

# Efficient Unrolled Networks for Large-Scale 3D Inverse Problems

## Supplementary Material

### A. Details about training configurations

Every learned model in Tabs. 2 and 3 is trained following the same global configuration: we use Adam optimizer [27] with an initial learning rate of  $10^{-4}$ . We train for  $10^5$  steps with a batch size of 4 (artificially increased via gradient accumulation if necessary). We use cosine annealing which decays the learning rate to  $10^{-8}$  at the end of training.

**Fitting the normal operator approximation.** We fit the normal operator approximation  $H(m, \lambda)$  on Gaussian random vectors. For each operator we train for 3000 steps with a batch size of 4 using Adam optimizer with a fixed learning rate of  $2.10^{-2}$  for CBCT and  $2.5 \times 10^{-2}$  for MC-MRI.

#### A.1. Walnut-CBCT

The dataset [10] contains 42 CBCT volumes of different walnuts. Each acquisition contains 1200 radiographies of size  $972 \times 768$  pixels, acquired over a full  $360^\circ$  circular trajectory. The reconstruction volumes have a size of  $501^3$  voxels. The ground-truth volumes are obtained by running an accelerated gradient descent scheme using the full-view data [10]. The train/val split contains 34 volumes, while the test split contains 8 volumes.

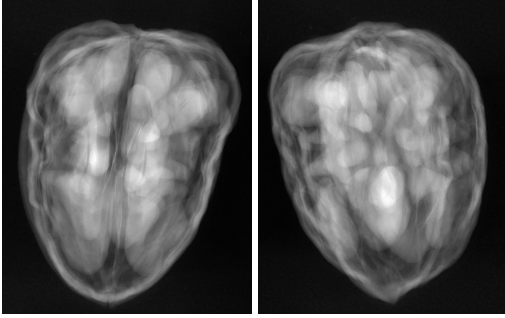


Figure 5. Illustrations of the **Walnut-CBCT** dataset. Examples of two radiographies.

**DRUNet initialization.** The 2D DRUNet [72] are initialized with the `deepinverse` weights [57]. During subsequent training, *i.e.* unrolling or post-processing, we fix the required noise value to  $\sigma = 0.03$ . The 3D DRUNet is initialized using the 2D DRUNet weights by inflating the 2D kernels to 3D. All kernels are of shape  $3 \times 3$ , for an equivalent  $3 \times 3 \times 3$  kernel, we copy the 2D weights to the central slice along the depth dimension and initialize the other slices with zeros. For upsampling and downsampling layers of shape  $2 \times 2$ , we duplicate the 2D weights along the depth dimension to form  $2 \times 2 \times 2$  kernels, and normalize them to preserve the overall kernel norm.

**Ours.** Training **our** method, *i.e.* 3D DRUNet unrolled for  $K = 3$  iterations with **domain partitioning** (cuboid patch size of  $8 \times 384^2$ ) takes approximately 185 hours on a single NVIDIA H100 GPU with 80GB of video memory. Combining **domain partitioning** with **normal operator approximation** further reduces the training time to approximately 135 hours on the same hardware.

**TV.** We solve (2) with  $g(x) = \|\nabla x\|_1$  using FISTA [5] for 1000 iterations with  $\lambda = 0.2$ , and step size  $\eta = 1/L$ , where  $L$  is the spectral norm of  $A^\top A$  estimated via power iteration.

**PnP- $\alpha$ PGD.** For 2D and 3D, we respectively use the post-processing DRUNet trained on 2D slices and 3D patches as described above as the prior. More precisely, we run an accelerated PGD scheme [5] for  $K = 40$  iterations with step size  $\eta = 1.0/L$ . For stability reasons, we relax the network evaluation with  $D_\alpha = (1 - \alpha)\text{Id} + \alpha D_\phi$ , where  $D_\phi$  is the corresponding DRUNet. Similar to [65], we define  $\alpha = \frac{\eta\lambda}{1+\eta\lambda}$ , where  $\lambda$  is the weight of the implicit prior defined by the artifact removal network. We set  $\lambda = 10$ . for both 2D and 3D.

**INR.** Training an `instant-ngp` requires specific library dependencies. We use the specific CUDA implementation from `tiny-cuda-nn` [39] and the ray-casting code from `nerfacc` [33]. The INR uses a hash grid encoding [40] with 16 levels and a capacity of  $T = 2^{21}$ , a feature dimension of 2, a base resolution of 16, and a finest resolution of 256. The MLP has 2 hidden layers with 64 hidden units each. We train the INR by stochastic coordinate descent with Adam optimizer. The learning rate is initialized to  $10^{-4}$  and decayed to  $10^{-8}$  following a cosine scheduling for 25k steps. We cast 2048 rays per step, compute the associated data-fidelity for these rays and backpropagate the gradients to update the MLP and hash grid parameters.

