Analysis Report on *Interpretable Lightweight Transformer via Unrolling of Learned Graph Smoothness Priors*

Kenneth Browder [1] **Fabien Lagnieu** [1]

Abstract

This report analyzes the paper *Interpretable Lightweight Transformer via Unrolling of Learned Graph Smoothness Priors* (Do et al., 2024). The authors propose an efficient and interpretable transformer-like model for graph-signal interpolation. Their method uses unrolled optimization algorithms based on graph smoothness priors. The first section presents its main contributions, methodology, and experimental results, and describes the reproduction of one key experiment and an additional experiment. The second section examines a related paper, discusses its relevance, and compares its approach with the first.

Section 1: Technical Review of the Main Paper

Interpretable Lightweight Transformer via Unrolling of Learned Graph Smoothness Priors
Tam Thuc Do, Parham Eftekhar, Seyed Alireza Hosseini, Gene Cheung, Philip Chou - Published in June 2024.

1.1. Introduction

Transformer architectures are central to modern machine learning, particularly in natural language processing and computer vision. Despite their success, they face two main limitations: high computational cost and lack of interpretability. These issues restrict their use in resource-constrained settings or applications requiring transparency.

The paper proposes a simple and effective solution by combining Graph Signal Processing (GSP) with neural network architectures. GSP models data structured as graphs by leveraging their relational information. The authors unroll optimization algorithms for graph signal interpolation into a neural network, resulting in a transformer-like model that is both lightweight and interpretable.

[*]Equal contribution [1]Institut Polytechnique de Paris, France. Correspondence to: Kenneth Browder <kenneth.browder@ip-paris.fr>, Fabien Lagnieu <fabien.lagnieu@polytechnique.edu>.

Machine Learning with Graphs course - IPP,
Palaiseau, France, 2025. Copyright 2025 by the author(s).

1.2. State of the Art

Graph-based learning has made important progress in recent years, so many methods focus on learning from data structured as graphs. GSP provides a solid framework for analyzing such data, it extends traditional signal processing tools to irregular graph domains and allows processing signals that live on nodes of a graph by using the graph structure (Shuman et al., 2013).

Two key tools in GSP are the Graph Laplacian Regularizer (GLR) and Graph Total Variation (GTV). GLR promotes smoothness, encouraging similar values for connected nodes (Ortega et al., 2018). In contrast, GTV allows sharp changes and preserves edges and discontinuities (Cheung et al., 2018).

In the analyzed paper, GLR and GTV guide both graph learning and signal interpolation, ensuring smoothness and edge preservation in the reconstructed signals.

1.3. Main Contributions of the Proposed Method

The proposed method rethinks the transformer paradigm by integrating graph learning and optimization unrolling into a unified framework for graph signal interpolation. Its core contributions are outlined below.

1.3.1. Overview of the Transformer-like Architecture

The architecture alternates between learning a similarity graph and updating the interpolated signal. At each layer, the graph and signal are refined through an unrolled optimization process. This iterative structure mimics transformers but relies on graph-based operations, ensuring interpretability and lower complexity.

1.3.2. Graph Learning via Mahalanobis Distance

The graph learning module constructs a similarity graph by computing pairwise distances between nodes. It uses the *Mahalanobis distance* (Hu et al., 2020), which measures the similarity between two feature vectors $\mathbf{f}_i$ and $\mathbf{f}_j$.

The *Mahalanobis distance* between two nodes i and j is defined as: $d(i,j) = (\mathbf{f}_i - \mathbf{f}_j)^\top \mathbf{M}(\mathbf{f}_i - \mathbf{f}_j)$, where $\mathbf{M}$ is a learnable, positive semi-definite matrix that adapts the metric to the data during training. And the edge weights $w_{i,j}$ between nodes are computed as: $w_{i,j} = \exp(-d(i,j))$. A local softmax normalizes outgoing edges so their weights sum to one, similar to transformer attention.

This approach guarantees the symmetry of the learned graph and provides an interpretable alternative to the key-query-value attention commonly used in standard transformers.

1.3.3. Optimization Unrolling for Signal Interpolation

Once the graph is defined, the model performs signal interpolation guided by graph smoothness priors. Two optimization strategies are employed: the *Conjugate Gradient* (CG) method used to minimize the ℓ_2-norm GLR (Shewchuk, 1994) and the *Alternating Direction Method of Multipliers* (ADMM) applied to minimize the ℓ_1-norm GTV (Wang & Shroff, 2017).

