F-INR: Functional Tensor Decomposition for Implicit Neural Representations

Sai Karthikeya Vemuri   Tim Büchner   Joachim Denzler

Computer Vision Group

Friedrich Schiller University Jena, Germany

{sai.karthikeya.vemuri, tim.buechner, joachim.denzler}@uni-jena.de

Abstract

Implicit Neural Representation (INR) has emerged as a powerful tool for encoding discrete signals into continuous, differentiable functions using neural networks. However, these models often have an unfortunate reliance on monolithic architectures to represent high-dimensional data, leading to prohibitive computational costs as dimensionality grows. We propose F-INR, a framework that reformulates INR learning through functional tensor decomposition, breaking down high-dimensional tasks into lightweight, axis-specific sub-networks. Each sub-network learns a low-dimensional data component (e.g., spatial or temporal). Then, we combine these components via tensor operations, reducing forward pass complexity while improving accuracy through specialized learning. F-INR is modular and, therefore, architecture-agnostic, compatible with MLPs, SIREN, WIRE, or other state-of-the-art INR architecture. It is also decomposition-agnostic, supporting CP, TT, and Tucker modes with user-defined rank for speed-accuracy control. In our experiments, F-INR trains $100\times$ faster than existing approaches on video tasks while achieving higher fidelity ($+3.4$ dB PSNR). Similar gains hold for image compression, physics simulations, and 3D geometry reconstruction. Through this, F-INR offers a new scalable, flexible solution for high-dimensional signal modeling.

1. Introduction

Implicit Neural Representations (INRs) are continuous, functional representations of discrete signals such as images [60, 61, 76], videos [4, 7, 17, 80], 3D scenes [5, 22, 42, 43], and geometries [35, 47]. Implemented via neural networks, these methods map discrete structured data into a continuous function space, facilitating smooth interpolation. This general nature promotes progress in architectural design and practical applications [42, 47, 53, 57].

Continuous parametrization offers several advantages over discrete, grid-based representations. These include higher memory efficiency, the ability to be defined over an
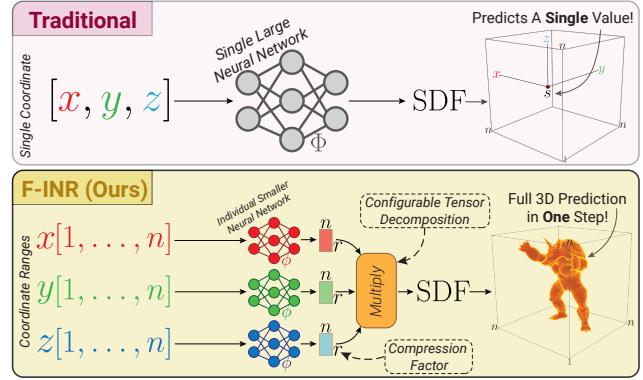

Figure 1. **Efficient INRs via Functional Tensor Decomposition:** INR models use a single large network to predict one value (batch of values) at a time. Our approach decomposes the function into smaller networks, enabling full prediction in a single step with configurable tensor decomposition modes and compression ranks.

unbounded domain, and resolution invariance [13]. Specifically, this approach captures fine-grained details, where resolution is determined by the network's capacity and expressiveness rather than the grid [57]. Additionally, the differentiability of these representations plays a key role in computing gradients and higher-order derivatives using automatic differentiation, relevant for inverse modeling [55, 56, 67].

Another form for representing multi-dimensional data is tensor decomposition, commonly used in signal and image processing and analysis [2, 3, 29, 30, 58, 63, 70, 79]. It models high-dimensional signals as combinations of low-rank, low-dimensional components. Yet, these are confined to discrete grid settings, limiting their applicability.

Both INR and tensor decomposition have their respective advantages, and a combination of these could efficiently represent complex data. Thus, we propose **F-INR**, an INR reformulation that leverages the strengths of tensor decompositions. F-INR uses dedicated univariate neural networks to learn a variably separated form of INR, retaining the benefits of continuous representations by facilitating low-dimensional components. A general illustration

of this setup is provided in Figure 1. In this study, we focus on combining neural networks with three tensor decomposition techniques, specifically CP [20], TT [45], and Tucker [64]. We establish versatility across various INR experiments, including image and video encoding, shape representation via SDFs, and physics simulation encoding for super-resolution. **F-INR** outperforms classical INRs in training speed (up to $100\times$), task-specific metrics, and qualitative results. This indicates that improvements can be achieved beyond network architectural modifications.

To summarize, our main contributions are fourfold: 1) A novel reformulation of INRs through functional tensor decompositions, providing a new perspective on continuous signal representation. 2) The F-INR framework, which utilizes three specific decomposition modes combined with existing network architectures, offers flexible and efficient modeling. 3) An empirical demonstration of F-INRs strong performance in key INR applications. 4) Open-sourcing our framework to promote the research area of F-INR[1].

2. Related Work

Implicit Neural Representation (INR) evolved through several stages. First ideas introduced explicit coordinate mechanisms like positional encoding [42] and random Fourier feature mappings [62] to improve the MLP representation capacity. Subsequent works focused on activation functions and model architectures: SIREN introduced sinusoidal activations [57], WIRE the Gabor wavelet activations [53], and InstantNGP the hash-based encoding [43]. Notably, these strongly enhanced the INR expressiveness. Recent work is application driven, e.g., data compression [13, 17, 61], computer vision [1, 4, 9, 15, 47], graphics [35, 42, 44, 55, 56, 77], and robotics [6, 33, 54].

Tensor decomposition represents high-dimensional data as compositions of smaller factors. Classical decompositions forms split data into mode-wise components [20, 29, 30, 45, 64]. Early works applied fixed functional bases (e.g., Gaussian, Fourier, or Chebyshev expansions) to represent each factor in a continuous domain [18, 27, 76], but these have limited expressiveness. Recent approaches use neural networks as learnable functional bases, combining deep learning with tensor factorization to overcome this. For example, some methods replace hand-crafted components (Tucker factors or PCA/SVD vectors) with neural networks [8, 10, 24, 52]. In physics-informed learning, tensor decomposition was used to solve PDEs by splitting the solution into lower-dimensional neural components. This yielded faster training and higher accuracy [11, 26, 66, 68]. However, to our knowledge, no prior work unifies INRs with functional tensor decomposition in a general way.

Tensor decompositions for INR recently emerged as a combination of low-rank tensor factorization with INR-specific applications, such as NeRFs [42]. Works like TensoRF factorized the radiance fields into compact low-rank components [5, 22]. This resulted in fast and memory-efficient view synthesis, an effective but domain-specific solution. Moreover, MLPs are not used to learn the components of tensor decomposition directly but only for feature decoding. Similarly, CoordX employed split MLPs for each coordinate dimension and low-rank representation [34]. They were fused in the deeper layers, omitting different decomposition modes. Using MLPs to represent low-rank tensor functions (instead of fixed bases) was proposed in works like [38, 39, 69]. They represent multidimensional data continuously, achieving state-of-the-art image in-painting and point cloud up-sampling [38, 39, 69]. **F-INR** unifies and generalizes prior methods into a single, modular paradigm. Akin to CoordX [34], separated MLPs handle dedicated input dimensions, leveraging smaller subnetworks for efficiency. It incorporates the low-rank structure found in LRTFR [22] or TensoRF [5], reducing redundancy while retaining expressiveness [11, 66]. Unlike these approaches, F-INR is unconstrained to a specific tensor factorization or application domain. It inherently supports several decomposition modes and ranks adaptable to the task data structure. As F-INR is backend-agnostic, it benefits from advances in architectures such as SIREN [57] or Fourier features [62] and makes structured INR representations more modular, scalable, efficient, and versatile for many tasks.

3. Functional Tensor Decomposition for INRs

Real-world signals, such as images or videos, must be stored as a discrete grid of values. These discretized signals, which are d-dimensional and c-variate, face limitations in resolution and memory. Neural networks offer a solution by modeling a continuous version of these signals, known as Implicit Neural Representations (INR), which are both resolution-independent and memory-efficient. Thus, an INR task is a function $\Phi : \mathbb{R}^d \mapsto \mathbb{R}^c$, here estimated by a neural network $\Phi_\theta : \mathbb{R}^d \mapsto \mathbb{R}^c$, mapping a coordinate vector $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ to a c-variate output signal.

We instead propose to reformulate the problem approximation as a tensor product of d smaller neural networks:

$$\Phi(\mathbf{x}) \approx \bigotimes_{i=1}^{d} \phi_i(x_i; \theta_i), \qquad (1)$$

where $\phi_i(\cdot)$ denotes a univariate neural network for the i-th dimension with learnable parameters θ_i. Each network produces a tensor of a particular *rank* to restore the original signal via classical tensor decomposition *modes* [20, 45, 64], denoted as $\otimes$. The decomposition tensors have continuous functions as basis [68], and the approach is considered func-

---

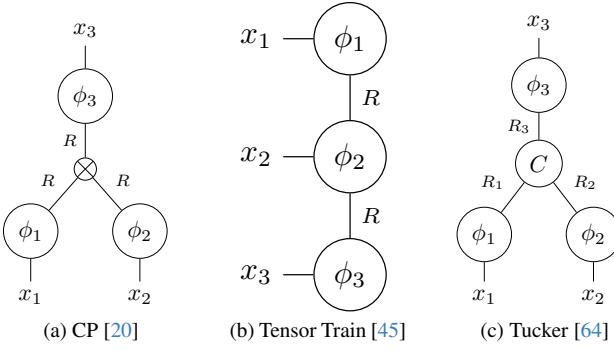(a) CP [20]   (b) Tensor Train [45]   (c) Tucker [64]

Figure 2. **Tensor diagrams for three decompositions (a)-(c):** This schematic [29, 48] describes each circle as a component, and spokes determine its dimension. The spokes connectivity shows how the decomposition is carried out and how to obtain the original tensor. In F-INR, each component is learned by an individual neural network. More visualizations and mathematical formulations are provided in the supplementary.

tional tensor decomposition. Hence, our reformulation introduces the idea of functional-INRs (F-INR).