**DPIR[RAM].** We use the PnP Half-Quadratic Splitting (HQS) method from [72] with a Reconstruct Anything Model (RAM) prior [60]. We run it for  $K = 20$  iterations, with  $\lambda = 1/0.23$  and a decreasing noise schedule  $\sigma_k$  starting from  $\sigma_1 = 49/255$ . to  $\sigma_K = 2.10^{-3}$ .

#### A.2. Calgary-Campinas MC-MRI

The dataset [52] contains 59 fully-sampled Cartesian Multi-Coil MRI acquisitions of different brains. Each acquisition contains 12 coils with a k-space of size  $256 \times 218 \times 170$ . The k-space data is provided in a hybrid format, *i.e.* x-ky-kz, where a 1D FFT has already been applied along the frequency-encoded direction. The acceleration is performed

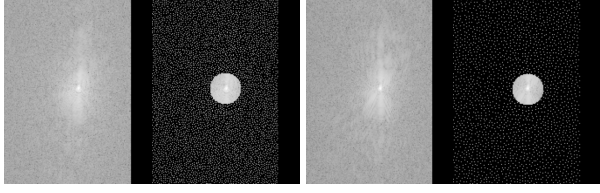


Figure 6. Illustrations of the **Calgary-Campinas** dataset. Examples of k-space measurements. (*left*) Slice of fully-sampled k-space (1st coil), and subsequently undersampled k-space with acceleration rate of 5. (*right*) Slice of fully-sampled k-space (1st coil) and, corresponding undersampled k-space with acceleration rate of 10.

along the phase encoding and slice encoding directions ( $k_y$  and  $k_z$ ). This reduces the 3D problem to a set of independent 2D problems along the frequency-encoded direction ( $x$ ), which allows us to train standard unrolled network without **domain partitioning**. We use a set of 48 samples for the train/val split and 21 samples for the test split. The ground-truth images are obtained by applying *root-sum-of-squares* (RSS) reconstruction on the fully-sampled k-space data. The sensitivity maps are estimated as in [44] by applying inverse FFT on the fully-sampled centered region of k-space data.

**DRUNet initialization.** The `deepinverse` library does not provide pretrained weights for complex data so both the 2D DRUNet and 3D DRUNet are trained from scratch.

**Unrolled.** As opposed to Walnut-CBCT, we can compare our method to a standard unrolled network without **domain partitioning** since the problem reduces to a set of independent 2D problems along the frequency-encoded direction ( $x$ ). We use a sub-problem size of  $8 \times 218 \times 170$  which maximizes VRAM usage during training. Training the 3D unrolled network for  $K = 5$  iterations takes approximately 60 hours on a single NVIDIA H100 GPU with 80GB of video memory.

**Ours.** Training **our** method, *i.e.* 3D DRUNet unrolled for  $K = 5$  iterations with **domain partitioning** (cuboid patch size of  $8 \times 128^2$ ) takes approximately 17 hours and twice less memory than standard unrolling on a single NVIDIA H100 GPU with 80GB of video memory. Combining **domain partitioning** with **normal operator approximation** does not reduce training time as the original MC-MRI forward operator is already efficient to compute.

**TV.** We solve (2) with  $g(x) = \|\nabla x\|_1$  using FISTA [5] for 1000 iterations with  $\lambda = 7.4 \times 10^{-4}$ , and step size  $\eta = 1.$ , as the MC-MRI forward already has a spectral norm of 1.

**PnP- $\alpha$ PGD.** For 2D and 3D, we respectively use the post-processing DRUNet trained on 2D slices and 3D patches as described above as the prior. More precisely, we run an accelerated PGD scheme [5] for  $K = 12$  iterations with step size  $\eta = 1.0$ . For stability reasons, we relax the network

evaluation with  $D_\alpha = (1 - \alpha)\text{Id} + \alpha D_\phi$ , where  $D_\phi$  is the corresponding DRUNet. Similar to [65], we define  $\alpha = \frac{\eta\lambda}{1+\eta\lambda}$ , where  $\lambda$  is the weight of the implicit prior defined by the artifact removal network. We set  $\lambda = 2.10^{-6}$  for both 2D and 3D.

