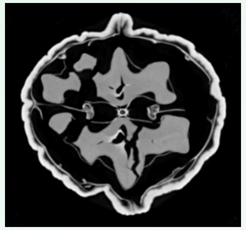
Consistency under known symmetries.



$f(gx) = g\,f(x)$

SOTA reconstruction quality comes with repeated, expensive physics and neural-network passes.

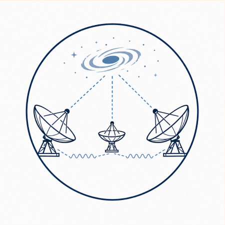
Applications...



Application · 01

3D tomography

GOAL	Reconstruct a 3D volume
MEASUREMENTS	2D projections (X-ray CT, electron tomography, ...)
PHYSICS	Tomographic projection and backprojection



Application · 02

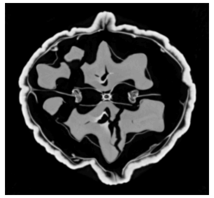
Radio interferometry

GOAL	Reconstruct an image of the sky
MEASUREMENTS	Complex visibilities from radio telescopes
PHYSICS	Non-uniform Fourier transform (NUFFT)

...need reconstruction methods that scale

Cryo-ET

Cryo-electron tomography



1000^3 voxels

in one reconstructed 3D volume

SKA

Radio interferometry

$20k^2$ pixels

in the reconstructed sky image

≈100M

measured visibilities

100M–1G pixel/voxel applications require distributed training and reconstruction.

What changes when 1 million pixels become 100 million?

Research scale

<1M

pixels / voxels

Large applications

>100M

pixels / voxels

AT APPLICATION SCALE

Runtime + Memory become the bottlenecks

At scale, the reconstruction no longer fits on one accelerator

APPLICATIONS

1000^3

Cryo-ET volume

$20k^2$

SKA sky image

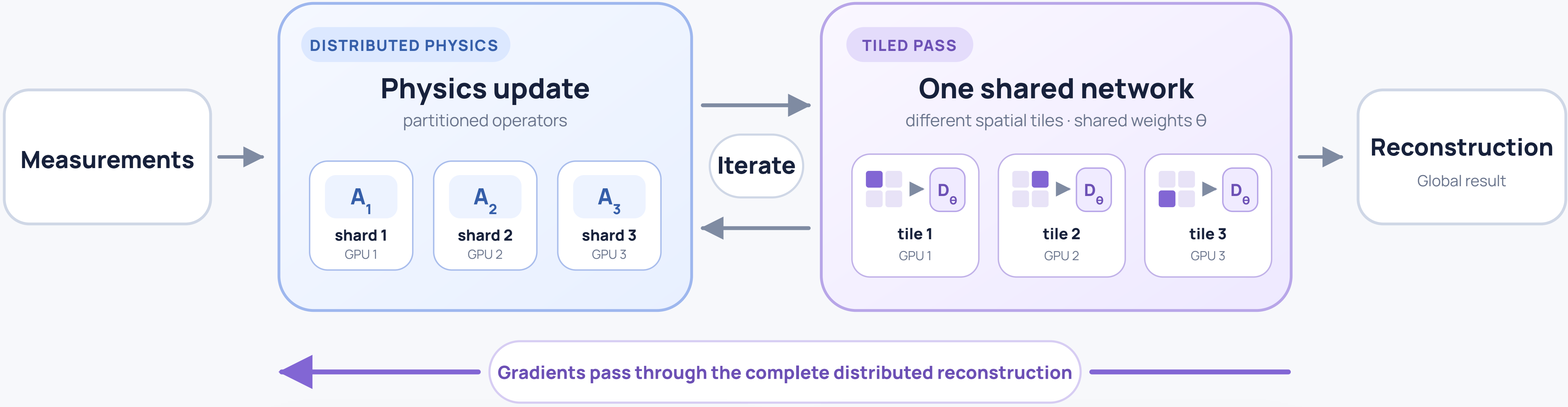
Huge signals + Non local physics + Neural networks

One accelerator is not enough

How do we distribute a solver whose computation alternates between large-scale physics and neural networks?

Scaling Deep Solvers for Imaging Inverse Problems on HPC Systems

Distribute both sides of the reconstruction loop



Checkpoint tiles

Recompute local activations instead of storing the full graph.