Notably, this approach is model-agnostic, and any state-of-the-art INR architecture can be a *backend*, inheriting its advantages implicitly [42, 53, 57, 62]. Two more hyperparameters comprise our approach: *mode* and *rank*. *Mode* refers to the type of the tensor decomposition used. In this study, we consider three established *modes* in tensor theory:

1. **Canonic-Polyadic Decomposition (CP) [20]** involves decomposing the d-dimensional tensor into d-factor matrices of *rank R*, visualized in the Figure 2a.
2. **Tensor-Train Decomposition (TT) [45]** involves a train of d-connected chains (trains) of low-*rank* tensors, shown in Figure 2b.
3. **Tucker Decomposition [64]** like CP, uses factor matrices to decompose a tensor. Still, it also includes a smaller core tensor C that captures the interactions between the components of each mode, shown in Figure 2c.

The next selectable parameter is *rank*, which specifies the decomposition's rank and determines the expressiveness and complexity a model can learn. This combination of *backends*, *modes*, and *ranks* offers precise control over complexity, compression, and performance concerning the specific application requirements.

3.1. Advantages

F-INR mitigates the curse of dimensionality by employing an effective separation of variables approach, which enables the efficient representation of high-dimensional functions [16, 21, 78]. Recent works have demonstrated that multiple neural networks connected through tensor decomposition forms are universal approximations [11, 26, 68]. Even if an exact, separable solution does not exist, a sufficiently large rank can approximate the solution, leveraging

the universal approximation power of neural networks [68]. Previous studies [16, 21, 49] have shown that the number of parameters required for function approximation grows exponentially with the dimensionality, hindering neural networks' ability to learn high-dimensional functions.

This is mitigated in F-INR, which adopts a divide-and-conquer approach, where each neural network learns a low-dimensional function, aggregating these functions to reconstruct the full high-dimensional function. Another benefit of our method is that a case trained on points along a grid (N^d) requires only $N \cdot d$ data points during the forward pass. This leads to significant speedups, as evident in the complexity of the forward pass for each mode; for the example of $d = 3$, see Table 1.

Decomposition	Complexity	Description
-	$\mathcal{O}(m^2 l n^3)$	Single neural network with n^3 inputs
CP [20]	$\mathcal{O}(m^2 l n r + n^2 r^2)$	Three networks for $n \times r$ factor matrices
Tensor-Train [45]	$\mathcal{O}(m^2 l n r^2 + n^2 r^2)$	Two $n \times r$ factors, one $r \times n \times r$ core
Tucker [64]	$\mathcal{O}(m^2 l n r + r n^3)$	Three $n \times r$ factors, one $r \times r \times r$ core

Table 1. **Forward pass complexity:** We assume a grid of $n \times n \times n$, i.e., n^3 data points, and tabulate the computations for a neural network (m features, l layers). Note that $r \ll m^2 l$ and the sequence of the operations are given in the supplementary.

4. Experiments

We evaluate the performance of functional tensor decomposition (FTD) methods in enhancing computational efficiency and accuracy for implicit neural representation (INR) problems in diverse applications. We investigate the impact of key parameters on F-INR in our experiments to demonstrate the generalizability of our approach:

1. **Backend**: The neural network architecture learning the decomposed components.
2. **Mode**: The specific tensor decomposition technique.
3. **Rank**: The rank of the decomposed tensor components.

We select a standard ReLU-based MLP [41], with and without Positional Encoding (PE) [42]. Further, we use SIREN [57], which utilizes periodic activations with specialized initialization, and WIRE [53], which incorporates a custom Gabor wavelet activation. These backends are recognized as the best-performing approaches across a wide range of INR applications.

The test three tensor decomposition modes concerning their applicability to different INR problem tasks: Canonical-Polyadic (CP) [20], Tensor-Train (TT) [45], and

3

Figure 3. **F-INR for Image encoding**: The visualizations of Image encoding are shown here, along with PSNR (dB) values in the inset. All the first-row images do not use any decompositions, and the second row is the F-INRs for images for different backends and ranks. Combinations with the highest PSNR are presented here. Additional results are in the supplementary.

Tucker [64]. Across all our experiments, we assess the performance of different combination modes and backends at various ranks. Evaluation metrics include domain-specific measures such as the Peak Signal To Noise ratio (PSNR) for encoded images and videos with uncompressed original recordings. L_2 error between ground truth and prediction for physics simulations and Intersection-over-Union (IoU) for geometry learning using signed distance functions (SDFs). Additionally, we provide the training times for each experiment.

We compare results to baseline versions of the same backend without tensor decomposition, using consistent hyperparameters to ensure that any observed improvements are due to the functional tensor decompositions. We use a standard MLP architecture with three layers, each with 256 features, and train for 50,000 iterations using Adam [28] for ten independent runs unless otherwise specified.

We present key results in this section, with additional ablation studies across different ranks, modes, and backends available in the supplementary material. Training times were measured on an NVIDIA GTX 1080, demonstrating that F-INRs achieve fast and computationally efficient formulations even on modest hardware and do not need to rely on adapted and customized hardware kernels [43].

4.1. Image and Video Encoding

We begin by demonstrating the application of F-INR to encode an image. Images are second-order tensors; a simple

matrix decomposition mode suffices. Instead of utilizing a single neural network to represent the entire image, we train two univariate neural networks, each responsible for one spatial dimension, to learn smaller patches. Their matrix product reconstructs the original image. In a conventional INR setup, an image is represented as:

$$\Phi_\theta(x, y) = (r, g, b), \qquad (2)$$

where Φ_θ denotes the network, (x, y) are pixel coordinates, and (r, g, b) represent the pixel color values. By contrast, we learn the image as:

$$\phi_1(x; \theta_1) \otimes \phi_2(y; \theta_2) = (r, g, b), \qquad (3)$$

where $\times$ operator denotes matrix multiplication. Here, x and y are image coordinates; the output is the pixel color.

We choose a publicly available image of a cat featuring intricate details such as a finely patterned scarf and whiskers. The encoded images and their respective PSNR (in dB) values demonstrate that F-INR-based representations achieve superior PSNR for identical backend architectures while offering a 100x speedup in computational efficiency. Ground truth and encoded images are depicted in Figure 3. Detailed numerical results can be found in Table 2. We also include LIIF [76], which encodes an image using neighbor information of pixels as a baseline. Also, DeepTensor [52] is similar to using a ReLU backend. Further, we also employed hash-based encoding [43] as an additional backend. Please note that we did not rely on custom

Figure 4. **Encoding video with nuanced facial features [80] (publicly available) using F-INR**: The mean PSNR (dB) and model are shown in the first column; training time is in the last. SIREN [57] and ReLU [41] with positional encoding [42] outperform their baseline, capturing facial details and maintaining temporal consistency, while WIRE [53] performs worse. The best performance (fifth row) was achieved using a combination of hash [43] and positional encoding for spatial and frame dimensions, respectively, highlighting the modularity of F-INR. Additional results are in the supplementary.

	Backend	Rank	PSNR (dB) (↑)	SSIM (↑)	Time (s)
INR	ReLU	-	25.72 ± 0.31	0.73 ± 0.01	5066
	ReLU+PE [42]	-	31.53 ± 0.19	0.82 ± 0.01	5920
	ReLU+Hash [43]	-	33.88 ± 0.14	0.92 ± 0.01	2100
	WIRE [53]	-	32.09 ± 0.21	0.86 ± 0.01	5290
	SIREN [57]	-	30.90 ± 0.19	0.78 ± 0.00	5280
	LIIF [76]	-	33.96 ± 0.14	0.91 ± 0.03	6147
	DeepTensor [52]	-	26.92 ± 0.31	0.78 ± 0.01	122
F-INR	WIRE	128	32.63 ± 0.38	0.88 ± 0.00	61
	WIRE	256	33.94 ± 0.38	0.89 ± 0.00	86
	WIRE	316	34.19 ± 0.12	0.90 ± 0.00	101
	ReLU+PE	128	33.90 ± 0.28	0.91 ± 0.00	82
	ReLU+PE	256	34.27 ± 0.37	0.92 ± 0.00	108
	ReLU+PE	316	34.55 ± 0.31	0.93 ± 0.01	123
	ReLU+Hash	256	35.61 ± 0.14	0.93 ± 0.01	76
	ReLU+Hash	**316**	$\mathbf{35.91 \pm 0.11}$	$\mathbf{0.93 \pm 0.01}$	**85**

Table 2. **Results for Image encoding using F-INR**: We show *PSNR* and *SSIM* values for F-INRs image encoding. We observe that F-INR models train faster and yield better results.

CUDA kernels for hash encoding [43], ensuring fair comparison in run-times, but the implementation details follow the original algorithm [43]. We apply our image encoding framework to tasks such as single image super-resolution and image denoising. This demonstrates how F-INRs are influenced by the strong architectural properties of backends and how we improve upon their baseline implementations, as detailed in the supplementary material.

We extend our experiments to video encoding, following the previous approach but with a temporal component. Specifically, the representation is:

$$\phi_1(x; \theta_1) \otimes \phi_2(y; \theta_2) \otimes \phi_3(t; \theta_3) = (r, g, b), \quad (4)$$

