Alpine: Flexible, User-Friendly, and Distributed PyTorch Library for Implicit Neural Representation Development

Kushal Vyas[1], Vishwanath Saragadam[2], Ashok Veeraraghavan[1], Guha Balakrishnan[1]
[1] Electrical and Computer Engineering, Rice University,
[2]Electrical and Computer Engineering, University of California, Riverside.
[1]{kvyas, vashok, guha}@rice.edu [2]vishwanath.saragadam@rice.edu

Abstract

Implicit neural representations (INRs) are the workhorse data structure in neural field algorithms, offering a flexible, continuous, and compact encoding of complex signals. While simple in concept, INR designs now vary along various axes, such as nonlinearities, parameter initialization schemes, training procedures, and interpretability techniques. As such, there is a growing need for a systematic library to ensure rapid, scalable, and reproducible INR development. To fill this need, we present Alpine, an open-source PyTorch library for flexible development, fitting, and function visualization for INRs, with a focus on rapid prototyping and ease of extensibility across a variety of scientific applications, from applied physics to medical imaging. Alpine provides a clean API to set up custom INR workflows, train them using gradient-based or sophisticated meta-learners, and visualize learned features, learned INR geometry, and metrics. This paper presents the components of Alpine, and its capabilities[1].

1. Introduction

Implicit neural representations (INRs) are powerful learned function approximators for signal data that are the workhorse of neural fields algorithms. An INR $F_\theta : \mathbb{R}^d \mapsto \mathbb{R}^D$ maps *coordinates* lying in a d-dimensional space to a value in a D-dimensional output space, with parameters θ, offering a continuous and potentially compact alternative to traditional array-based discrete signal representations. Due to their elegant and general formulation, they have been successfully applied in a variety of domains including image and video representation [6, 7, 23, 24, 26, 39, 40, 50, 52], shape representation [18, 19], signed distance fields (SDFs) [33], neural radiance fields [30, 47], physics models [22, 35, 55], material rendering [25], computational imaging [3, 9], medical imaging [42, 46, 51], linear inverse problems [8, 45], virtual reality [11] and compression [13, 27, 44, 56]. Despite the conceptual simplicity of INRs, there are many design decisions that continue to be explored and discovered that have significant impact on their performance. These include choices of nonlinearities [39, 40], parameter initializations [14, 50], training methodologies [17, 40], and of late, interpretability methods [20, 32]. Navigating through these methods for a specific application requires practical expertise, which is often infeasible for a scientist or engineer outside of the core disciplines developing INRs, such as computer graphics and vision. Furthermore, existing INR implementations do not follow standardized templates and often involve complex codebases, making it challenging to reproduce and extend INR workflows across different disciplines or data.

There is therefore a pressing need for a standardized library that provides an easy and convenient way to prototype, deploy, and test existing INRs on a variety of problems. Furthermore, as scientific signals grow in size, e.g., long video sequences and multispectral satellite imagery, this library should gracefully scale up computation across multiple GPUs and perform fast data I/O. And parallel to core INR functionality, this library should also provide visualization methods, such as XINC [32] and SplineCam [20, 21] that can provide users interpretability, if they require it, into the function encoded by a given INR.

This paper presents *Alpine*, a new open-source PyTorch [34] library that addresses all of these needs. It's key characteristics are:

- **Extensible data interfaces:** *Alpine* offers a variety of I/O functions and data-loaders for efficient handling of $d-$dimensional coordinates and signals from various domains.
- **Rapid INR prototyping:** *Alpine* is designed with modularity at its core, providing low level INR building blocks such as nonlinearities and layers which can be used interchangeably as well as high level functionality that allows to quickly prototype INRs. Furthermore, we offer base classes which can be inherited to seamlessly integrate