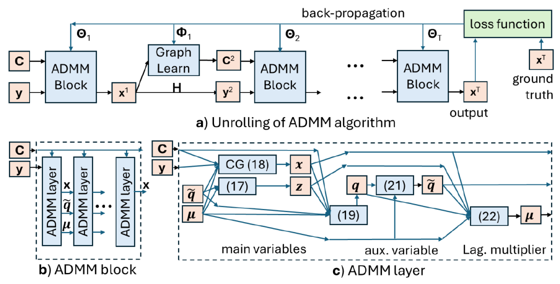


Figure 1. Unrolling of GTV-based signal interpolation algorithm (Do et al., 2024).

Each iteration of these algorithms is implemented as a neural network layer, enabling end-to-end training through backpropagation. The process is illustrated in Figure 1; a schematic algorithm is also provided in Appendix A.

1.3.4. Interpretability and Efficiency

Each layer performs a well-defined, interpretable operation, replacing traditional self-attention with a graph learning module guided by graph smoothness priors, avoiding the opacity of key-query-value mechanisms.

Using unrolled optimization and graph-based operations, the model eliminates large parameter matrices, reducing computational cost while maintaining strong performance.

This model advances the state-of-the-art by showing that transformer-like architectures can achieve competitive performance with fewer parameters, while offering greater interpretability than conventional attention mechanisms.

1.4. Experimental Validation

1.4.1. Experimental Setup

All experiments are implemented in PyTorch. Models are trained on the DIV2K dataset, containing 800 high-resolution (HR) training images and 100 validation images. To reduce computational cost, the authors extract 64×64 pixel patches and sample only 1% to 4% of all available patches for training. Testing is performed on three standard benchmarks: McM (Zhang et al., 2011), Kodak (Eastman Kodak, 1993), and Urban100 (Huang et al., 2015).

Two tasks are considered: *Image Interpolation* (reconstructing high-resolution images from sparsely sampled data) and *Demosaicking* (reconstructing full-color images from Bayer-patterned inputs).

Performance is evaluated using Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) (Wang et al., 2004).

1.4.2. Results on Image Interpolation

The authors evaluate their models on image interpolation tasks. The proposed models, uGLR and uGTV, are compared to established methods including MAIN (Ji et al., 2020) and SwinIR-lightweight (Liang et al., 2021), as well as a simple Bicubic interpolation baseline.

uGTV consistently achieves the best PSNR and SSIM scores across all datasets. Notably, it outperforms MAIN and SwinIR, despite having significantly fewer parameters (319k versus 10.9M for MAIN and 0.9M for SwinIR).

1.4.3. Results on Demosaicking

For demosaicking, uGTV shows competitive performance with a lightweight design. Table 1 presents selected results on the Kodak and Urban100 datasets. To maintain readability, only the most relevant models (Bilinear, RSTCANet-B/S (Xing et al., 2022), uGLR, and uGTV) are reported.

uGTV achieves top PSNR and SSIM on Urban100 and closely matches RST-S on Kodak, while using nearly ten times fewer parameters. These results confirm uGTV's balance between accuracy and efficiency. Its performance generalizes well across datasets without complex architectures.

Table 1. Demosaicking performance (PSNR / SSIM) for selected models, trained on 10k samples from DIV2K.

Method	Params #	Kodak	Urban100
Bilinear	-	28.22 / 0.890	24.18 / 0.873
RST-B	931,763	38.75 / 0.986	32.82 / 0.973
RST-S	3,162,211	**39.81 / 0.988**	33.87 / 0.978
uGLR	323,410	37.88 / 0.982	33.60 / 0.977
uGTV	323,435	39.11 / 0.986	**34.01 / 0.979**

Table 2. Demosaicking performance (PSNR) under Gaussian noise (trained on noiseless data).

METHOD	$\sigma = 10$	$\sigma = 20$	$\sigma = 30$	$\sigma = 50$
RST-B	28.01	22.70	19.34	15.03
uGLR	28.24	22.84	19.49	15.20
uGTV	**28.31**	**22.89**	**19.56**	**15.38**

1.4.4. Robustness to Covariate Shift

The authors also assess the robustness of their models under covariate shift by adding Gaussian noise to the inputs during testing. Table 2 shows the demosaicking PSNR values for different noise levels (σ). uGTV consistently outperforms RST-B, especially as noise increases.