where t denotes the temporal dimension (frame index). We employ all specified tensor decomposition modes for this task. We use a 256×256 resolution video of a person [80], comprising 300 frames with varying facial expressions and head movements visualized in Figure 4. The modular nature of F-INR allows for flexible integration of different encoding strategies and network backends, optimizing task performance. Unlike a single monolithic neural network, which is constrained to a fixed encoding or architecture, F-INR enables a plug-and-play approach, where
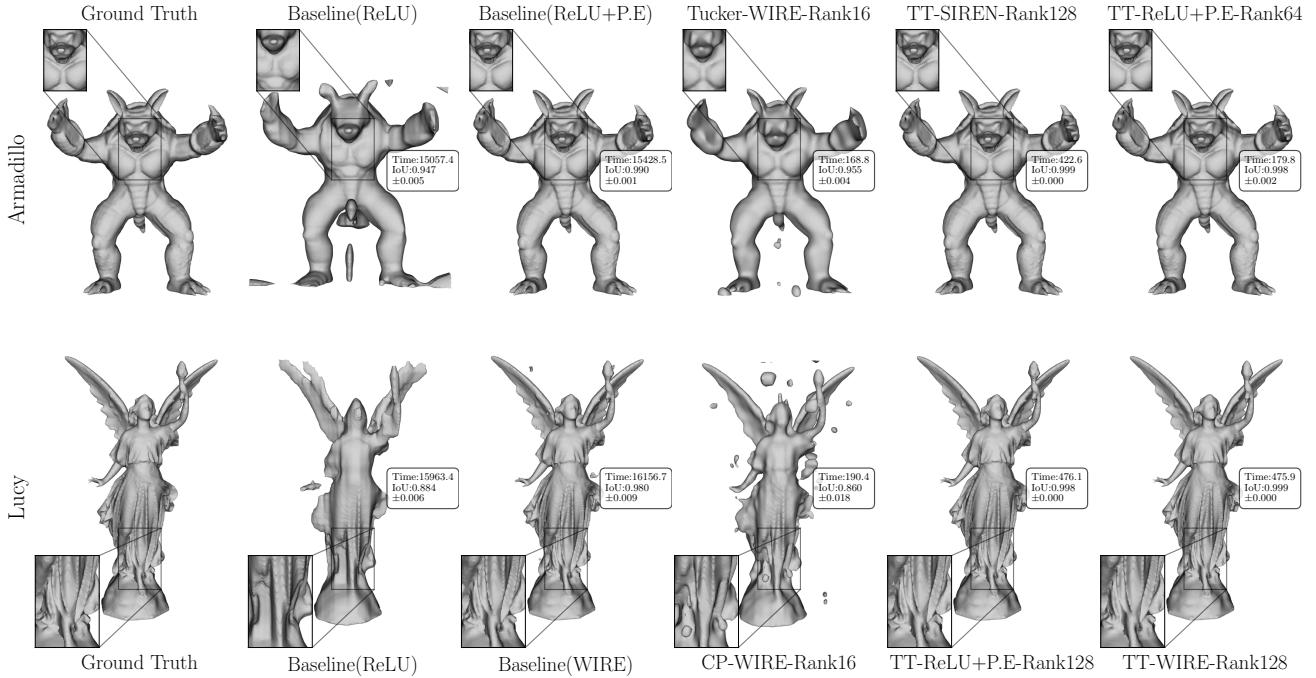
Figure 5. We compare the performance of our method with two 3D scan reconstructions from [12, 14, 31, 65]. The time taken for training and mean IoU are reported in the inset box. All models were obtained from SDF using the marching cubes algorithm with Laplacian smoothing to reduce artifacts [37]. We compare our results to the ground truth SDF, the worst and best-performing baseline methods, and the best-decomposed version. Notably, a rank that is too low ($r = 16$) does not yield a successful reconstruction.

different encodings can be tailored to specific dimensions. This adaptability is exemplified here. The best performance was achieved using hash encoding for spatial networks ϕ_1, ϕ_2 and Fourier features for the temporal network ϕ_3, leveraging the strengths of each encoding for their respective domains. We do not include existing neural network-based video compression algorithms like NeRVs or COIN++ [7, 13, 40]. They relate to frame-wise learning of videos and subsequent neural network quantization. Quantitative results are provided in the supplementary.

4.2. Signed Distance Functions for Geometries

A Signed Distance Function (SDF) is a continuous function that assigns a value to every point in 3D space relative to the nearest surface point. While learning SDFs from point clouds and solving the Eikonal PDE has seen significant success [47, 53, 57], the use of voxel grids for SDF learning remains a relatively underexplored area. Voxel grids offer a structured and dense representation, making them particularly well-suited for applications that require spatial coherence and efficient 3D convolutions [36, 74, 75]

We leverage F-INR to solve the Eikonal PDE and learn geometric representations directly from voxel grid-based SDF data, building on the foundational formulations of [15,

46, 47, 57]. Our loss function is defined as:

$$
\begin{aligned}
\mathcal{L}_{\text{SDF}} = \int_{\Omega} &\|\triangledown \Psi (x, y, z, \Theta) - 1\| \, dx dy dz \qquad (5) \\
+ \int_{\Omega} &\left\| \Psi (x, y, z, \Theta) - \hat{\Psi}(x, y, z) \right\| dx dy dz \\
+ \int_{\Omega < \Omega_0} &\left\| \Psi (x, y, z, \Theta) - \hat{\Psi}(x, y, z) \right\| dx dy dz,
\end{aligned}
$$

where Ω represents the spatial domain in which the SDF Ψ is learned. The loss function consists of three terms. The first term enforces the Eikonal PDE constraint. The second term minimizes the discrepancy between the predicted SDF values Ψ and ground truth values $\hat{\Psi}$ across the domain Ω. The third term prioritizes data points near the surface within a specified threshold $\Omega < \Omega_0$. Following [59], we truncate SDF values beyond a threshold of 0.1.

To evaluate the performance of F-INRs on geometries, we use models from the Stanford 3D Scan Repository [12, 14, 31, 65]. We select a subset of objects with intricate geometries and varying levels of detail, providing a robust testbed for assessing the accuracy and efficiency of our SDF-based approach. The experiments utilize the same underlying model architectures, baseline configurations, and decomposition ranks described in the broader experimental section. We did not use hash encoding for this task as it
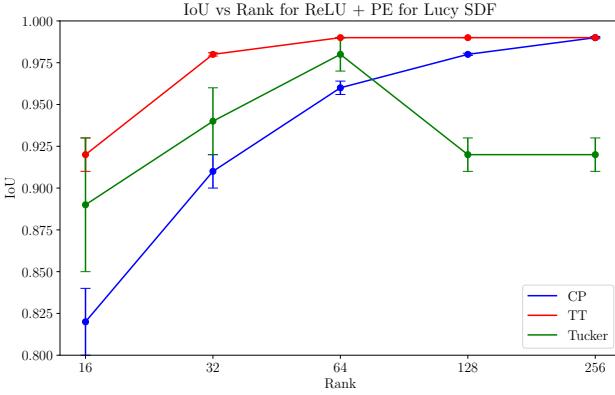
Figure 6. IoU vs. Rank for all the modes of tensor decomposition for the backend ReLU + Positional Encoding of learning SDF of Lucy. Tensor-Train outperforms the other two modes, while Tucker mode degrades performance for larger ranks. In contrast, TT mode exhibits stable performance with increasing rank

lacks global differentiability, making it unsuitable for applications requiring well-behaved gradients [23, 43].

As visualized in Figure 5, F-INR produces more detailed and accurate representations of objects. The results are obtained using the same marching cube algorithm settings for all learned SDF voxel grids [37]. An overview of the raw learned SDF results and further comparison with other methods like DeepSDF [47] and IGR [15] are given in the supplementary material. To compare different modes, we varied the ranks and modes for Lucy SDF, using the best-performing backend, as shown in Figure 6. The results show that Tensor-Train outperforms both in stability and accuracy metrics, attributed to its connecting structure [45]. Interestingly, Tucker mode worsens with increasing rank, attributed to its large core, which forms a bottleneck as explained in [29, 45, 70]. We also observe that the effect of rank is significant for TT mode, with performance improving up to rank 64 and remaining constant after that. Additional results for all the backends, modes, and geometries are in the supplementary.

4.3. Super Resolution of Simulations

High-fidelity simulations are computationally intensive and often limited by the resolution of traditional numerical methods. While Physics-Informed Neural Networks (PINNs) [50, 68] are a promising alternative, they, too, face scalability challenges when increasing the number of collocation points or handling complex simulations [32, 71, 72]. Therefore, a trade-off between two approaches is helpful: Coarse simulations are more accessible for numerical methods, and INRs can learn a resolution-free form using PINN loss. Our method, F-INR, is trained using low-resolution data and a physics-informed loss function. We validate this
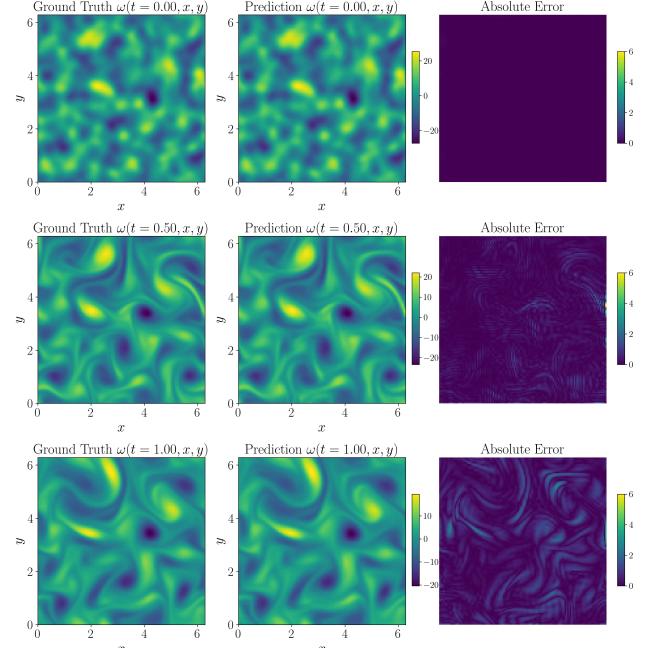

Figure 7. Visualizations of vorticity for F-INR- TT WIRE Rank 128 at time steps 0.0 (start), 0.5 (middle), and 1.0 (end). The first column is the ground truth vorticity, and the second is the predicted high-fidelity vorticity trained from coarse data and underlying Navier-Stokes PDE. The last column is the absolute pointwise error between ground truth and prediction. We see high alignment between the ground truth and predictions without employing time marching schemes [32].

approach on a decaying turbulent flow simulation using a dataset with fine details of vortices dissipating. We evaluate performance based on the L_2 error between the true high-resolution ground truth and our predicted high-resolution outputs. The partial differential equation (PDE) for the simulation is the vorticity form of the incompressible Navier-Stokes [73] equations, expressed as:

$$\begin{aligned} \partial_t \omega + u \cdot \nabla \omega &= \nu \Delta \omega, & x \in \Omega, t \in \Gamma, \\ \nabla u &= 0, & x \in \Omega, t \in \Gamma, \\ \omega(x, 0) &= \omega_0(x), & x \in \Omega, \end{aligned} \quad (6)$$

where u denotes the velocity field, $\omega = \nabla \times u$ is the vorticity, ω_0 represents the initial vorticity, and $\nu = 0.01$ is the viscosity coefficient. The spatial domain is $\Omega \in [0, 2\pi]^2$, and the temporal domain is $\Gamma \in [0, 1]$.