[1]Alpine public repository: https://github.com/kushalvyas/alpine

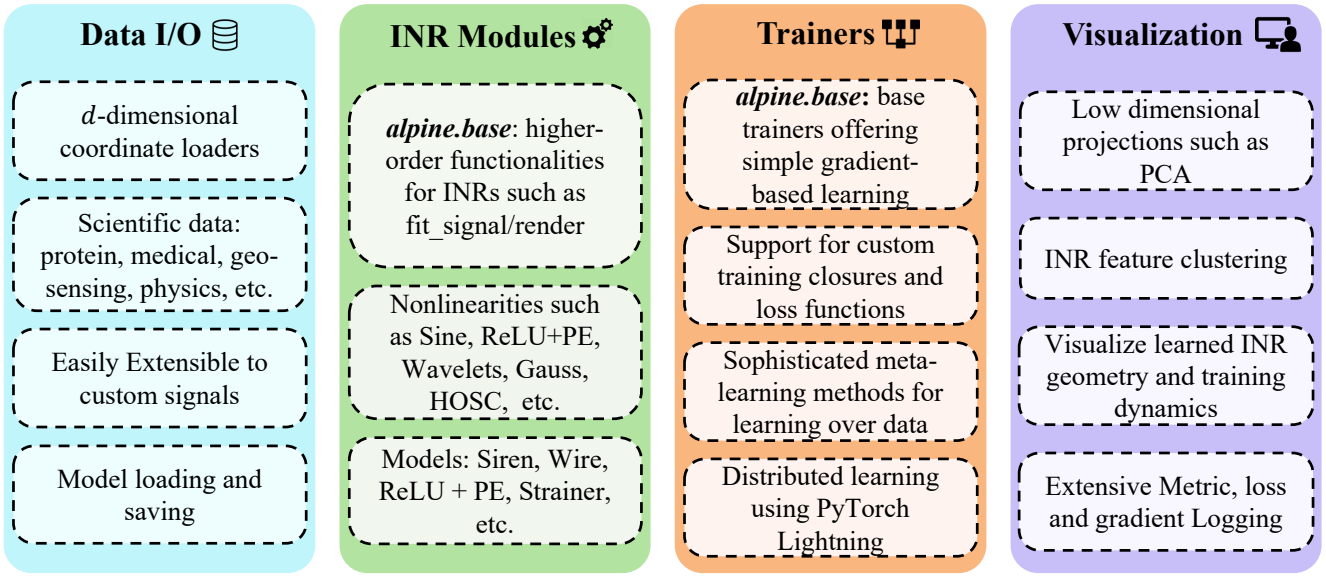

Figure 1. ***Alpine*, a library for all your INR needs.** *Alpine* provides ready interfaces for data loading, INR modules, training routines, and visualization. *Alpine* is modular, easily extensible, and comes with a rich set of examples, allowing users across disciplines immediate plug-and-play access to INR functionality and extensibility.

custom models and activation functions into the *Alpine* workflow.

- **Reconfigurable INR training procedures:** *Alpine* uses a gradient-based `fit_signal` method which can be easily reconfigured using closures to allow custom forward propagation and loss functions.
- **Powerful under-the-hood visualizations:** *Alpine* offers visualization tools for tracking various metrics such as PSNR and SSIM while training INRs, low-dimensional projections of high-dimensional INR features and clustering methods for analyzing trends in the fit signal. *Alpine* also includes tools to study the learned geometry and spline partitions of INR models INRs [20, 21].
- **Plug-n-Play integration with PyTorch ecosystem:** Based on PyTorch, *Alpine* permits plug-n-play INRs with other PyTorch pipelines allowing users to freely take advantage of the myriad offerings of the PyTorch ecosystem such as PyTorch-Lightning [15], TorchMetrics [12], and TorchGeo [43].

2. *Alpine* Modules

In this section we describe each module in *Alpine* and their key functionalities. Fig. 1 presents an overview.

alpine.dataloaders

`alpine.dataloaders` have the simple functionality to directly load $d-$dimensional coordinates and D-dimensional signal values in a scalable way, with options to load in random batches suitable for available GPU memory. Taking advantage of PyTorch's dimension broadcasting, we provide vectorized coordinate signal data pairs as well as $d-$dimensional *spatial* coordinate signal data pairs. We further provide support for loading atypical file formats such as NIFTI [1] for neuroimaging, RCSB PDB formats [4] for protein structures, loading categorical land cover data [54] used in remote sensing etc.

alpine.models

`alpine.models` houses all INR building blocks such as layers, Fourier encodings, and nonlinearities. We make a conscious decision to decouple nonlinearities and INR layer assembly to promote an object-oriented workflow and interchangeability of building blocks. For all objects, we follow PyTorch's convention of defining model layers in the constructor, and the `forward` function implementing the forward pass. We additionally provide a new abstract method `forward_w_features` for each model which returns intermediate layer outputs which may be useful for downstream visualization tasks such as clustering and low-dimensional projection.