These results confirm that both uGLR and uGTV are more robust to noise compared to RST-B, with uGTV showing the best stability across all tested noise levels.

1.4.5. Discussion

The experimental results confirm the effectiveness of the proposed method. Despite having significantly fewer parameters, uGLR and uGTV outperform or match the performance of more complex models like RSTCANet and MAIN. Additionally, they demonstrate better robustness under covariate shifts.

1.5. Reproduction and Additional Experiments

In this section, we describe the experiments we performed to reproduce the key results of the paper.

1.5.1. Reproduction of a Key Experiment

Objective: We aimed to evaluate the upscaling performance of the proposed uGTV model on the Div2K dataset. The task consists in reconstructing a high-resolution 64×64 image from its downscaled 32×32 version — effectively retaining only one pixel out of four.

Methodology: As the authors did not release any official code, we reimplemented the entire architecture in PyTorch. Both the Conjugate Gradient (CG) and ADMM procedures were coded from scratch. While we initially considered using the PyTorch Geometric (PyG) library for graph operations, we ultimately relied solely on native PyTorch functions, as the original method is described primarily using matrix operations. Our experiments were run on A100 GPUs with 40 GB VRAM, available via Télécom Paris and Google Colab.

Model Parameters: Following the paper, we constructed a graph with $N = 64 \times 64 = 4096$ nodes and a 5×5 local neighborhood, leading to 24 edges per node. Each ADMM block contained 5 unrolled layers, with each layer performing 10 CG steps. Graph learning was performed using a shallow 4-layer CNN extracting 48 features per pixel, split into 4 sets of 12 — inspired by attention heads. More technical details are provided in Appendix B.

Results: We were unable to fully reproduce the original results due to memory constraints. The model relies on intermediate tensors indexed over the number of edges M, which—although smaller than N^2—still scales to over 10^5 elements. This led to memory overflows on all tested GPUs when using batched dense matrices.

Discussion: The paper efficiently leverages sparsity to mitigate such issues, but PyTorch's sparse tensor support remains limited, especially for batched operations. While dedicated sparse libraries exist, they are often incompatible with batched 3D tensors. Our decision to keep a batched design increased implementation complexity. Despite these limitations, we successfully adapted the model to run on reduced settings, as described in the following section.

1.5.2. Additional Experiment

Objective: To test the model under constrained settings, we trained a reduced version of the uGTV model on synthetic image data sampled from a normal distribution, shaped as $32 \times 32 \times 3$ tensors.

Methodology: We used the same architecture as in the previous experiment, but with significantly reduced parameters to meet GPU memory constraints. Details are provided in Appendix B. Despite its smaller size, the model still pushed the memory limits of the A100 GPU.

Results: The model successfully trained, reaching a reasonable loss. Visual inspection confirmed that the reconstructed output was coherent and consistent with the underlying distribution of the synthetic inputs.

Discussion: Training remained challenging. During backpropagation, tensors that were sparse in the forward pass often became dense, leading to memory overflows. In some cases, PyTorch attempted to allocate over five times the available GPU memory. Conversely, forcing more sparsity resulted in unsupported backward operations. This highlights the limitations of current PyTorch sparse support and suggests that, while the method is promising, the surrounding ecosystem still lacks maturity.

Further details on these experiments are provided in the Appendix B. The full codebase is available at:

`https://github.com/XwaeK/ML_with_Graphs_Project.git`

1.6. Critical Analysis

The proposed model successfully combines interpretability, computational efficiency, and strong empirical results, achieving state-of-the-art performance with significantly fewer parameters.

However, its scalability to very large graphs remains uncertain, and it has so far only been applied to image-based signal interpolation and demosaicking tasks; future work could explore its extension to more diverse graph learning problems such as node classification or link prediction.

Section 2: Cross-Paper Reflection and Methodological Contrast

Inpainting-Driven Graph Learning via Explainable Neural Networks
Subbareddy Batreddy, Pushkal Mishra, Yaswanth Kakarla, Aditya Siripuram — IEEE SPL, 2025.