The original dataset has a resolution of $101 \times 128 \times 128$. We train our F-INR models using a lower resolution of $10 \times 64 \times 64$ corresponding to approximately 40-fold sparsity. Our loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{Coarse}} + \mathcal{L}_{\text{Phy}} . \quad (7)$$

Where $\mathcal{L}_{\text{Coarse}}$ corresponds to the loss calculated on the coarse simulation data, and $\mathcal{L}_{\text{Phy}}$ is the physics-informed

Backend	Mode	Rank	L_2 Error	Train (hh:mm)
ReLU+PE [42]	-	-	0.097 ± 0.009	20:30
WIRE [53]	-	-	0.073 ± 0.004	20:25
SIREN [57]	-	-	0.184 ± 0.010	20:24
ModifiedPINN [72]	-	-	0.074 ± 0.008	28:40
CausalPINN [73]	-	-	0.070 ± 0.011	33:12
MFF Net [25]	-	-	0.048 ± 0.003	35:18
PhySR [51]	-	-	0.038 ± 0.020	27:05
WIRE	TT	64	0.034 ± 0.004	00:59
WIRE	TT	128	0.036 ± 0.002	01:08
WIRE	TT	256	0.033 ± 0.002	01:40
WIRE	Tucker	128	0.037 ± 0.003	01:28
WIRE	Tucker	256	0.035 ± 0.004	02:01
ReLU+PE	TT	64	0.033 ± 0.001	00:51
ReLU+PE	**TT**	**128**	$\mathbf{0.030 \pm 0.002}$	**01:12**
ReLU+PE	**TT**	**256**	$\mathbf{0.030 \pm 0.002}$	**01:59**
ReLU+PE	Tucker	64	0.039 ± 0.005	00:45
ReLU+PE	Tucker	128	0.032 ± 0.004	01:34
ReLU+PE	Tucker	256	0.032 ± 0.005	02:03

Table 3. Comparison of L_2 Errors for F-INRs and baseline implementations for super-resolution of decaying vorticity simulation using Navier Stokes equation. We tabulate only top-performing settings with an average L_2 Error of less than 0.04 here, all outperforming the baseline implementations (remaining in the supplementary material). F-INRs consistently outperforms, having lesser L_2 error and convergence time.

loss term enforcing the PDE constraint from Equation (6). Notably, we do not require techniques such as time marching [32], which divides the domain into smaller temporal intervals as we learn the solution across the entire domain.

In addition to the sparse data, we sample 200 points per dimension for collocation to compute the physics-informed loss. The L_2 errors for our results are presented in Table 3. We compare our results against architectures such as Modified PINNs [72] and CausalPINNs [73], designed to enhance the original PINN formulation and can be directly applied for super-resolution task. We also compare our approach against MeshFreeFlowNet [25] and PhySR [51], architectures specifically developed for super-resolution of simulations. Our findings reveal that F-INR consistently delivers faster and better solutions. We also would like to emphasize that this work is the first to demonstrate the applicability of WIRE [53] in physics simulation scenarios. A representative solution is visualized in Figure 7, which compares predicted vorticity after training by F-INR with mode TT, backend WIRE, and Rank 128 with ground truth high fidelity data. Only selected best performing F-INR results are shown here. The ablations of various ranks, backends, and sparsity levels are provided in the supplementary.

5. Limitations and Future Directions

While F-INR offers several advantages, it is currently limited to structured data formats, making it challenging to apply to unstructured scenarios such as point clouds or ray-marching [42]. Unlike structured grids, unstructured data requires application-specific modifications, often increasing computational complexity [5, 38]. Adapting our framework in such settings would require mapping individual data points with additional tensor components, reducing forward pass efficiency. While promising, this direction demands further research to balance efficiency and representation quality, which we leave for the future.

Another interesting direction is exploring more expressive tensor decompositions, such as tensor rings or topology-aware tensor networks [29, 30] to enhance compression-accuracy trade-offs. Exploring adaptive rank selection techniques to enable automatic complexity control is also interesting. Additionally, integrating tensor decomposition with CUDA-based optimizations, as seen in InstantNGP [43], could further accelerate inference while preserving structured representation benefits.

Finally, our work remains largely empirical. While decomposed neural networks are effective universal approximators [11, 19, 26, 68], deeper theoretical insights are needed to understand better how tensor decomposition influences expressivity and efficiency in coordinate-based implicit neural representations.

6. Conclusions

We introduce Functional Tensor Decomposition-based Implicit Neural Representations (F-INR), a framework that unifies tensor decomposition with implicit neural representations for efficient high-dimensional function modeling. By leveraging univariate neural networks to learn low-dimensional components, F-INR mitigates the curse of dimensionality and accelerates training.

A key advantage of F-INR is its modularity. Unlike monolithic INRs, it seamlessly integrates various tensor decompositions (CP, TT, Tucker) and neural architectures (ReLU with positional encoding, hash encoding, SIREN, WIRE). This flexibility allows easy adaptation to new decomposition strategies and network backends.

Our experiments demonstrate significant improvements in training speed (up to $100\times$) and accuracy across tasks, including image and video encoding, PDE-based super-resolution, and SDF-based geometry encoding. While currently limited to structured data, extending F-INR to unstructured settings is a promising direction. This work lays the foundation for further advances in scalable high-dimensional function learning by bridging tensor decomposition with implicit neural representations.

References

[1] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[2] Woody Austin, Grey Ballard, and Tamara G. Kolda. Parallel tensor compression for large-scale scientific data. In *IPDPS'16: Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium*, pages 912–922, 2016. 1

[3] Grey Ballard, Alicia Klinvex, and Tamara G. Kolda. TuckerMPI: A parallel C++/MPI software package for large-scale data compression via the Tucker tensor decomposition. *ACM Transactions on Mathematical Software*, 46(2):13, 2020. 1

[4] Tim Büchner, Orlando Guntinas-Lichius, and Joachim Denzler. Improved obstructed facial feature reconstruction for emotion recognition with minimal change cyclegans. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 262–274. Springer, 2023. 1, 2

[5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 1, 2, 8

[6] Boyuan Chen, Robert Kwiatkowski, Carl Vondrick, and Hod Lipson. Full-body visual self-modeling of robot morphologies, 2021. 2

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

[8] Yurong Chen, Hui Zhang, Yaonan Wang, Yimin Yang, and Jonathan Wu. Flex-dld: Deep low-rank decomposition model with flexible priors for hyperspectral image denoising and restoration. *IEEE Transactions on Image Processing*, 33: 1211–1226, 2024. 2

[9] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2

[10] Jen-Tzung Chien and Yi-Ting Bao. Tensor-factorized neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):1998–2011, 2018. 2

[11] Junwoo Cho, Seungtae Nam, Hyunmo Yang, Seok-Bae Yun, Youngjoon Hong, and Eunbyung Park. Separable physics-informed neural networks. *Advances in Neural Information Processing Systems*, 2023. 2, 3, 8

[12] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 303–312. ACM, 1996. 6, 24, 25

[13] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goli'nski, Yee Whye Teh, and A. Doucet. Coin++: Neural compression across modalities. *Trans. Mach. Learn. Res.*, 2022, 2022. 1, 2, 6

[14] Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. Linear light source reflectometry. In *ACM SIGGRAPH 2003 Papers*, pages 749–758, New York, NY, USA, 2003. Association for Computing Machinery. 6

[15] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of the 37th International Conference on Machine Learning*. JMLR.org, 2020. 2, 6, 7, 24

[16] Lars Grüne. Overcoming the curse of dimensionality for approximating lyapunov functions with deep neural networks under a small-gain condition. *IFAC-PapersOnLine*, 54(9): 317–322, 2021. 3

[17] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Implicit neural representations with levels-of-experts. *Advances in Neural Information Processing Systems*, 35:2564–2576, 2022. 1, 2

[18] Behnam Hashemi and Lloyd N. Trefethen. Chebfun in three dimensions. *SIAM Journal on Scientific Computing*, 39(5): C341–C363, 2017. 2

[19] Indupama Herath. *Multivariate Regression using Neural Networks and Sums of Separable Functions*. PhD thesis, Ohio University, 2022. 8

[20] Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6 (1-4):164–189, 1927. 2, 3

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. 3

[22] Wenbo Hu, Yuling Wang, Lin Ma, Bangbang Yang, Lin Gao, Xiao Liu, and Yuewen Ma. Tri-miprf: Tri-mip representation for efficient anti-aliasing neural radiance fields. In *ICCV*, 2023. 1, 2

[23] Xinquan Huang and Tariq Alkhalifah. Efficient physics-informed neural networks using hash encoding. *Journal of Computational Physics*, 501:112760, 2024. 7

[24] Masaaki Imaizumi and Kohei Hayashi. Tensor decomposition with smoothness. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1597–1606. PMLR, 2017. 2

[25] Chiyu "Max" Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A. Tchelepi, Philip Marcus, Prabhat, and Anima Anandkumar. Meshfreeflownet: a physics-constrained deep continuous space-time super-resolution framework. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2020. 8, 21

[26] Pengzhan Jin, Shuai Meng, and Lu Lu. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022. 2, 3, 8

[27] Nikos Kargas and Nicholas D. Sidiropoulos. Supervised learning and canonical decomposition of multivariate functions. *IEEE Transactions on Signal Processing*, 69:1097–1107, 2021. 2

[28] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017. 4

[29] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. 1, 2, 3, 7, 8

[30] Tamara G. Kolda and David Hong. Stochastic gradients for large-scale tensor decomposition. *SIAM Journal on Mathematics of Data Science*, 2(4):1066–1095, 2020. 1, 2, 8

[31] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 313–324, New York, NY, USA, 1996. Association for Computing Machinery. 6

[32] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, 34:26548–26560, 2021. 7, 8

[33] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR, 2022. 2

[34] Ruofan Liang, Hongyi Sun, and Nandita Vijaykumar. Coordx: Accelerating implicit neural representation with a split mlp architecture. *ArXiv*, abs/2201.12425, 2022. 2

[35] Chen-Hsuan Lin, Chaoyang Wang, and Simon Lucey. Sdf-srn: Learning signed distance 3d object reconstruction from static images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 1, 2

[36] Feng Liu and Xiaoming Liu. Voxel-based 3d detection and reconstruction of multiple objects from a single image. *Advances in Neural Information Processing Systems*, 34:2413–2426, 2021. 6

[37] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery. 6, 7

[38] Yisi Luo, Xile Zhao, Zhemin Li, Michael K Ng, and Deyu Meng. Low-rank tensor function representation for multidimensional data recovery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. 2, 8

[39] Yisi Luo, Xile Zhao, and Deyu Meng. Revisiting nonlocal self-similarity from continuous representation, 2024. 2

[40] Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2020. 6

[41] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 3, 5

[42] 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, 2, 3, 5, 8

[43] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 1, 2, 4, 5, 7, 8

[44] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[45] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. 2, 3, 7