**INR.** As `tiny-cuda-nn` [39] does not natively support complex computation, required for FFT computations, we implement a similar grid-based INR architecture using PyTorch. The encoding is a latent grid of size  $80^3$  with 8 features on each vertex. The MLP has 2 hidden layers with 64 hidden units each. We train the INR by gradient descent with Adam optimizer for 1000 steps. The learning rate is fixed and set to  $1e - 2$ . At each step we compute the data-fidelity on the full volume and backpropagate the gradients to update the MLP and grid parameters.

**DPIR[RAM].** We use the PnP Half-Quadratic Splitting (HQS) method from [72] with a Reconstruct Anything Model (RAM) prior which natively provides the possibility to process complex data [60]. We run it for  $K = 20$  iterations, with  $\lambda = 1/0.23$  and a decreasing noise schedule  $\sigma_k$  starting from  $\sigma_1 = 5.10^{-2}$  to  $\sigma_K = 2.10^{-3}$ .

## B. Details about test-time configurations

In this section we detail the sub-procedure of Algorithm 1, *i.e.* the procedure to get  $\tilde{x}$ , namely we deploy a standard unrolling scheme by evaluating the prior on sequential patches.

We break each step (4) in  $R_\phi(y, A)$  as follows: (i) a first gradient descent step, computed on the full volume and full problem (1)  $x'_k = x_k - \eta \nabla_{x_k} d(Ax_k, y)$ , (ii) followed by a prior step on sequential patches,  $x_{k+1,p} = D_\phi(S_p x'_k)$ , where  $S_p$  extracts the  $p$ -th patch from the full volume. Finally, we aggregate the processed patches to form the full volume  $x_{k+1}$ . We repeat this procedure for  $K$  iterations to get  $\tilde{x} = x_K$ .

As opposed to training, this procedure is possible at test-time since we do not need to store activations for each processed patch, allowing us to process arbitrarily large volumes with limited memory.

**Patch-based strategy.** At test-time, we observe that the trained networks are robust to the choice of patch size. For **MC-MRI**, we only deploy a standard unrolling procedure and use patches of size  $8 \times 218 \times 170$  with 3D methods and  $218 \times 170$  with 2D methods, *i.e.* full spatial dimensions in  $H \times W$ . We either use a stride of size 4 along the depth dimension in 3D, or evaluate slice-by-slice in 2D. Then we aggregate to build a prediction of size  $256 \times 218 \times 170$ . For **CBCT**, we use two different strategies. When computing the estimation of the ground-truth  $\tilde{x}$  with the standard unrolling procedure, we use patches of size  $8 \times 501^2$  with 3D methods and  $501^2$  with 2D methods. We either use a stride of size 4 along the depth dimension in 3D, or evaluate slice-by-

slice in 2D. Then we aggregate to build a prediction of size  $501^3$ . In the second part, we deploy unrolling with domain partitioning and use patches of size  $8 \times 384^2$  or  $384^2$  for 3D and 2D methods respectively.

**Evaluation strategy.** When evaluating the methods on the Walnut-CBCT dataset, we crop the ground-truth and predicted volumes to the central  $300^3$  voxels to avoid boundary artifacts and focus on part of the volumes that contains well-defined material.

## C. Additional results

### C.1. Gradient computation in unrolled methods

Let us recall the post-processing optimization problem:

$$\begin{aligned} \mathcal{L}_{\text{POST}}(\phi) &= \mathbb{E}_{\mathbf{x}^*, \mathbf{y}} \|\mathbf{R}_\phi(\mathbf{y}, \mathbf{A}) - \mathbf{x}^*\|_2^2 \\ \text{with } \mathbf{R}_\phi(\mathbf{y}, \mathbf{A}) &= \mathbf{D}_\phi(\mathbf{x}_0), \text{ and e.g. } \mathbf{x}_0 = \mathbf{A}^\top \mathbf{y} \end{aligned} \quad (20)$$

The gradient computation during post-processing training does not involve the normal operator  $\mathbf{A}^\top \mathbf{A}$ :

$$\nabla_\phi \mathcal{L}_{\text{POST}} = \frac{\partial \mathcal{L}_{\text{POST}}}{\partial \hat{\mathbf{x}}}^\top \frac{\partial \mathbf{D}_\phi}{\partial \phi} \bigg|_{\mathbf{x}_0}. \quad (21)$$