Train end to end

Keep autograd across the complete iterative architecture.

Both applications expose a natural physics partition

3D tomography

Projection angles

SPLIT THE ACQUISITION

θ_1	θ_2	θ_3

→

COMPUTE IN PARALLEL

GPU 1 · GPU 2 · GPU 3

SAME DECOMPOSITION PATTERN

Radio interferometry

W-stacks

SPLIT THE ACQUISITION

W_1	W_2	W_3

→

COMPUTE IN PARALLEL

GPU 1 · GPU 2 · GPU 3

02 · DISTRIBUTED SOLVERS FOR LARGE-SCALE INVERSE PROBLEMS

Overlap tiles, reuse one network, blend one result

01 Split the image with overlap

TILES + HALO →

02 Process locally · same denoiser Dθ · shared weights

↓ D_θ GPU 1

↓ D_θ GPU 2

↓ D_θ GPU 3

REDUCE →

03 Blend overlaps

Checkpoint only when memory is tight

During training, patch-batch activations can be recomputed in backward instead of retained.

`checkpoint_batches="auto"`

User-friendly API in DeepInverse

PYTHON

```python
from deepinv.distributed import DistributedContext, distribute
from deepinv.physics import stack

with DistributedContext() as ctx:
    x = x.to(ctx.device)

    physics = distribute(stack(*physics_list), ctx)
    denoiser = distribute(
        denoiser, ctx,
        patch_size=1024, overlap=64,
    )

    y = physics(x)
    x_0 = physics.A_adjoint(y)
    x_hat = denoiser(x_0, sigma=0.1)
```

01 · Runtime

`DistributedContext()`

Discovers rank and world size, selects the local device, and manages the process group.

02 · Entry point

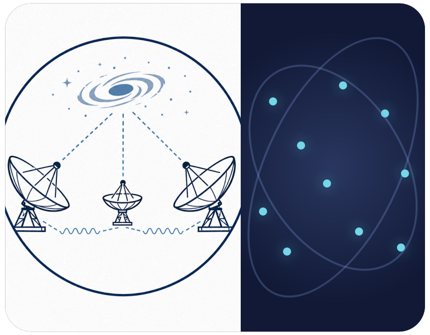
`distribute(…)`

Detects the object type and returns the matching distributed wrapper.

Execution scales with minimal changes to the codebase.