[46] 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 (CVPR)*, 2019. 6

[47] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, Long Beach, CA, USA, 2019. IEEE. 1, 2, 6, 7, 24

[48] Roger Penrose. Applications of negative dimensional tensors. *Combinatorial Mathematics and its Applications*, 1971. 3

[49] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14(5):503–519, 2017. 3

[50] 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. 7

[51] Pu Ren, Chengping Rao, Yang Liu, Zihan Ma, Qi Wang, Jian-Xun Wang, and Hao Sun. Physr: Physics-informed deep super-resolution for spatiotemporal data. *Journal of Computational Physics*, 492:112438, 2023. 8, 21

[52] Vishwanath Saragadam, Randall Balestriero, Ashok Veeraraghavan, and Richard Baraniuk. Deeptensor: Low-rank tensor decomposition with deep network priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46:10337–10348, 2022. 2, 4, 5

[53] 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, 2, 3, 5, 6, 8

[54] Anthony Simeonov, Yilun Du, Andrea Tagliasacchi, Joshua B Tenenbaum, Alberto Rodriguez, Pulkit Agrawal, and Vincent Sitzmann. Neural descriptor fields: Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400. IEEE, 2022. 2

[55] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 1, 2

[56] Vincent Sitzmann, Eric R. Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. In *Proc. NeurIPS*, 2020. 1, 2

[57] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit

neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020. 1, 2, 3, 5, 6, 8

[58] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jong-soo Park, Xing Liu, and George Karypis. FROSTT: The formidable repository of open sparse tensors and tools, 2017. 1

[59] Christiane Sommer, Lu Sang, David Schubert, and Daniel Cremers. Gradient-sdf: A semi-implicit surface representation for 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6280–6289, 2022. 6

[60] Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007. 1

[61] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. In *Computer Vision – ECCV 2022*, pages 74–91, Cham, 2022. Springer Nature Switzerland. 1, 2

[62] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020. 2, 3

[63] Keita Teranishi, Daniel M. Dunlavy, Jeremy M. Myers, and Richard F. Barrett. Sparten: Leveraging kokkos for on-node parallelism in a second-order method for fitting canonical polyadic tensor models to poisson data. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2020. 1

[64] Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 2, 3, 4

[65] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21th Annual Conference on Computer Graphics and Interactive Techniques, SIG-GRAPH 1994, Orlando, FL, USA, July 24-29, 1994*, pages 311–318. ACM, 1994. 6, 24

[66] Sai Karthikeya Vemuri and Joachim Denzler. Gradient statistics-based multi-objective optimization in physics-informed neural networks. *Sensors*, 23(21), 2023. 2

[67] Sai Karthikeya Vemuri, Tim Büchner, and Joachim Denzler. Estimating soil hydraulic parameters for unsaturated flow using physics-informed neural networks. In *Computational Science – ICCS 2024*, pages 338–351, Cham, 2024. Springer Nature Switzerland. 1

[68] Sai Karthikeya Vemuri, Tim Büchner, Julia Niebling, and Joachim Denzler. Functional tensor decompositions for physics-informed neural networks, 2024. 2, 3, 7, 8

[69] Jianli Wang and Xile Zhao. Functional transform-based low-rank tensor factorization for multi-dimensional data recovery. In *Computer Vision – ECCV 2024*, pages 39–56, Cham, 2025. Springer Nature Switzerland. 2

[70] Maolin Wang, Yu Pan, Zenglin Xu, Xiangli Yang, Guangxi Li, and Andrzej Cichocki. Tensor networks meet neural networks: A survey and future perspectives. *CoRR*, abs/2302.09019, 2023. 1, 7

[71] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021. 7

[72] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022. 7, 8, 21

[73] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421:116813, 2024. 7, 8, 21

[74] Tong Wu, Jiaqi Wang, Xingang Pan, Xudong XU, Christian Theobalt, Ziwei Liu, and Dahua Lin. Voxurf: Voxel-based efficient and accurate neural surface reconstruction. In *The Eleventh International Conference on Learning Representations*, 2023. 6

[75] Shun Yao, Fei Yang, Yongmei Cheng, and Mikhail G Mozerov. 3d shapes local geometry codes learning with sdf. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2110–2117, 2021. 6

[76] Tatsuya Yokota, Rafal Zdunek, Andrzej Cichocki, and Yukihiko Yamashita. Smooth nonnegative matrix and tensor factorizations for robust multi-way data analysis. *Signal Process.*, 113(C):234–249, 2015. 1, 2, 4, 5

[77] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2

[78] Gizem Yüce, Guillermo Ortiz-Jiménez, Beril Besbinar, and Pascal Frossard. A structured dictionary perspective on implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19228–19238, 2022. 3

[79] Kai Zhao, Sheng Di, Xin Lian, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. Sdrbench: Scientific data reduction benchmark for lossy compressors. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2716–2724, 2020. 1

[80] Yufeng Zheng, Victoria Fernández Abrevaya, Marcel C. Bühler, Xu Chen, Michael J. Black, and Otmar Hilliges. I M Avatar: Implicit morphable head avatars from videos. In *Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 5

F-INR: Functional Tensor Decomposition for Implicit Neural Representations

Supplementary Material

Table of Contents

A. Modes and Backends

Here, we provide more information on the modes and backends used for formulating the F-INR.

A.1. Modes

An INR problem (dimension ≥ 3) can be reformulated as an equivalent F-INR-problem using three modes of functional tensor decompositions. There are even more modes like Higher-Order SVD, Tucker2, Hierarchical Tucker, Tensor Ring, Block Tensor forms [8, 22]. However, in this study, we confine ourselves to the three most common and widely used modes: Canonic Polyadic, Tensor-Train, and Tucker decomposition forms. We refer to works like [8, 22] for comprehensive information about tensor decompositions.

Let $\Phi \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ denote an N-mode tensor of order N, where I_n represents the dimensionality along the
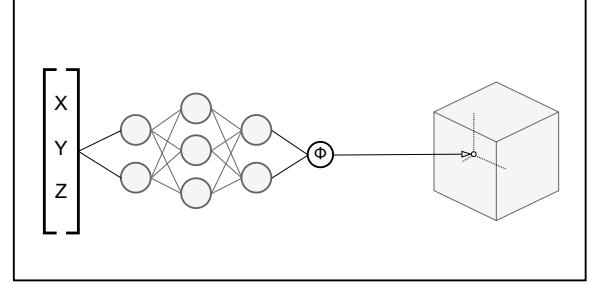

Figure 8. Classical INR of shape $X \times Y \times Z$. A single neural network predicts all the entries.

n-th mode. The objective of tensor decomposition is to approximate Φ by a structured decomposition that minimizes the number of parameters while preserving the data's essential features. Each decomposition represents Φ in terms of factor matrices, vectors, or core tensors, enabling compact and often interpretable representations. We use the symbol $\approx$ to indicate an approximation.

A classical INR is visualized in Figure 8. For simplicity, we visualize three-dimensional INR of original dimensions $X \times Y \times Z$ and can be extended similarly to any arbitrary dimension. To estimate the forward pass complexity, we assume $X = Y = Z = n$ the neural network is of m features and l layers, and the n^3 points are trained as a single batch. Considering the complexity of the multiplication of two layers of m features is m^2, the complexity of a forward pass for a classical INR (m features, l layers) is $O(m^2 l n^3)$ [10].

A.1.1. Canonical Polyadic (CP) Decomposition

The Canonical Polyadic (CP) decomposition [5], also known as PARAFAC or CANDECOMP, approximates a tensor as a sum of rank-one tensors. Specifically, for an N-mode tensor $\Phi \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, the CP decomposition can be formulated as:

$$\Phi \approx \sum_{r=1}^{R} \phi_r^{(1)} \circ \phi_r^{(2)} \circ \dots \circ \phi_r^{(N)} , \tag{8}$$

where $\phi_r^{(n)} \in \mathbb{R}^{I_n} (n \in [1, N])$ represents the factor vector associated with the n-th mode for component r, and R is the rank of the decomposition. Here, $\circ$ denotes the outer product, and the CP decomposition is a sum of R outer products.

An F-INR in CP mode is visualized in Figure 9. For simplicity, we visualize three-dimensional INR of original
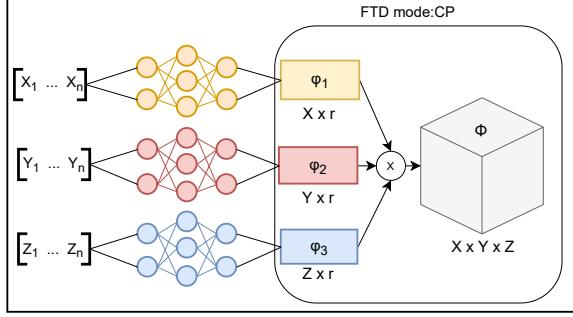
Figure 9. F-INR in CP mode for INR of shape $X \times Y \times Z$. Three individual neural networks predict the factor matrix of rank r for each corresponding dimension. The shape of the matrix/tensor is provided below.

Figure 10. F-INR in TT mode for INR of shape $X \times Y \times Z$. Three individual neural networks predict the *cores* of rank r for each corresponding dimension. The shape of the matrix/tensor is provided below.

dimensions $X \times Y \times Z$ and can be extended similarly to any arbitrary dimension.

In this case, the estimation of forward pass complexity is done in two steps. First, the complexity of forward pass until the factor matrices and then multiplying them together to reconstruct the original tensor. Following the previous case, the first step of three neural networks to output factor matrix of shape $n \times r$ will be $O(3m^2lnr) = O(m^2lnr)$. The next step is to multiply these three-factor matrices of shape $n \times r$. We multiply the first two matrices to get an intermediate tensor of shape $n \times n \times r$ and then multiply this intermediate tensor with the final factor matrix to get the final $n \times n \times n$. So, the complexity of this operation is $O(n^2r^2)$. Combining them both will give the complexity of the forward pass of F-INR in CP mode as:

$$O(m^2lnr + n^2r^2) . \tag{9}$$

A.1.2. Tensor Train (TT) Decomposition

The Tensor Train (TT) [13] decomposition represents a tensor as a sequence of lower-dimensional tensors (often called "cores") linked together in a chain. This decomposition is particularly effective for higher-order tensors due to its sequential structure, which reduces memory requirements. The TT decomposition of an N-mode tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ can be expressed as:

$$\Phi \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \phi_{i_1,r_1}^{(1)} \phi_{i_2,r_1,r_2}^{(2)} \ldots \phi_{i_N,r_{N-1}}^{(N)} , \tag{10}$$

where each $\phi^{(n)}$ represents a core tensor associated with the n-th mode. Here, R_n denotes the TT rank between modes n and $n+1$, and $\phi^{(1)}$ through $\phi^{(N)}$ are the core tensors. In this work, we confine to the case where $R_1 = R_2 = ... = R_n$

Similar to the previous case, we estimate the forward pass complexity in two steps. In the first step of the forward pass of neural networks to predict the tensor cores, there is a three-dimensional tensor of shape $r \times n \times r$. Therefore the complexity of this step is $O(m^2lnr^2)$.