Each `alpine.model` inherits the `alpine.base` class which holds `fit_signal` and `render` methods. To further promote reconfigurable and custom training processes (further discussed in Sec. 2,) we enforce all outputs from an `alpine` model be a dictionary object holding outputs and auxiliary data (e.g., layer-wise features). Each INR model comes with default parameter and hyperparameter initialization strategies. We also offer data-driven initialization techniques such as meta-learning (MAML) [17, 48] and Strainer [50].

wire_ct.py

```python
import alpine
import torch
from torchmetrics.image import PeakSignalNoiseRatio as PSNR

# Instantiate a model
wire = alpine.models.Wire(in_features=2, out_features=1,
                          hidden_features=256, hidden_layers = 5,
                          omegas = [10.0], sigmas = [10.0]).cuda()
wire.register_loss_function(MSE_TV_Loss())

coord_signal_dataloader = alpine.dataloaders.BatchedNDSigalLoader(...)

# Fit Wire for inverse CT reconstruction
_ = wire.fit_signal(coord_signal_dataloader,
                    closure = inverse_ct_closure
                    metrics={'psnr': PSNR()})

# Render
outputs = wire.render(...points ...)
```

custom_loss.py

```python
class MSE_TV_Loss(nn.Module):
  def __init__(self, alpha=0.9, beta=0.1):
    super(MSE_TV_Loss, self).__init__()
    self.alpha = alpha
    self.beta = beta
    self.mse = nn.MSELoss()
    self.tv = TVLoss()

  def forward(self, x, y):
    x_sinogram = x['output']
    x_img = x['output_img']
    y_sinogram = y['signal']

    mse = self.alpha * self.mse(x_sinogram, y_sinogram)
    tv = self.beta * self.tv(x_img)

    return mse + tv
```

```python
def inverse_ct_closure(model_ctx, input, signal, iteration):
    output_packet = model_ctx(input)
    output_img = output_packet['output']
    output_sinogram = radon(output_img.permute(0, 3, 1, 2), thetas)[None,...]
    return {'output' : output_sinogram, 'output_img':output_img}
```

Figure 2. ***Alpine* makes INR workflows straightforward.** We illustrate a sample *Alpine* workflow for sparse view CT reconstruction using Wire [39], shown in red. *Alpine* makes it effortless to introduce a custom closure (shown in yellow) to further process the reconstructed CT before computing loss. The custom loss function(shown in green) can also be integrated with the fitting process seamlessly.