Radio imaging: one pipeline from visibilities to a trained solver

01 Data



`.ms`	complex visibilities
`.fits`	sky / reference
`.json`	cell size + metadata

→

02 Physics + initialization

A / A* ⇄ y visibilities

Dirty image

Source catalogue
PyBDSF initialization

→

03 Unrolled network + training

x_0

LAYER 1
source update
A / A* → D_θ

→

LAYER 2
source update
A / A* → D_θ

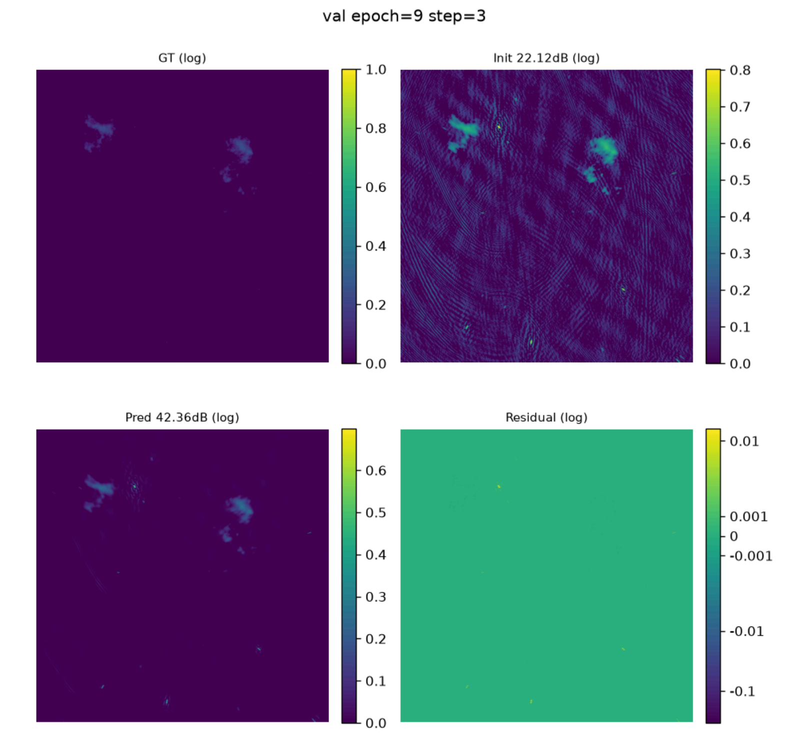
→

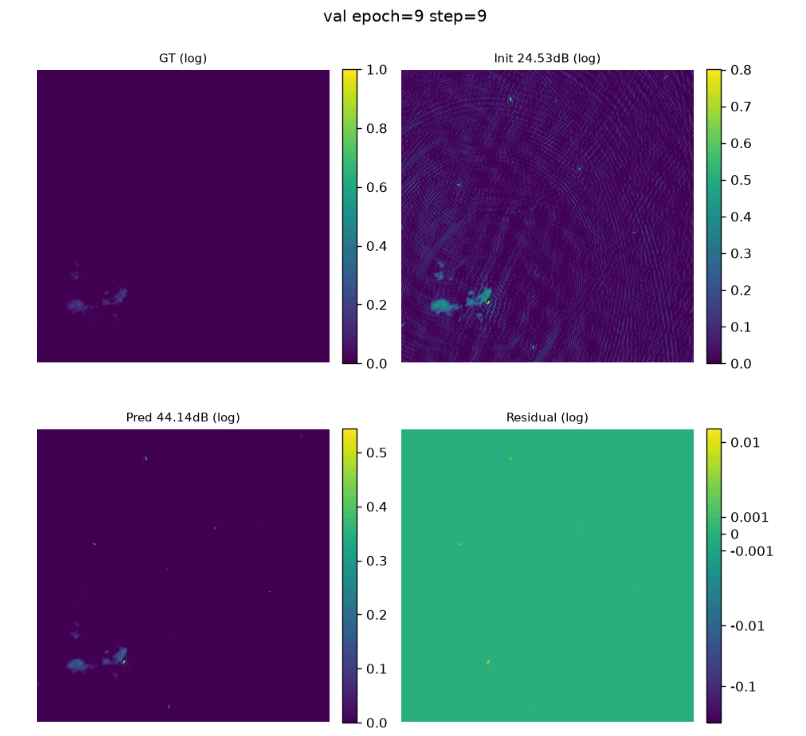
LAYER 3
source update
A / A* → D_θ

$\hat{x}$

Examples: radio imaging

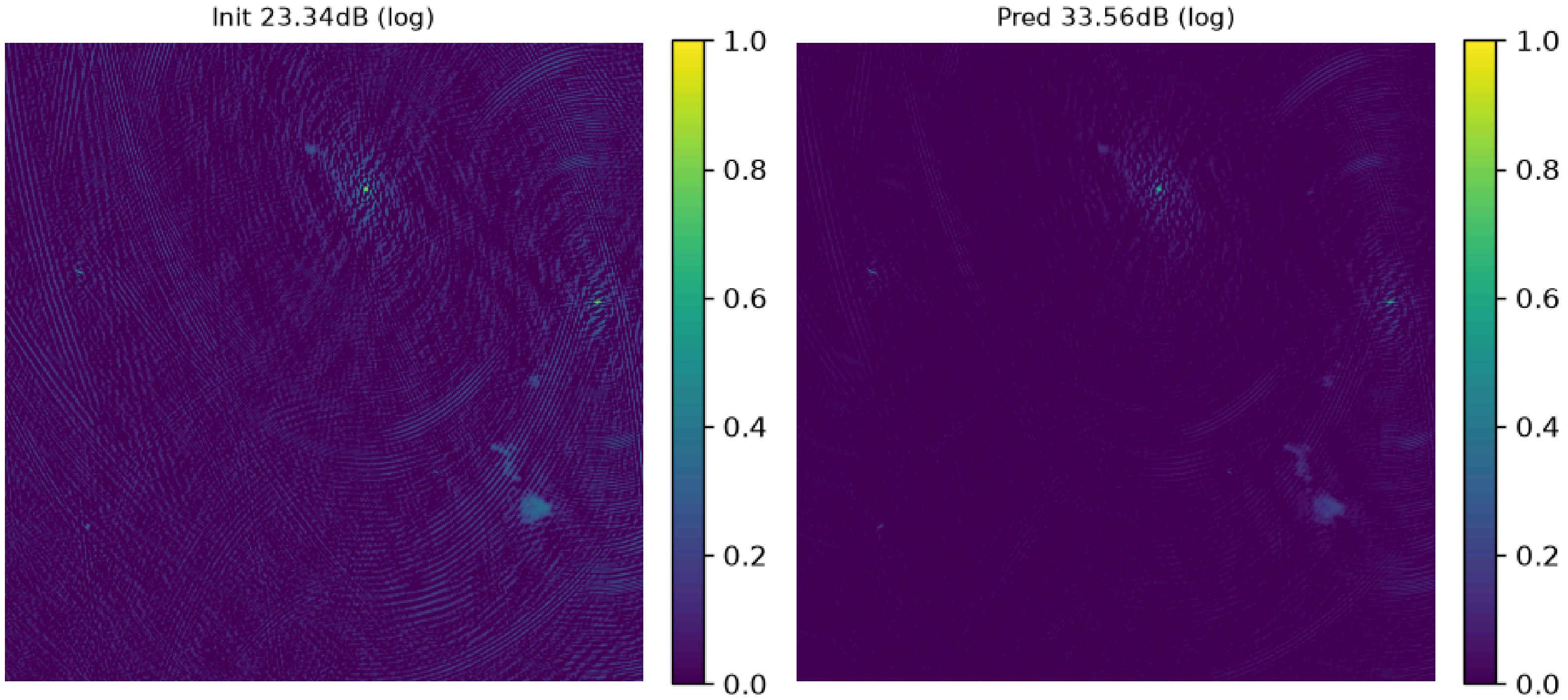
Validation images · 1024^2 pixels · after a few training epochs