In the next step, these cores of shapes $n \times r, r \times n \times r$, and $n \times r$ are multiplied to get the original $n \times n \times n$. Therefore, the complexity of this step will be $O(n^2r^2)$. Combining these two, the total complexity of F-INR in TT mode will be:

$$O(m^2lnr^2 + n^2r^2) . \tag{11}$$

A.1.3. Tucker Decomposition

The Tucker decomposition [20] is a generalization of the CP decomposition that represents the tensor using a core tensor and multiple factor matrices, providing a flexible way to capture interactions across modes. For a tensor $\Phi \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, the Tucker decomposition is defined as:

$$\Phi \approx \mathcal{C} \times_1 \phi^{(1)} \times_2 \phi^{(2)} \ldots \times_N \phi^{(N)}, \tag{12}$$

where $\mathcal{C} \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ is the core tensor, $\phi^{(n)} \in \mathbb{R}^{I_n \times R_n}$ are the factor matrices for each mode, and $\times_n$ denotes the mode-n tensor-matrix product.

The first step of tucker mode is the same as CP. Therefore, the complexity of the first step in this case will be the same as CP: $O(m^2lnr)$. The second case involves the multiplication of three-factor matrices of shapes $n \times r$ with a core of shape $r \times r \times r$, to get the original tensor of shape $n \times n \times n$. This step involves n^3 computations, each taking $O(r)$ multiplications, therefore having a total complexity of $O(n^3r)$. Therefore, the forward pass complexity of an F-INR in Tucker mode is:
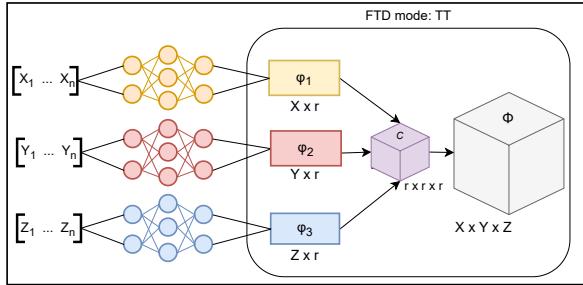
$$O(m^2lnr + n^3r) . \tag{13}$$

Figure 11. F-INR in Tucker mode for INR of shape $X \times Y \times Z$. C is the tucker-core tensor. Three individual neural networks predict the *factor matrices* of rank r for each corresponding dimension. The shape of the matrix/tensor is provided below.

A.2. Backends

As explained, each tensor component of a particular mode for a F-INR-setup is learned using a parameterized MLP. This MLP is what we refer to as the backend. This backend can also be any neural network, but we stick to SOTA architectures of ReLU with Positional Encoding [11], SIREN [18] and WIRE [17]. This is because they are not application-specific and are proven to apply to a wide range of INR-related problems. Moreover, this study aims not to focus on a particular application of INR but on a novel way to formulate the problem by leveraging Tensor decompositions. Nevertheless, any architecture can be used as a backend similarly depending on the application at hand. Here, we explain the backends used in this study and (informally) the impact of F-INR reformulation on the backends.

A.2.1. ReLU MLP with Positional Encoding

Using Fourier features as positional encoding to improve the learning capabilities of ReLU-based MLPs was introduced in [11, 19]. The inputs ν are passed through the mapping

$$\gamma(\nu) = [cos(2\pi B\nu), sin(2\pi B\nu)]^T , \qquad (14)$$

where B is a random Gaussian matrix whose entries are randomly drawn from Gaussian $N(0, \sigma^2)$, this term, σ, referred to as frequency, determines the frequency of the matrix B. The impact of F-INR, or in general, using univariate neural networks becomes prevalent because higher-dimensional inputs need broader frequency coverage to capture complex spatial patterns; otherwise, some features may be poorly represented. In other words, univariate neural networks and, by extension, F-INR-models have more representational capacity for the same spectral coverage. The same explanation applies to the other backends. In this work, we also tested for different frequencies of σ. We show the σ value as a suffix for all the architectures. For example,

ReLU100 means that $\sigma = 100$. This usage of different σ is why some results differ from other implementations.

A.2.2. SIREN

Introduced by [18], SIRENs use periodic sinusoidal activation functions facilitated by a principled weight initialization scheme, where the weights w_i are drawn from the uniform distribution $U(-\sqrt{6/n}, +\sqrt{6/n})$. This initialization ensures that the input to each sine activation is normally distributed with a standard deviation of 1 [18].

Finally, SIREN has the first layer of activations as $sin(\omega_0 w_i x + b_i)$, where $\omega_0 = 30$, to ensure periodicity[18]. In our implementations, we use the same prescribed ω and initialization of weights.

A.2.3. WIRE

The Gabor wavelet activation function was introduced in [17] as a direct extension and generalization of SIREN and Gaussian non-linearity. The wavelet activation offers localization property in a Gaussian/Radial basis activation function and the frequency property offered by positional encoding [11] and SIREN [18]. The complex form of a Gabor wavelet is written as

$$\psi(x; \omega, s) = e^{j\omega x} e^{-|sx|^2} , \qquad (15)$$

where ω is the parameter controlling the frequency and s controlling the scale (localization) of the wavelet activation. These two are hyperparameters, giving rise to a lot of combinations. The authors suggested that the activation function itself is robust in performance and initialization over a large set of combinations of these hyperparameters and suggest using the combination of $\omega = 30, s = 30$, which we use in this paper for all the experiments unless and until specified otherwise.

A.2.4. Hash Encoding

Hash encoding was introduced in Instant Neural Graphics Primitives [12] as an efficient encoding mechanism for spatially varying inputs. Unlike dense grid-based feature representations, hash encoding provides a memory-efficient alternative by mapping input coordinates to a compact set of feature vectors stored in a hash table. This encoding mechanism is particularly beneficial for high-resolution function approximation, as it enables fast feature retrieval while maintaining a lightweight memory footprint.

The fundamental operation of hash encoding, following [12], is defined as follows. Given an input coordinate $\mathbf{x} \in \mathbb{R}^d$, the space is partitioned into L resolution levels, where each level corresponds to a grid of resolution:

$$R_l = R_0 \cdot \alpha^l, \quad l = 0, 1, \dots, L-1 , \qquad (16)$$

where R_0 is the base resolution, α is the per-level scale factor, and L is the total number of levels.

For each level l, the input coordinate is mapped to a discrete grid cell index:

$$\mathbf{i} \cdot l = \lfloor R_l \mathbf{x} \rfloor \ . \tag{17}$$

Rather than storing a dense grid, a hash function is applied to map grid coordinates to a fixed-size hash table of size T:

$$h(\mathbf{i}l) = \left(\sum j = 1^d il, j p_j \right) \mod T \ , \tag{18}$$

where p_j are large prime numbers used to reduce hash collisions. The retrieved feature vector $\mathbf{f}l$ is then interpolated using trilinear interpolation:

$$\mathbf{f}(\mathbf{x}) = \sum c \in 0, 1^d w_c \cdot \mathbf{f}_h(h(\mathbf{i}_l + c)) \ , \tag{19}$$

where w_c are interpolation weights based on the distance from $\mathbf{x}$ to the grid corners.

The final multi-resolution encoding is obtained by concatenating the interpolated features from all levels:

$$\mathbf{f}(\mathbf{x}) = \bigoplus_{l=0}^{L-1} \mathbf{f}_l \ . \tag{20}$$

This encoding scheme enables neural networks to efficiently learn high-dimensional functions with significantly reduced computational and memory overhead. We only employ hash encoding for video and image encoding tasks because of the presence of gradient-based operations in the remaining tasks, which hinders the usage of the global hash grid due to non-differentiability and artifact generation [6, 12]. There are effective ways to implement hash encoding for PDEs and Eikonal geometry solving [6], but we leave this extension to the future.

B. Image Encoding

B.1. Architecture

There are two neural networks for each dimension, x and y, and the outputs of these neural networks are of shapes $x \times r$ and $y \times r$, respectively, which are multiplied to get the original image. Please note that we do not consider the channel a dimension because it is not continuous. Instead, we consider the channel as a variable, which means each point in the space has three associated RGB values. Both neural networks are taken to be two layers with 256 neurons each for all the backends. The frequency(σ) of positional encoding is tested for both 10 and 100 for the ReLU MLPs, which, along with the baseline version, are represented as ReLU0, ReLU10, and ReLU100, the integer representing the frequencies of positional encoding. All the backends are two layers of 256 neurons. For baselines, we use four layers with 256 neurons each.

Figure 12. We display how F-INRs and baseline models compare regarding speed and PSNR value for the image encoding task. This is a visualization of Table 4. The circle radius describes the compression rate compared to the baseline. Models with a rank of 16 are similar to the worst baseline method, indicating that such models cannot fully represent the image data. The first major quality improvements can be seen at rank 128, where the ReLU100 models outperform the baseline while having a roughly 75x speedup. The performance is shown by hash encoding, followed by ReLU100 and WIRE models.

Each model is trained for 50,000 iterations using the Adam optimizer with default parameters to learn the image of a cat with a resolution of 560×720.

B.2. Results

Here, we provide the PSNRs for all the models for varying ranks and the baselines in Figure 12 and Table 4.

We use such visualizations of quantitative results to simplify the information given in the tables. In the visualization, the green shaded region indicates that the model outperformed the best-performing baseline, while the red region means it performed worse than the worst-performing baseline. We provide such visualizations for all the experiments. The trend is this: As the rank increases, the PSNR increases, and the representation capacity increases. This comes along with a slight increase in computation time. Larger ranks even outperform the baselines, having the same backend neural network, all while still having a 100x speedup. This proves the effectiveness of F-INRs. We also find some interesting patterns in the speedups. The highest speedup is achieved for Hash encoding, owing to using 1D hash tables instead of 2D. Also, for positional encoding, the time increases with increasing frequencies because of an increase in the size of the encoded input for higher frequencies.