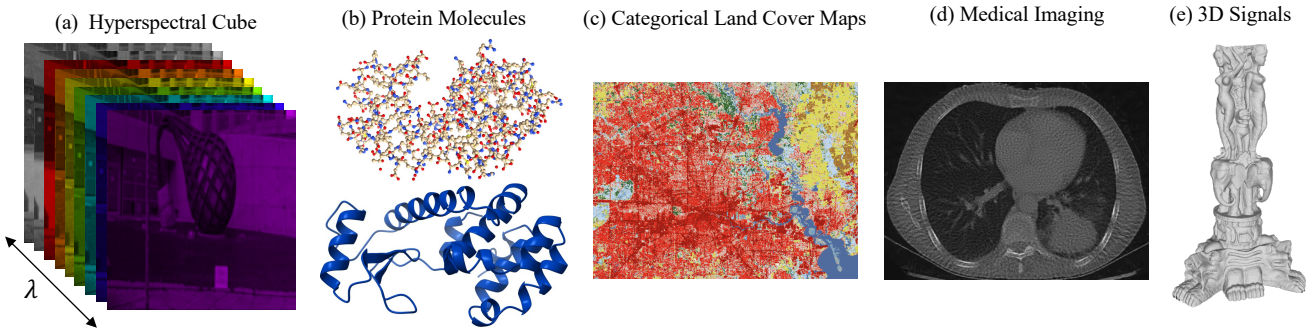

Figure 3. ***Alpine* can easily model signals of various types, across many scientific tasks.** We illustrate various signals fit using *Alpine*: (a) hyperspectral cube [2], (b) protein molecule from RCSB PDB [4], (c) land cover map from NLCD [54], (d) medical CT scan [10], and (e) a 3D shape [37]. In all cases, loading, fitting, and rendering the signals required just a few lines of code.

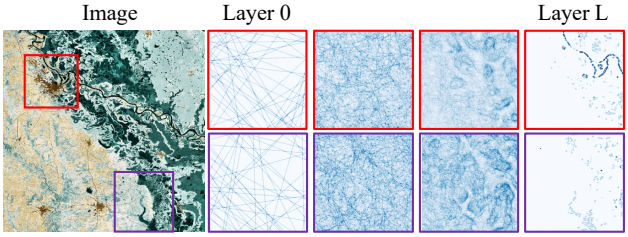

Figure 4. ***Alpine* provides a lens for visualizing training dynamics for INRs.** Using local complexity measure proposed by Humayun et.al. [21], we visualize learned INR partition geometry on the input domain.

alpine.trainers

INRs are generally trained using iterative gradient-based updates to fit a given signal or measurement. *Alpine*'s base class (which is inherited by the INR model as described above) offers in-built and reconfigurable training and rendering functionality exposed through the `fit_signal` and `render` methods. `fit_signal` supports coordinate and signal inputs as tensors, or as a torch or `alpine` dataloader. Notably, while the `fit_signal` method provides a standard iterative training procedure for fitting the INR, it also accepts an optional callable *closure* argument which implements custom forward or inverse problem routines. We discuss *Alpine*'s training workflow as shown in Fig. 2 in detail in Sec. 2.2. *Alpine* trainers also offer sophisticated

training routines such as using MAML-based meta learning for training generalizable INRs over a dataset and also training distributedly across multiple GPUs using PyTorch-Lightining as discussed further in Section 2.1

We also include progress bars in `alpine`'s trainers that display key variables such as loss, reconstruction quality, and estimated time remaining for user-friendly, informative training process.

alpine.vis

Further qualitative and theoretical understanding of learned INR characteristics is of increasing importance to the research community [20, 21, 32]. To address this, we provide a rich toolkit with convenient visualization functions, from tracking reconstruction metrics such as PSNR, to visualizing learned INR features. As mentioned previously in Sec. 2, `alpine` models retain layer-wise features, which are typically high-dimensional. *Alpine* offers low-dimensional projections of these feature vectors using PCA and k-means clustering. We also provide computation of the Local Complexity measure [21] proposed by Humayun et.al., to visualize underlying partition geometry of the learned INR as shown in Figure 4.

2.1. Extending to the PyTorch ecosystem

We intentionally build Alpine in PyTorch to take advantage of PyTorch's wide ecosystem across various disciplines such as optics (torchoptics [16]), climate science (ClimateLearn [31], TorchGeo [43]), medical imaging (Monai [5]), drug discovery (TorchDrug [57]), also leverage high performance computing (Lightning [15], PyTorch-elastic [34]) and extending to modular runtime configurations using Hydra [53].

First, we integrate TorchMetrics [12] as the default metric logger for *Alpine*. The `fit_signal` methods accepts optional *torchmetric* objects and can automatically compute and track metrics throughout training iterations using its inbuilt *update* and *increment* functions. For extending the same workflow to any custom metric or a collection of metrics, we simply wrap those objects as a torchmetrics object.

Furthermore, fitting INRs to large signals requires training across multiple GPUs. To this end, we integrate PyTorch-Lightning into *Alpine* to support distributed machine learning and show an example fitting of 3D Thai statue in Section 2.2. Crucially, we use Lightning's DDP strategy to enable distributed INR learning. Furthermore, we introduce a lightning trainer in `alpine.trainers` module as a wrapper class that automatically promotes an *Alpine* INR model into a PyTorch-Lightning model.