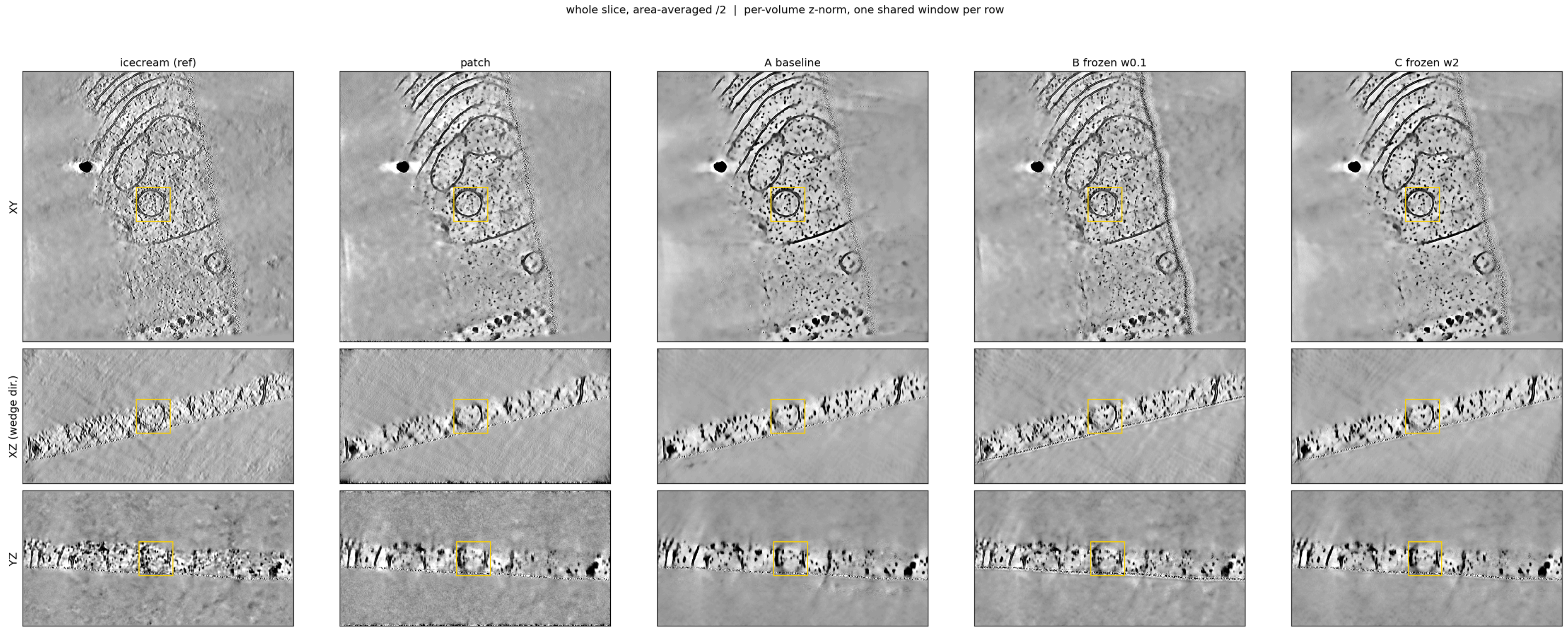
Examples: radio imaging

$20k^2$ image · 2 w-stacks · 2 AMD MI300A GPUs on Adastra | Forward + backward completed · image shown at network initialization



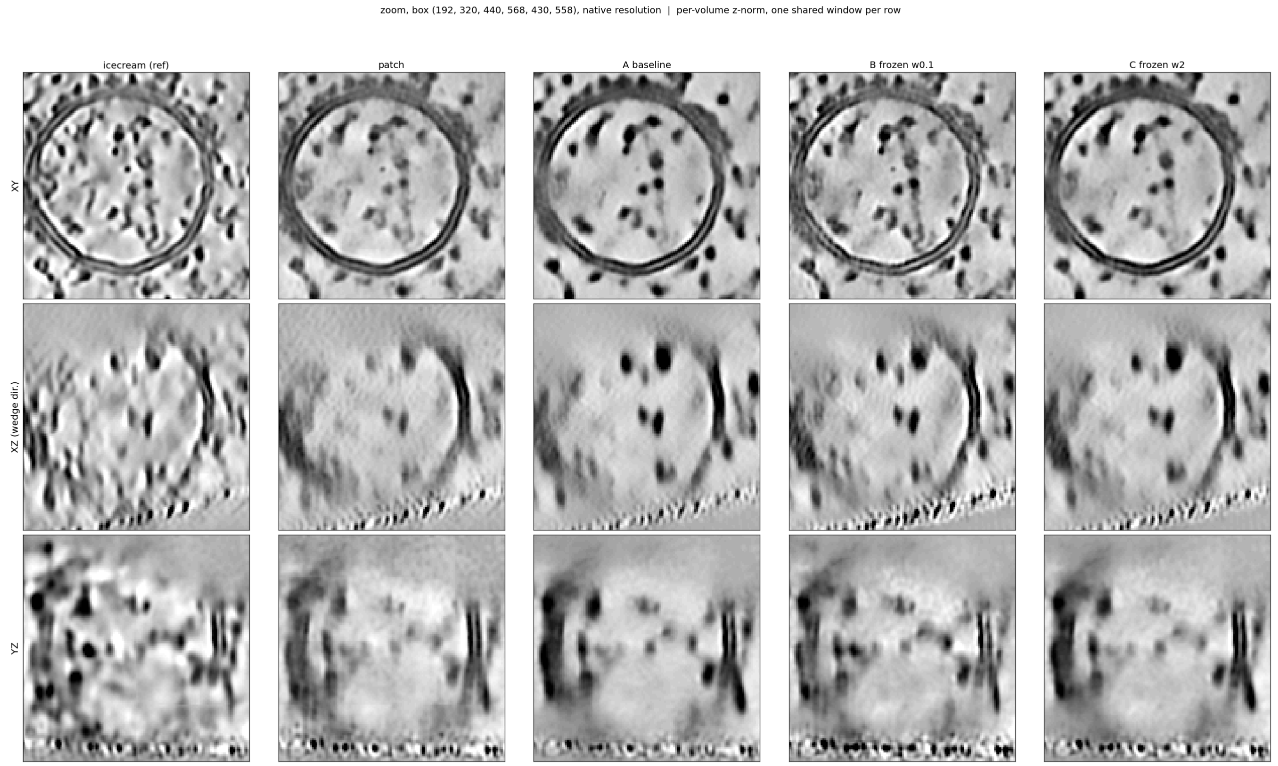
Examples: tomography

Self-supervised training · 1000 × 1000 × 500 volume



Examples: tomography

Self-supervised training · 1000 × 1000 × 500 volume



How do we evaluate these solvers fairly, reproducibly, and at scale?