Backend	Rank	PSNR (dB)	SSIM	Time (s)
ReLU0	-	25.72 ± 0.31	0.73 ± 0.01	5066
ReLU10	-	30.15 ± 0.19	0.78 ± 0.00	5618
ReLU100	-	31.53 ± 0.18	0.81 ± 0.00	5920
SIREN	-	30.90 ± 0.19	0.78 ± 0.00	5280
WIRE	-	32.09 ± 0.20	0.85 ± 0.00	5290
ReLU+Hash	-	33.88 ± 0.14	0.92 ± 0.01	2100
ReLU0	16	24.34 ± 0.53	0.70 ± 0.00	55
ReLU0	32	25.89 ± 0.20	0.74 ± 0.00	56
ReLU0	64	26.76 ± 0.18	0.77 ± 0.00	58
ReLU0	128	26.68 ± 0.23	0.77 ± 0.00	78
ReLU0	256	26.81 ± 0.35	0.78 ± 0.01	104
ReLU0	316	26.92 ± 0.31	0.78 ± 0.00	121
ReLU10	16	24.68 ± 0.16	0.71 ± 0.00	56
ReLU10	32	27.24 ± 0.28	0.76 ± 0.00	58
ReLU10	64	29.65 ± 0.21	0.82 ± 0.00	61
ReLU10	128	30.54 ± 0.49	0.86 ± 0.01	80
ReLU10	256	30.86 ± 0.49	0.84 ± 0.02	106
ReLU100	16	24.70 ± 0.19	0.71 ± 0.00	55
ReLU100	32	27.36 ± 0.20	0.77 ± 0.00	57
ReLU100	64	30.52 ± 0.18	0.84 ± 0.00	63
ReLU100	128	33.89 ± 0.28	0.90 ± 0.00	82
ReLU100	256	34.26 ± 0.37	0.92 ± 0.00	108
ReLU100	**316**	$\mathbf{34.55} \pm \mathbf{0.31}$	$\mathbf{0.92} \pm \mathbf{0.00}$	**123**
SIREN	16	26.24 ± 0.15	0.69 ± 0.00	45
SIREN	32	27.08 ± 0.23	0.73 ± 0.00	47
SIREN	64	28.03 ± 0.22	0.77 ± 0.00	54
SIREN	128	30.87 ± 0.47	0.78 ± 0.00	65
SIREN	256	31.14 ± 0.29	0.84 ± 0.00	86
SIREN	316	31.85 ± 0.17	0.85 ± 0.00	90
WIRE	16	26.59 ± 0.15	0.70 ± 0.00	46
WIRE	32	29.93 ± 0.24	0.75 ± 0.00	41
WIRE	64	30.16 ± 0.29	0.82 ± 0.00	43
WIRE	128	32.63 ± 0.38	0.88 ± 0.00	61
WIRE	256	33.94 ± 0.38	0.89 ± 0.00	86
WIRE	316	34.19 ± 0.11	0.89 ± 0.00	101
ReLU + Hash	16	24.66 ± 0.47	0.71 ± 0.03	23
ReLU + Hash	32	27.12 ± 0.42	0.75 ± 0.03	26
ReLU + Hash	64	29.25 ± 0.40	0.79 ± 0.02	33
ReLU + Hash	128	32.43 ± 0.41	0.87 ± 0.01	41
ReLU + Hash	**256**	$\mathbf{35.61} \pm \mathbf{0.14}$	$\mathbf{0.93} \pm \mathbf{0.01}$	**76**
ReLU + Hash	**316**	$\mathbf{35.91} \pm \mathbf{0.11}$	$\mathbf{0.93} \pm \mathbf{0.01}$	**85**

Table 4. PSNR and SSIM values for more combinations of ranks and backends. Rank - means the implementation is the baseline INR for a full pixel-coordinate batch. The times are calculated for complete 50000 iterations for all the runs. We see that F-INR gives faster, better results for the same backend for a large enough rank. The top three performing models based on PSNRs are highlighted here. We also observe a strong correlation between the PSNR and SSIM values.

C. Video Encoding

C.1. Architecture

This is similar to image encoding with an added dimension of frames. From the results of image encoding, we employ
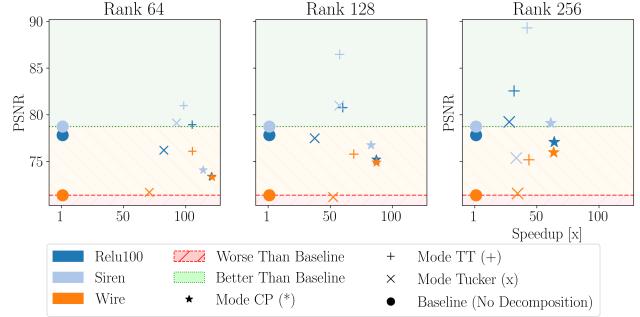

Figure 13. We visualize PSNR values given in the Table 5. Here, different markers are assigned for each mode, and different colors are assigned for each backend. We only use rank ≥ 64 and three backends, WIRE, SIREN, and ReLU100. Many configurations surpass the baseline implementation (above green), and of all the modes, TT has consistently the best results.

only ReLU100 (i.e., P.E with $\sigma = 100$), along with SIREN and WIRE. Here, three modes are employed, and depending on the mode, the neural network learns the corresponding component of rank r. For TT, we used the frame dimension for this three-dimensional component since there is a component with three dimensions (a connector).

We train for 50000 epochs with Adam optimizer with default settings. All the backends are 256 neurons with three layers. We use five layers with 256 neurons each for baselines and take the batch size to fill the GPU memory. For F-INRs we take the full batch size.

C.2. Results

The open source video of the face of the girl, the results for various ranks and backends are given in Figure 13 and Table 5. Our results indicate that the proposed method yields faster and more accurate results for the same backend when the rank is sufficiently large. Notably, the top three models in terms of PSNR are highlighted in the table. We omit Hash encoding because of its poor performance when used for all the neural networks as the backend. However, the best performance is achieved when the neural network for frame dimension has a Positional Encoding and neural networks for spatial networks have hash encoding. We conducted additional experiments for this specific combination for rank 256 to show how the modularity of F-INRs allows for getting better solutions. Furthermore, our experiments reveal that the WIRE backend exhibits suboptimal performance for the video encoding task under default parameters, whereas the TT mode achieves the best results. This poor performance of WIRE may be remedied by decreasing the scale parameter. We provide the videos together in MP4 format with the supplementary material.

Backend	Mode	Rank	PSNR (dB)	SSIM	Time (s)
ReLU100	-	-	77.82 ± 0.18	0.93	13327
SIREN	-	-	78.74 ± 0.19	0.95	12735
WIRE	-	-	71.38 ± 0.22	0.75	12923
ReLU + Hash	-	-	71.82 ± 0.23	0.74	8243
	CP	64	73.38 ± 0.03	0.79	110
	CP	128	75.21 ± 0.15	0.83	152
	CP	256	77.07 ± 0.33	0.86	208
	TT	64	78.92 ± 0.32	0.91	126
ReLU100	TT	128	80.74 ± 0.17	0.93	221
	TT	**256**	$\mathbf{82.55 \pm 0.63}$	**0.95**	**420**
	Tucker	64	76.20 ± 0.81	0.85	161
	Tucker	128	77.48 ± 0.81	0.88	355
	Tucker	256	79.23 ± 0.83	0.91	479
	CP	64	74.08 ± 0.20	0.80	111
	CP	128	76.74 ± 0.14	0.85	153
	CP	256	79.10 ± 0.16	0.89	208
	TT	64	80.96 ± 0.31	0.93	129
SIREN	**TT**	**128**	$\mathbf{86.46 \pm 0.19}$	**0.97**	**220**
	TT	**256**	$\mathbf{88.28 \pm 0.19}$	**0.98**	**301**
	Tucker	64	79.11 ± 0.18	0.90	137
	Tucker	128	80.98 ± 0.27	0.93	222
	Tucker	256	75.37 ± 0.52	0.90	380
	CP	64	73.32 ± 1.45	0.78	1063
	CP	128	74.89 ± 0.81	0.82	148
	CP	256	75.97 ± 0.85	0.84	203
	TT	64	76.09 ± 0.83	0.85	122
WIRE	TT	128	75.77 ± 1.88	0.84	187
	TT	256	75.17 ± 1.81	0.83	296
	Tucker	64	71.69 ± 1.96	0.74	182
	Tucker	128	71.19 ± 1.88	0.73	247
	Tucker	256	71.57 ± 2.05	0.74	374
ReLU100 +Hash	CP	256	80.4 ± 0.20	0.92	186
ReLU100 + Hash	**TT**	**256**	$\mathbf{89.2 \pm 0.13}$	**0.98**	**385**
ReLU100 + Hash	Tucker	256	80.35 ± 2.63	0.74	427

Table 5. Mean PSNR and SSIM values across all the encoded video frames and ground truth for more combinations of ranks and backends. The times are calculated for complete 50000 iterations for all the runs. We see that F-INR s gives faster, better results for the same backend for a large enough rank. The top three performing models based on PSNRs are highlighted here. The WIRE backend performs poorly with the video encoding task (for the default parameters), and TT mode performs the best. The final row contains a combination of hash and positional encoding, yielding better results. This table is visualized for speedup comparisons (without combinations of PE and Hash) in Figure 13.

D. Single Image Super Resolution and Denoising

Single Image Super-Resolution (SISR) and image denoising are fundamental tasks in implicit neural representations (INRs), closely related to those explored in WIRE [17]. These tasks test the ability of an INR model to reconstruct fine details and filter out unwanted noise, with the backend architecture playing a dominant role in determining performance. At the same time, tensor decomposition primarily contributes to computational efficiency.
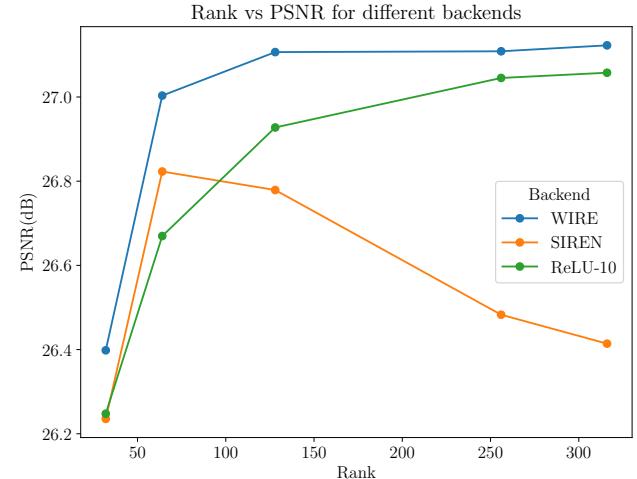

Figure 14. Evolution of Rank vs. PSNR for different backends for Single Image Super-resolution task. We observe WIRE performs best and is robust to rank. SIREN degrades performance with increased rank.