2.1. Relevance to the Primary Study

This paper (Batreddy et al., 2025) builds upon the unrolling-based paradigm introduced in our main reference (Do et al., 2024), extending it to time-varying signals. It introduces a neural model that jointly infers both the graph structure and missing data, trained end-to-end through a closed-loop inpainting process.

Notably, it cites our main paper as a state-of-the-art baseline. Both methods couple signal reconstruction and graph learning, but this second work targets temporal data and adopts a fully unsupervised learning scheme.

2.2. Methodological Contributions

The authors propose a trainable unrolled architecture alternating between: 1. *Signal interpolation:* Filling in missing values of temporal graph signals. 2. *Graph update:* Adjusting the topology based on the updated signal.

Each component is differentiable, enabling *end-to-end training*. A feedback loop ensures that signal recovery quality directly informs graph adaptation.

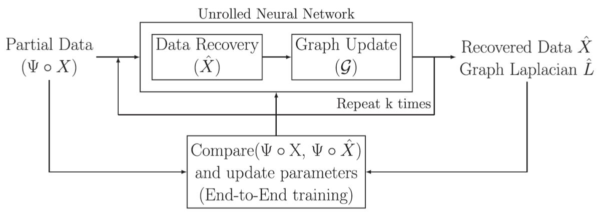


Figure 2. Unrolled model with a closed-loop feedback system. (Batreddy et al., 2025).

2.2.1. A Closed-Loop Framework

Unlike open-loop methods where the graph is fixed prior to reconstruction, this model iteratively refines the graph through reconstruction feedback. Fig. 2 illustrates this synergy between tasks.

2.2.2. Unsupervised Learning for Time-Varying Data

The model operates without ground-truth graphs. Training is supervised solely by the inpainting loss on observed entries, making it especially suitable for domains like neuroimaging or environmental monitoring.

2.2.3. Empirical Results

On real and synthetic datasets, the model outperforms two-stage baselines, both in reconstructing the signal and in recovering graphs closer to the ground truth. This supports its dual effectiveness and interpretability.

2.3. Connections and Complementarities with the Main Paper

2.3.1. Shared Design Principles

Both papers use unrolling to build interpretable, optimization-inspired neural architectures. Signal estimation and graph refinement alternate at each layer, embedding prior knowledge (e.g., smoothness) into learning. This joint training improves reconstruction and ensures that graph updates are guided by recovery performance.

2.3.2. Contrasts in Scope and Implementation

The key distinction lies in data modality and training paradigm. The main paper targets spatially structured image signals, using fixed graph priors (e.g., Mahalanobis distances) and allowing supervised training with full ground truth. In contrast, the second paper focuses on time-varying signals with unknown topology, requiring an unsupervised framework from partial observations.

Architecturally, the main model unrolls ADMM iterations with explicit regularization, while the second relies on flexible neural unrolling. Graph structure is jointly learned with signal recovery, guided solely by reconstruction performance.

2.3.3. Complementary Contributions

Together, these works highlight the versatility and promise of unrolling-based graph learning. The first focuses on interpretable and lightweight modeling in image restoration tasks, while the second tackles temporal graph data with a fully unsupervised, task-driven training approach.

Their complementary designs contribute to advancing the state of the art, showing how task-adaptive graph inference can benefit diverse GSP problems—from static image interpolation to dynamic time-series reconstruction.

References

Batreddy, S., Mishra, P., Kakarla, Y., and Siripuram, A. Inpainting-driven graph learning via explainable neural networks. *IEEE Signal Processing Letters*, 32:111–115, 2025. doi: 10.1109/LSP.2024.3501273.

Cheung, G., Leus, G., and Ortega, A. Graph spectral image processing. In *Proceedings of the IEEE*, volume 106, pp. 907–930, 2018.

Do, T. T., Eftekhar, P., Hosseini, S. A., Cheung, G., and Chou, P. A. Interpretable lightweight transformer via unrolling of learned graph smoothness priors. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS 2024)*, Vancouver, Canada, 2024. PMLR.

Eastman Kodak. Kodak lossless true color image suite (photocd pcd0992), 1993. http://r0k.us/graphics/kodak.

Hu, W., Gao, X., Cheung, G., and Guo, Z. Feature graph learning for 3d point cloud denoising. *IEEE Transactions on Signal Processing*, 68:2841–2856, 2020.

Huang, J.-B., Singh, A., and Ahuja, N. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5197–5206, 2015.