On the other hand, the gradient  $\nabla_\phi \mathcal{L}_{\text{UNR}}$  accumulates contributions from all iterations, especially the data-step, which requires the full volume to be evaluated. We write the unrolled optimization problem for  $K$  as:

$$\begin{aligned} \mathcal{L}_{\text{UNR}}(\phi) &= \mathbb{E}_{\mathbf{x}^*, \mathbf{y}} \|\mathbf{R}_\phi(\mathbf{y}, \mathbf{A}) - \mathbf{x}^*\|_2^2 \\ \text{with } \mathbf{x}_{k+1} &= \mathbf{D}_\phi(\mathbf{z}_k), \mathbf{z}_k = \mathbf{x}_k - \eta \nabla_{\mathbf{x}_k} d(\mathbf{A}\mathbf{x}_k, \mathbf{y}) \\ \text{and } \mathbf{R}_\phi(\mathbf{y}, \mathbf{A}) &= \mathbf{x}_K(\phi), \mathbf{x}_0 = \mathbf{A}^\top \mathbf{y} \end{aligned} \quad (22)$$

Then it follows that the gradient of the loss  $\mathcal{L}_{\text{UNR}}(\phi)$  with respect to the parameters  $\phi$  is given by:

$$\begin{aligned} \nabla_\phi \mathcal{L}_{\text{UNR}} &= \sum_{k=0}^{K-1} \frac{\partial \mathcal{L}_{\text{UNR}}}{\partial \mathbf{x}_{k+1}}^\top \frac{\partial \mathbf{D}_\phi}{\partial \phi} \bigg|_{\mathbf{z}_k} \\ \frac{\partial \mathcal{L}_{\text{UNR}}}{\partial \mathbf{x}_{k+1}} &= \frac{\partial \mathcal{L}_{\text{UNR}}}{\partial \mathbf{x}_K}^\top \prod_{j=k+1}^{K-1} \frac{\partial \mathbf{x}_{j+1}}{\partial \mathbf{x}_j}, \\ \text{with } \frac{\partial \mathbf{x}_{j+1}}{\partial \mathbf{x}_j} &= \frac{\partial \mathbf{D}_\phi}{\partial \mathbf{z}_j} \bigg|_{\mathbf{z}_j} \frac{\partial \mathbf{z}_j}{\partial \mathbf{x}_j}, \\ \text{and } \frac{\partial \mathbf{z}_j}{\partial \mathbf{x}_j} &= (\mathbf{I} - \eta \mathbf{A}^\top \mathbf{A}). \end{aligned} \quad (23)$$

### C.2. Influence of the patch size on the performances

In Figs. 7 and 8 we provide the evolution of average PSNR and time complexity against peak memory consumption

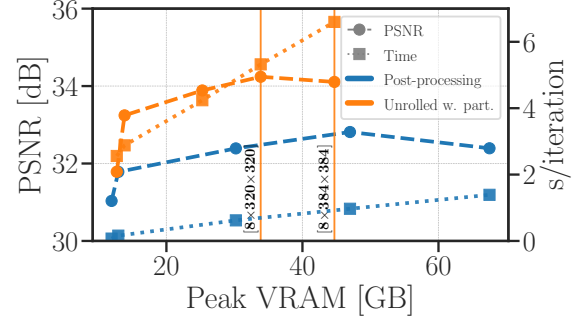


Figure 7. **Walnut-CBCT** - Average PSNR and time complexity against peak memory consumption during training. We vary the VRAM budget by changing the patch size used during domain partitioning. Larger patches lead to better performance at the cost of higher memory consumption. We do not show the complexity of standard unrolling (without partitioning) as a single H100 GPU is not sufficient for training it.

during training for different patch sizes on Walnut-CBCT and Calgary-Campinas MC-MRI datasets respectively.

We observe that larger patches lead to better performance at the cost of higher memory consumption. We note that on both dataset, for the same memory budget, our method with domain partitioning outperforms the standard post-processing. On Walnut-CBCT, we use patch sizes in  $[(384 \times 384), (8 \times 128^2), (8 \times 256^2), (8 \times 320^2), (8 \times 384^2)]$ , while on Calgary-Campinas MC-MRI, we use patch sizes in  $[(218 \times 170), (8 \times 64^2), (8 \times 128^2), (8 \times 160^2), (8 \times 218 \times 170)]$ .

Interestingly, we observe that on the MC-MRI experiment, performance starts to plateau for patch sizes larger than  $8 \times 128^2$ , suggesting that the standard unrolling with no partitioning does not bring significant benefits compared to our method with domain partitioning for this specific problem and network complexity.