2.2. Examples

Figure 2 shows an illustration of *Alpine*'s compact interface to instantiate, fit, and visualize INRs. Figure 2 *(red)*, presents a straightforward way to instantiate a baseline Wire model [39] from `alpine.models` and to fit a sinogram signal using a custom *closure* implementing a radon projection operator, explained in Figure 2 *(yellow)*. We also showcase *Alpine*'s modular ability to seamlessly integrate a custom loss function, shown in Figure 2 *(green)*, for fitting the CT sinogram.

We further present a diverse set of examples obtained using *Alpine*, demonstrating the package's adaptability to various scientific and computational disciplines in Figure 3. The figure includes fitting a hyperspectral data cube [2], protein macromolecule from PDB [4] (final molecular structure rendered using Chimera-X [29]), categorical land cover map [54] for geo-spatial modeling, CT scan from The Cancer Genome Atlas [10], and 3D Thai statue [37]. The statue was fit using *Alpine*'s distributed training functionality. To highlight a glimpse of our visualization toolkit, in Figure 4, we show the learned partition geometry of a Siren [40] INR trained to fit soil moisture patches [49] using the local complexity measure [21] providing a more theoretical insight into INR features.

3. Conclusion and Next Steps

The conceptual simplicity and compactness of INRs makes them an attractive data structure for scientists and engineers for various computational and scientific applications. INR usage is expanding to various signal types beyond normal RGB images, such as gigapixel imagery [28, 38], videos [6, 7], and physical data [36, 41]. Furthermore, INR methods related to architectures, activation functions, and visualization techniques, are also expanding as researchers understand more about their properties. Our goal with *Alpine* is to facilitate INR development and integration to general sciences and engineering, beyond just computer vision and computer graphics, with a gentle learning curve. To enable easy adoption, we ship *Alpine* with detailed documentation and a diverse set of demo examples.

Alpine currently offers basic functionality for all the modules described in Sec. 2 and Fig. 1. We will work on expanding *Alpine* with other researchers in the field along several axes. These include expanding data I/O for more scientific data types, continually integrating state-of-the-art INR developments as they arise, and scaling *Alpine* for high-performance computing (HPC) applications.

References

[1] NIfTI; Neuroimaging Informatics Technology Initiative — nifti.nimh.nih.gov. https://nifti.nimh.nih.gov/. [Accessed 25-04-2025]. 2

[2] Boaz Arad and Ohad Ben-Shahar. Sparse recovery of hyperspectral signal from natural rgb images. In *European Conference on Computer Vision*, pages 19–34. Springer, 2016. 3, 4

[3] Benjamin Attal, Eliot Laidlaw, Aaron Gokaslan, Changil Kim, Christian Richardt, James Tompkin, and Matthew O'Toole. Törf: Time-of-flight radiance fields for dynamic scene view synthesis. *Advances in neural information processing systems*, 34:26289–26301, 2021. 1

[4] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000. 2, 3, 4

[5] M Jorge Cardoso, Wenqi Li, Richard Brown, Nic Ma, Eric Kerfoot, Yiheng Wang, Benjamin Murrey, Andriy Myronenko, Can Zhao, Dong Yang, et al. Monai: An open-source framework for deep learning in healthcare. *arXiv preprint arXiv:2211.02701*, 2022. 4

[6] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser Nam Lim, and Abhinav Shrivastava. Nerv: Neural representations for videos. *Advances in Neural Information Processing Systems*, 34:21557–21568, 2021. 1, 4

[7] Hao Chen, Matthew Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. HNeRV: Neural representations for videos. In *CVPR*, 2023. 1, 4

[8] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *CVPR*, 2021. 1

[9] Chiun-Hong Chien and Jake K Aggarwal. Volume/surface octrees for the representation of three-dimensional objects. *Computer Vision, Graphics, and Image Processing*, 36(1): 100–113, 1986. 1