Scaling Deep Solvers for Imaging Inverse Problems on HPC Systems

At scale, algorithmic and system choices are linked

MEASURE TOGETHER	Algorithmic questions	Systems questions
Outcome	Reconstruction quality	Throughput
Scale	Decomposition and patch size	Scaling w/ GPUs and problem size
Cost	Physics and network calls	Time breakdown (computation, communication)
Capacity	Signal context / overlap	Memory usage

Benchmark outputs: **quality + throughput** · **strong / weak scaling** · **time breakdown** · **peak memory**

One configuration becomes a complete scaling study

01 Configure the study

RESOURCE SWEEP · SIMPLIFIED

```
dataset:
  image_size: [2048, 4096, 8192]

solver:
  slurm_gres, slurm_nodes:
    - ["gpu:1", 1]
    - ["gpu:4", 4]
    - ["gpu:4", 16]
```

02 Build the experiment matrix

Benchopt

```
benchopt run … --config …
```

03 Submit jobs

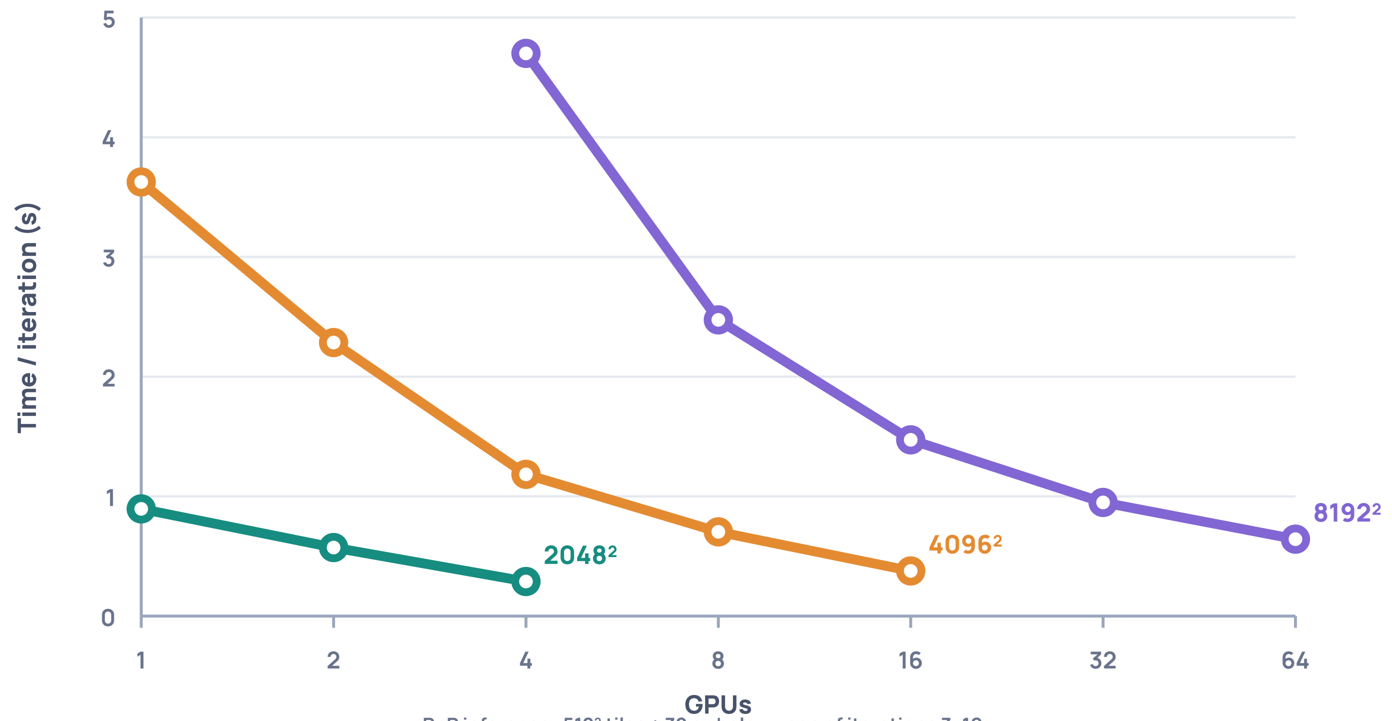
JOB 01 2K · 1 GPU	JOB 02 4K · 2 GPUs	JOB 03 4K · 4 GPUs
JOB 04 8K · 4 nodes	JOB 05 8K · 8 nodes	JOB 06 8K · 16 nodes