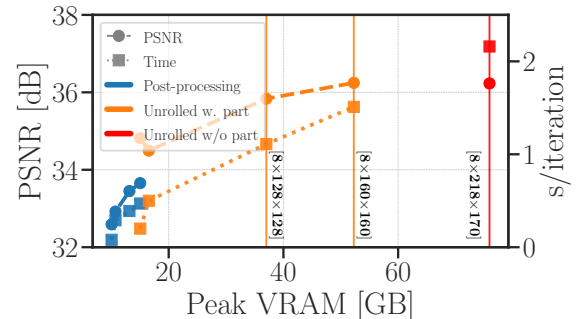


Figure 8. **Calgary-Campinas MC-MRI** - Average PSNR and time complexity against peak memory consumption during training. We vary the VRAM budget by changing the patch size used during domain partitioning. Larger patches lead to better performance at the cost of higher memory consumption.

### C.3. Normal operator approximations

**Walnut-CBCT.** In Fig. 9, we provide additional illustrations of the normal operator approximation on Walnut-CBCT. We show a slice of the original volume  $x$ , the exact normal operator evaluation  $A^\top Ax$ , and the approximated normal operator  $H(m, \lambda)x$ . We see that  $\lambda$ , which corresponds to a filter in the Fourier domain, exhibits patterns predicted by the Fourier slice theorem, *i.e.* it performs sampling along radial directions.

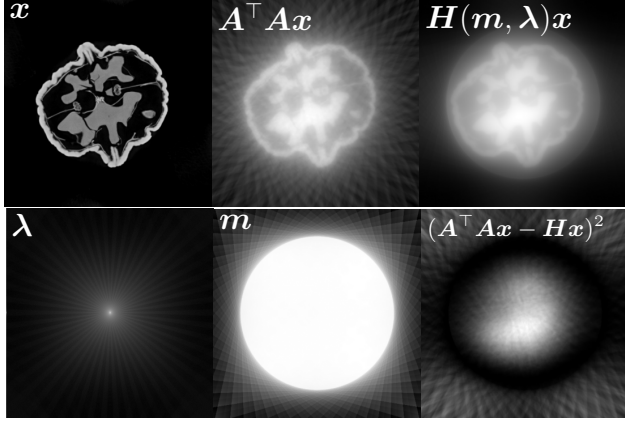


Figure 9. Illustrations of the normal operator approximation on **Walnut-CBCT**. (*top row*) Original volume slice  $x$ , exact normal operator evaluation  $A^\top Ax$ , and approximated normal operator  $H(m, \lambda)x$ . (*bottom row*) Learned filter  $\lambda$ , learned mask  $m$ , and squared approximation error  $(A^\top Ax - Hx)^2$ .

**Calgary-Campinas MC-MRI.** In Fig. 10, we provide additional illustrations of the normal operator approximation on Calgary-Campinas MC-MRI. We show a slice of the original volume  $|x|$ , the exact normal operator evaluation  $|A^H Ax|$ , and the approximated normal operator  $|H(m, \lambda)x|$ . We see that  $\lambda$ , which corresponds to a filter in the Fourier domain, performs the same Fourier masking operation as the original acceleration pattern (Fig. 6). The spatial modulation  $m$  performs an operation similar to coil sensitivity maps.

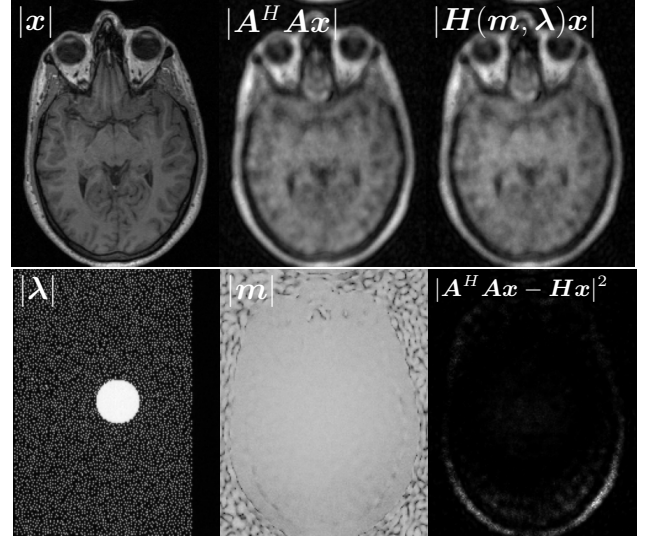


Figure 10. Illustrations of the normal operator approximation on **Calgary-Campinas**. (*top row*) Original volume slice  $|x|$ , exact normal operator evaluation  $|A^H Ax|$ , and approximated normal operator  $|H(m, \lambda)x|$ . (*bottom row*) Learned filter  $|\lambda|$ , learned mask  $|m|$ , and squared approximation error  $|A^H Ax - Hx|^2$ .