[10] Kenneth Clark, Bruce Vendt, Kirk Smith, John Freymann, Justin Kirby, Paul Koppel, Stephen Moore, Stanley Phillips, David Maffitt, Michael Pringle, Lawrence Tarbox, and Fred Prior. The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository. *Journal of Digital Imaging*, 26(6):1045–1057, 2013. 3, 4

[11] Nianchen Deng, Zhenyi He, Jiannan Ye, Budmonde Duinkharjav, Praneeth Chakravarthula, Xubo Yang, and Qi Sun. Fov-nerf: Foveated neural radiance fields for virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, 28(11):3854–3864, 2022. 1

[12] Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh Jha, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. Torchmetrics - measuring reproducibility in pytorch. *Journal of Open Source Software*, 7(70):4101, 2022. 2, 4

[13] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 1

[14] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo J. Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you should treat it like one. *CoRR*, abs/2201.12204, 2022. 1

[15] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. 2, 4

[16] Matthew J. Filipovich and A. I. Lvovsky. Torchoptics: An open-source python library for differentiable fourier optics simulations, 2024. 4

[17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 1126–1135. JMLR.org, 2017. 1, 2

[18] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7154–7164, 2019. 1

[19] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4857–4866, 2020. 1

[20] Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard G. Baraniuk. Splinecam: Exact visualization and characterization of deep network geometry and decision boundaries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3789–3798, 2023. 1, 2, 4

[21] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Deep networks always grok and here is why, 2024. 1, 2, 3, 4

[22] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. 1

[23] Alper Kayabasi, Anil Kumar Vadathya, Guha Balakrishnan, and Vishwanath Saragadam. Bias for action: Video implicit neural representations with bias modulation. *arXiv preprint arXiv:2501.09277*, 2025. 1

[24] Subin Kim, Sihyun Yu, Jaeho Lee, and Jinwoo Shin. Scalable neural video representations with learnable positional features. In *Advances in Neural Information Processing Systems*, pages 12718–12731. Curran Associates, Inc., 2022. 1

[25] Alexandr Kuznetsov. Neumip: Multi-resolution neural materials. *ACM Transactions on Graphics (TOG)*, 40(4), 2021. 1

[26] Zhen Liu, Hao Zhu, Qi Zhang, Jingde Fu, Weibing Deng, Zhan Ma, Yanwen Guo, and Xun Cao. Finer: Flexible spectral-bias tuning in implicit neural representation by variable-periodic activation functions, 2023. 1

[27] Shishira R Maiya, Sharath Girish, Max Ehrlich, Hanyu Wang, Kwot Sin Lee, Patrick Poirson, Pengxiang Wu, Chen Wang, and Abhinav Shrivastava. Nirvana: Neural implicit representations of videos with adaptive networks and autoregressive patch-wise modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14378–14387, 2023. 1

[28] Julien N.P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: Adaptive coordinate networks for neural representation. 2021. 4

[29] Elaine C. Meng, Thomas D. Goddard, Eric F. Pettersen, Greg S. Couch, Zach J. Pearson, John H. Morris, and Thomas E. Ferrin. Ucsf chimerax: Tools for structure building and analysis. *Protein Science*, 32(11):e4792, 2023. 4

[30] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[31] Tung Nguyen, Jason Jewik, Hritik Bansal, Prakhar Sharma, and Aditya Grover. Climatelearn: Benchmarking machine learning for weather and climate modeling. *arXiv preprint arXiv:2307.01909*, 2023. 4

[32] Namitha Padmanabhan, Matthew Gwilliam, Pulkit Kumar, Shishira R Maiya, Max Ehrlich, and Abhinav Shrivastava. Explaining the implicit neural canvas: Connecting pixels to neurons by tracing their contributions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10957–10967, 2024. 1, 4

[33] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019. 1

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 1, 4

[35] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. 1

[36] Nathan Ranno and Dong Si. Neural representations of cryo-em maps and a graph-based interpretation, 2022. 4

[37] Stanford 3D Scans Repository. Thai statue. 3, 4

[38] Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard Baraniuk, and Ashok Veeraraghavan. Miner: Multiscale implicit neural representations. In *European Conf. Computer Vision*, 2022. 4