04 · COLLECT EVERY RUN

Measure the complete workflow

Inference	quality	time / iteration	memory	communication
Training	step throughput	forward + backward	memory	communication

Distributed inference scaling



PnP inference · 512^2 tiles + 32 px halo · mean of iterations 3–10

Inference · V100 32 GB

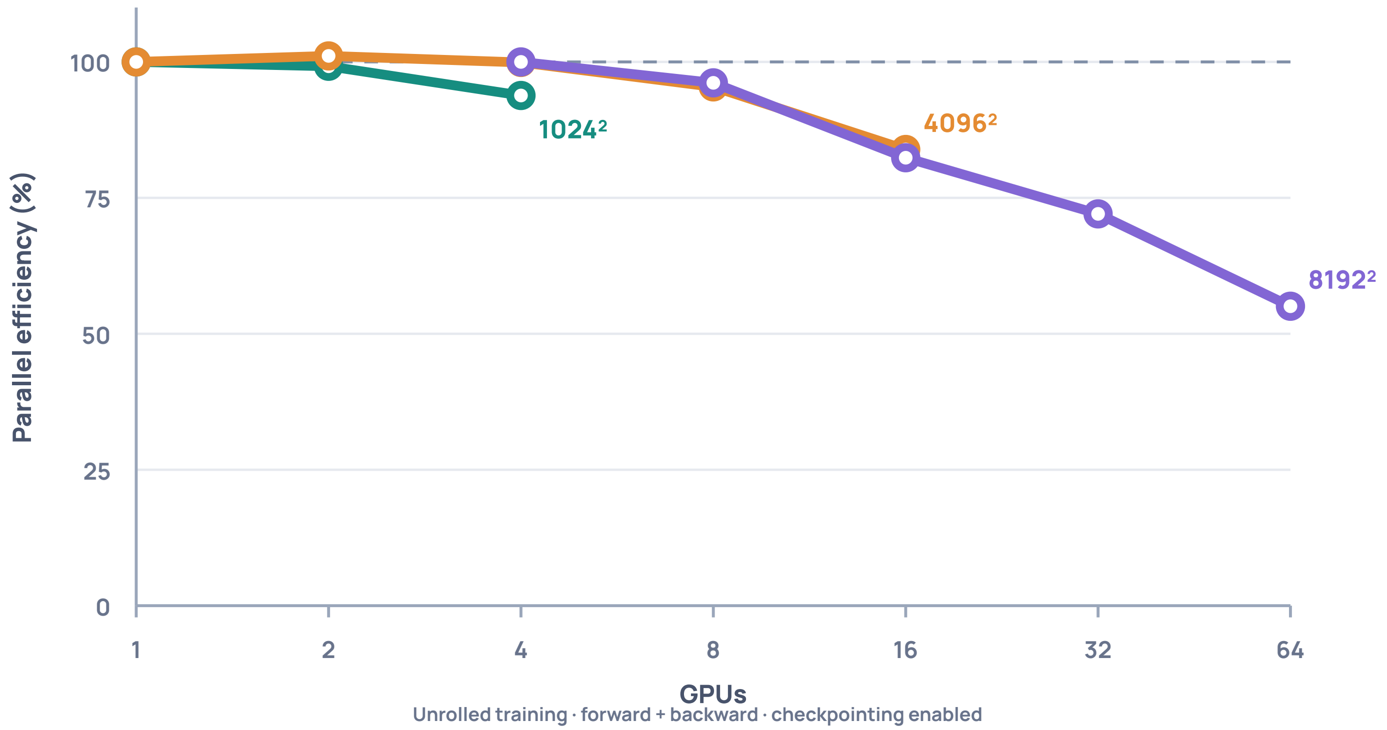
7.3× faster on the 8192^2 case
4 → 64 GPUs

4.70 s → 0.64 s
per PnP iteration

ΔPSNR ≤ 0.000929 dB
maximum deviation from the 1-GPU trajectory on cases with a reference

Larger images keep enough work per GPU to benefit longer.