Ji, J., Zhong, B., and Ma, K.-K. Image interpolation using multi-scale attention-aware inception network. *IEEE Transactions on Image Processing*, 29:9413–9428, 2020.

Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., and Timofte, R. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 1833–1844, 2021.

Ortega, A., Frossard, P., Kovacevic, J., Moura, J. M., and Vandergheynst, P. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA: Carnegie Mellon University, 1994.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

Wang, S. and Shroff, N. A new alternating direction method for linear programming. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

Xing, M., Liu, Y., Ye, P., Zhu, L., Yang, Q., and Wang, C. Residual spatial-transformer channel attention network for image demosaicking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1852–1861, 2022.

Zhang, L., Wu, X., Buades, A., and Li, X. Color demosaicking by local directional interpolation and nonlocal adaptive thresholding. *Journal of Electronic Imaging*, 20 (2):023016–023016, 2011.

A. Unrolled Optimization Framework from the Main GTV-based Model

Algorithm 1 Unrolling of GTV-based signal interpolation algorithm.

```
**Input:** Initial signal observations $y$, initial graph $C_1$
**Parameters:** Number of unrolled layers $T$, ADMM iterations $K$, operator $H(\cdot)$
**Learnable:** Mahalanobis matrix $M$, ADMM parameters, $H$
**Initialization:** $y_1 \leftarrow y$                                # First signal input
**for** $t = 1$ **to** $T$ **do**
    Initialize $x_t^{(0)}$, $q_t^{(0)}$, $\mu_t^{(0)}$
    **for** $k = 1$ **to** $K$ **do**
        $(x_t^{(k)}, q_t^{(k)}, \mu_t^{(k)}) \leftarrow$ ADMM_Layer$(C_t, y_t, q_t^{(k-1)}, \mu_t^{(k-1)})$
    **end for**
    $x_t \leftarrow x_t^{(K)}$                                        # Final signal at layer $t$
    **if** $t < T$ **then**
        $C_{t+1} \leftarrow$ GraphLearn$(x_t; M)$                      # Learnable graph from $x_t$
        $y_{t+1} \leftarrow H(x_t)$                                    # Feature transform for next iteration
    **end if**
**end for**
**Compute loss:** $\mathcal{L}(x_T, x_{gt})$                          # Compare with ground truth
**Backpropagate gradients through all layers to update parameters**
```

B. Details of key and additional experiments

Overview: Each ADMM operation was controlled by its own parameter γ, and had a sequence of CG steps with learned parameters α and β. Our implementation, and the implementation of the paper, used 10 CG steps in each ADMM layer and 5 ADMM layers per ADMM block. Inside each Graph Learning block, a 4 layer CNN was used to extract 48 features per pixel, which were split into 4 sets of 12 features. Each set of features was fed into a Graph Learn layer, which was further fed into an ADMM block. This duplication and splitting of features was inspired by the concept of attention heads, and is shown visually in 3.