[39] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18507–18516, 2023. 1, 3, 4

[40] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 1, 4

[41] Luke Thomas Smith, Tom Horrocks, Naveed Akhtar, Eun-Jung Holden, and Daniel Wedge. Implicit neural representation for potential field geophysics. *Scientific Reports*, 15(1): 9799, 2025. 4

[42] Bowen Song, Liyue Shen, and Lei Xing. Piner: Prior-informed implicit neural representation learning for test-time adaptation in sparse-view ct reconstruction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1928–1938, 2023. 1

[43] Adam J. Stewart, Caleb Robinson, Isaac A. Corley, Anthony Ortiz, Juan M. Lavista Ferres, and Arindam Banerjee. TorchGeo: Deep learning with geospatial data. *ACM Transactions on Spatial Algorithms and Systems*, 2024. 2, 4

[44] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. In *European Conference on Computer Vision*, pages 74–91. Springer, 2022. 1

[45] Yu Sun, Jiaming Liu, Mingyang Xie, Brendt Wohlberg, and Ulugbek S Kamilov. Coil: Coordinate-based internal learning for tomographic imaging. *IEEE Transactions on Computational Imaging*, 7:1400–1412, 2021. 1

[46] Yiran Sun, Tucker Netherton, Laurence Court, Ashok Veeraraghavan, and Guha Balakrishnan. Ct reconstruction from few planar x-rays with application towards low-resource radiotherapy. In *Deep Generative Models*, pages 225–234, Cham, 2024. Springer Nature Switzerland. 1

[47] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020. 1

[48] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855, 2021. 2

[49] Noemi Vergopolan, Nathaniel W Chaney, Ming Pan, Justin Sheffield, Hylke E Beck, Craig R Ferguson, Laura Torres-Rojas, Sara Sadri, and Eric F Wood. Smap-hydroblocks, a 30-m satellite-based soil moisture dataset for the conterminous us. *Scientific data*, 8(1):264, 2021. 4

[50] Kushal Vyas, Ahmed Imtiaz Humayun, Aniket Dashpute, Richard G. Baraniuk, Ashok Veeraraghavan, and Guha Balakrishnan. Learning transferable features for implicit neural representations. In *Advances in Neural Information Processing Systems*, pages 42268–42291. Curran Associates, Inc., 2024. 1, 2

[51] Yuehao Wang, Yonghao Long, Siu Hin Fan, and Qi Dou. Neural rendering for stereo 3d reconstruction of deformable tissues in robotic surgery. In *Intl. Conf. Medical Image Computing and Computer-Assisted Intervention*, 2022. 1

[52] Shaowen Xie, Hao Zhu, Zhen Liu, Qi Zhang, You Zhou, Xun Cao, and Zhan Ma. Diner: Disorder-invariant implicit neural representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. 1

[53] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. 4

[54] Limin Yang, Suming Jin, Patrick Danielson, Collin Homer, Leila Gass, Stacie M. Bender, Adam Case, Catherine Costello, Jon Dewitz, Joyce Fry, Michelle Funk, Brian Granneman, Greg C. Liknes, Matthew Rigge, and George Xian. A new generation of the united states national land cover database: Requirements, research priorities, design, and implementation strategies. *ISPRS Journal of Pho-*

togrammetry and Remote Sensing, 146:108–123, 2018. 2, 3, 4

[55] Lingchen Yang, Byungsoo Kim, Gaspard Zoss, Baran Gözcü, Markus Gross, and Barbara Solenthaler. Implicit neural representation for physics-driven actuated soft bodies. *ACM Transactions on Graphics*, 41(4):1–10, 2022. 1

[56] Yunfan Zhang, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. Implicit neural video compression. In *ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022. 1

[57] Zhaocheng Zhu, Chence Shi, Zuobai Zhang, Shengchao Liu, Minghao Xu, Xinyu Yuan, Yangtian Zhang, Junkun Chen, Huiyu Cai, Jiarui Lu, Chang Ma, Runcheng Liu, Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Torchdrug: A powerful and flexible machine learning platform for drug discovery, 2022. 4