Distributed forward backward scaling



Training · V100 32 GB

13.4× faster on the 4096^2 case
1 → 16 GPUs

101.2 s → 7.5 s
forward + backward step

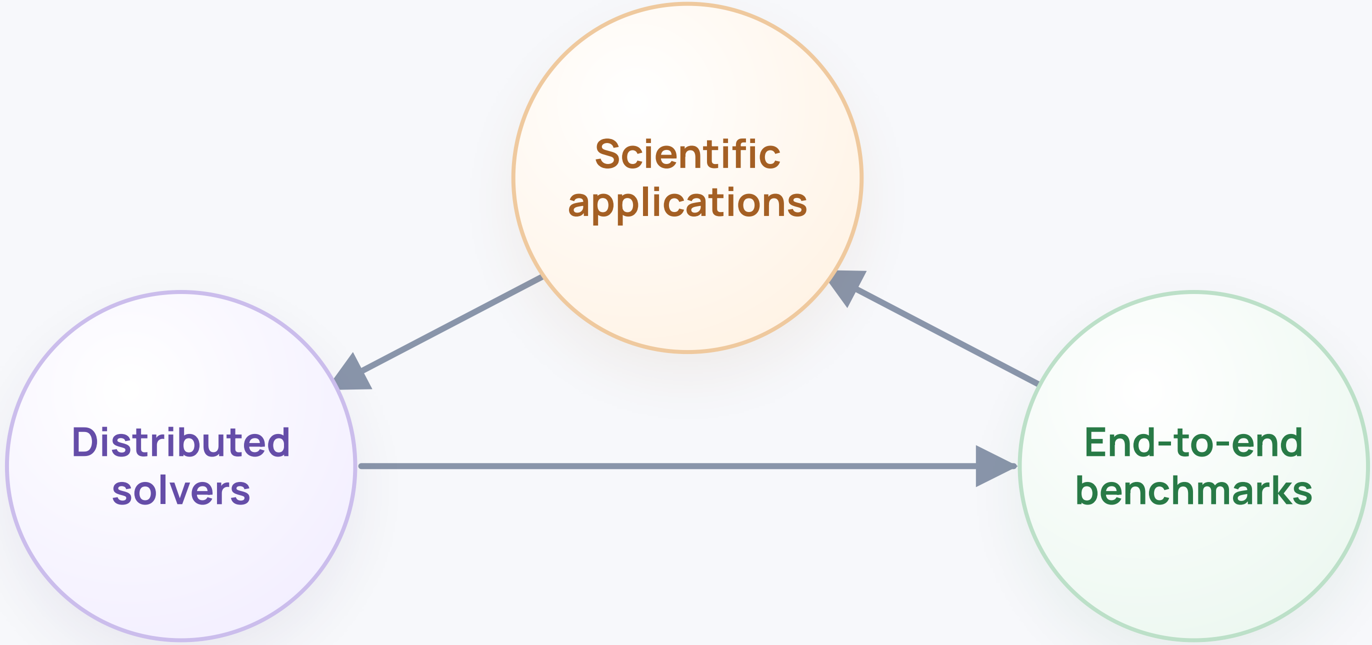
Scale by local workload—not by GPU count alone.

AMD porting depends on the scientific kernels

PyTorch	**Mostly portable**	No major CUDA–ROCm difference for our workloads.
cuFINUFFT	**CUDA-only**	Needs a maintained HIP implementation.
ASTRA	**Blocked on gfx942 (MI300A)**	Hardware linear filtering (tex2D / tex3D) is unavailable; fallback is a slower PyTorch-native projector.

Co-designing solvers, benchmarks, and scientific applications

NumPEx AI for Inverse Problems working group



EMAIL
benoit.malezieux@cea.fr

PROJECT WEBSITE
numpex.github.io/benchmark_invprob_largescale