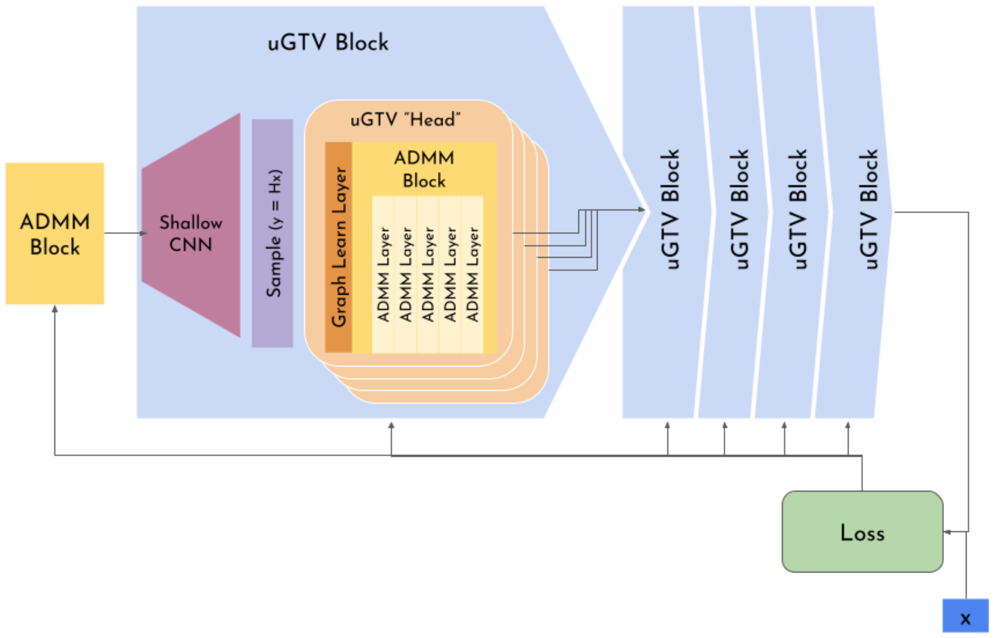
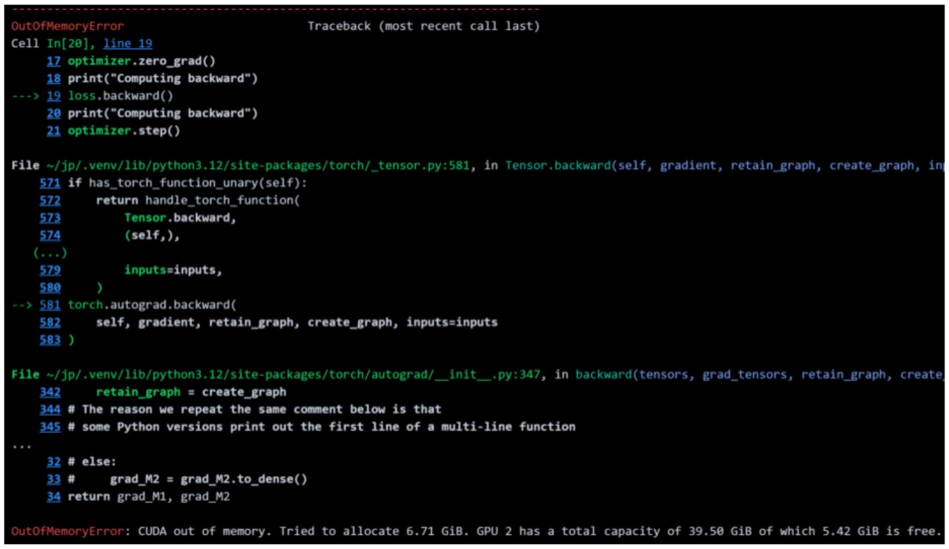


Figure 3. A brief overview of our network architecture

Key Experiment



```
---------------------------------------------------------------------------
OutOfMemoryError                          Traceback (most recent call last)
Cell In[20], line 19
     17 optimizer.zero_grad()
     18 print("Computing backward")
---> 19 loss.backward()
     20 print("Computing backward")
     21 optimizer.step()

File ~/jp/.venv/lib/python3.12/site-packages/torch/_tensor.py:581, in Tensor.backward(self, gradient, retain_graph, create_graph, in
    571 if has_torch_function_unary(self):
    572     return handle_torch_function(
    573         Tensor.backward,
    574         (self,),
   (...)
    579         inputs=inputs,
    580     )
--> 581 torch.autograd.backward(
    582     self, gradient, retain_graph, create_graph, inputs=inputs
    583 )

File ~/jp/.venv/lib/python3.12/site-packages/torch/autograd/__init__.py:347, in backward(tensors, grad_tensors, retain_graph, create_
    342     retain_graph = create_graph
    344 # The reason we repeat the same comment below is that
    345 # some Python versions print out the first line of a multi-line function
...
     32 # else:
     33 #     grad_M2 = grad_M2.to_dense()
     34 return grad_M1, grad_M2

OutOfMemoryError: CUDA out of memory. Tried to allocate 6.71 GiB. GPU 2 has a total capacity of 39.50 GiB of which 5.42 GiB is free.
```

Figure 4. CUDA Out-of-Memory error encountered during training on A100 GPU.

Additional Experiment For the low resolution version of the experiment, we chose significantly reduced task and parameters:

1. **Task**: Upscale $32 \times 32 \times 3$
2. **CG Steps**: 3
3. **ADMM Layers**: 3
4. **uGTV Blocks**: 1
5. **uGTV Heads**: 2 *(rather than 4)*
6. **CNN Features per Head**: 4 *(rather than 12*