We use a standard butterfly image for the super-resolution task and induce a 4× sparsity by sub-sampling pixels before training, following the same setup as in [17]. The original dimensions of image are 1356 x 2040, while the downsampled image is: 339×510 The F-INR model is trained on this sparse representation for backends and ranks until 316. At inference, the learned function is evaluated at the original resolution, producing a reconstructed high-resolution image. The reconstruction is then quantitatively compared to the ground truth.

Our experiments show that the choice of backend architecture is the most significant factor affecting performance. The decomposition method does not inherently improve reconstruction quality but accelerates training and inference. Notably, WIRE emerges as the best-performing backend, aligning with observations from its baseline implementation. In contrast, hash encoding and large positional encodings introduce artifacts as they struggle to generalize from sparse training data, leading to aliasing effects and undesired high-frequency components [6, 12, 19].

Interestingly, rank has minimal impact on super-resolution performance. Since the task primarily requires the model to interpolate missing pixel values rather than compressing or filtering data, increasing rank does not significantly alter results. This highlights the backend's importance over decomposition choices in super-resolution tasks. The visualizations and the evolution of PSNR values over different ranks and backends are provided in Figure 14 and Figure 15.

For the denoising task, we introduce shot Gaussian noise to the original image of a parrot (1356×2040) and train
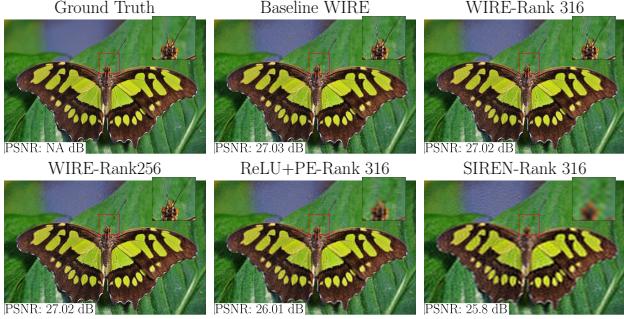
Figure 15. Single Image Super-Resolution using F-INR. Here, we observe that qualitatively F-INR performs the same as baseline WIRE. Therefore, F-INR preserves the inherent quality salient to WIRE architecture, forming better priors. Image taken from [17].

Figure 17. Visualizations for the denoising task of a parrot. Image taken from [17]. F-INR with WIRE backend retains the robustness of WIRE, emphasizing the influence of backend for the task-specific performance of F-INRs.
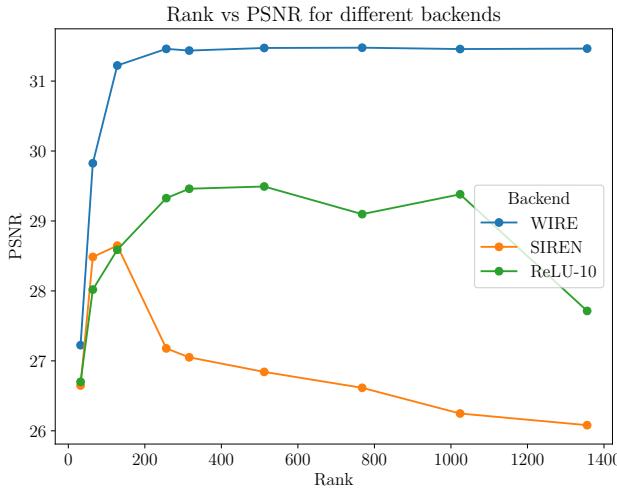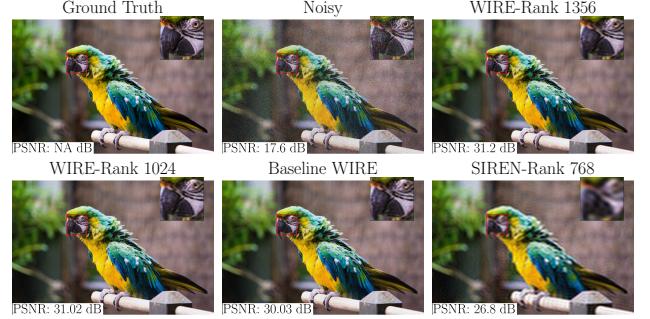

Figure 16. Rank vs PSNR for denoising task for various backends of F-INR. We observe that the wire is robust and almost rank-independent. In contrast, SIREN and ReLU, with PE, have a deterioration with increasing rank, which might be due to overfitting of the noisy image.

the F-INR model to reconstruct the clean version, following [17]. The performance is evaluated by comparing the denoised output to the ground truth, assessing whether the INR learns to remove noise effectively while preserving structural details. The plot for comparing various Ranks and backends is given in Figure 16, and the visualizations are provided in Figure 17.

Unlike super-resolution, where rank is negligible, rank selection is critical for denoising. A rank that is too high negatively impacts performance, as an excessively expressive model overfits the noise rather than learning the underlying clean structure. Lower-rank decompositions act as a natural regularizer, preventing the model from capturing high-frequency noise and improving denoising performance.

As in the super-resolution task, WIRE remains the most effective backend, producing cleaner reconstructions. Hash encoding and large positional encodings again lead to undesirable artifacts, reinforcing the importance of selecting a backend suited for structured image tasks.

E. Super Resolution of simulations

Super-resolution setup is: We have sparse, discrete observations, and we train an INR with these sparse observations and physics-informed loss [15] to get a continuous, differentiable simulation encoded in the neural network model. As the main paper discusses, this has similarities and differences from simulating the complete system using PINNs. We also performed experiments on models specifically developed for super-resolution of simulations, most notably PhySR [16] and MeshFreeFlowNet [7] and did not include because they do not come under the family of INRs. There is no straightforward F-INR- equivalent of such methods. We include them here for completeness, and we observe that even for standard backends, F-INR shows competitive performance.

E.1. Architecture

The training strategy is similar to that of [1]. The input for each neural network is coordinates of respective dimensions, and the output is a velocity vector in the x and y direction, respectively. The Navier-Stokes equation is in its vorticity form, so this velocity is converted into vorticity, which is then used to enforce the PDE loss term and compressibility constraint. The sparse observations are also given as a loss term along with the initial condition. The weights used for physics loss term, initial, and sparse observation loss terms are 1, 10^3, and 10^4, respectively. These are tuned to get the best-performing results. We uniformly sample 100 points per dimension as collocation points to enforce the PDE loss. We use WIRE and SIREN backends

and ReLU10 and ReLU20 for all three modes. Each neural network for F-INR is three layers of 256 neurons each. For the baseline implementations, we use six layers of 256 neurons each. MeshFreeFlowNet [7] and PhySR [16] are implemented as the authors prescribe. Since they involve 3D operations, a F-INR equivalent is not as straightforward as other backends and, hence, is out of the scope of this current work. The sparse data used for training is of shape $10 \times 64 \times 64$, uniformly sampled from the original resolution dataset $101 \times 128 \times 128$. Thereby inducing a 40x sparsity. Each model is trained for 50000 Adam iterations, with resampling of collocation points for every 1000 iterations. We test the learned models by predicting the simulation in the original resolution of the dataset, thereby achieving the super-resolution task. We quantify the relative L_2 error between the ground truth and the prediction.

E.2. Results

The results are given in Figure 18 and Table 6. We see that F-INR achieves better results than the baseline, starting from a Rank as low as 32. This highlights the effectiveness of these models in terms of speed and efficiency. A visualization of the prediction of F-INR with Mode Tucker, Rank 32, and WIRE backend is provided in Figure 20.

E.2.1. Sparsity Ablation

Here, we tested the performance of the F-INR s for different sparsity levels. We use the same setup and combination of models and train them using three varying levels of sparsity: 160x, 40x, and 10x. We show that even for a sparsity level 160x, F-INR s gives a competitive performance. We also observe that more data corresponds to a better solution overall. The results are given in Figure 19. We see that for all the backends, sparsities, and ranks, TT mode stands out as best performing, followed by Tucker mode and CP mode, respectively.

F. Geometry Learning via Signed Distance Functions

The task of Geometry encoding via SDF involves solving the Eikonal PDE. The input is the voxel coordinate, and the output is the corresponding SDF value. We use three publicly available models that have intricate details [2, 3, 21], and refer to them as the Armadillo, Lucy, and Thai statues. For training, the `.obj` files are converted into SDF using `mesh2sdf` package [23]. The learned SDFs are visualized using marching cubes with Laplacian smoothing [9].

F.1. Architecture and training

For F-INR s, Three neural networks have the input of coordinates and output of the r-ranked tensor component. All three modes are used. For backends, we observe that the

Figure 18. L_2 Error vs. Speedup plots for Super-Resolution of simulation task. This is a visualization of Table 6. This demonstrates that the proposed method consistently outperforms baseline methods in terms of speed and solution quality. Notably, the ReLU20 backend exhibits poor performance, whereas the ReLU10 backend achieves superior results, surpassing baseline methods. This highlights the sensitivity of the positional encoding parameter σ. Our results show that the TT mode consistently yields better across all modes, underscoring its effectiveness in this context.

results deteriorate for ReLU with positional encoding for $\sigma = 20$. Hence, we only include $\sigma = 10$, $\sigma = 20$.

Three neural networks of three layers, each having 256 neurons, are used for all the backends and modes. The baselines have five layers with the same number of neurons. We also applied SOTA methods like DeepSDF [14] and IGR [4]. They are usually trained with point clouds, but we trained them here with the same voxel grid that we train F-INR with. They are trained with the same instructions prescribed by the authors.

Figure 19. L_2 Errors for varying sparsity levels for super-resolution using F-INRs. Three levels of sparsity are tested for all the modes and backends. We observe that ReLU20 performs badly, and mode TT performs the best. Less sparsity leads to a better solution because more data is available. Nevertheless, F-INR s achieves competitive results even for higher sparsities.

All the models are trained for 50000 Adam iterations, and the model with the least loss is saved for the prediction.

F.2. Results

We use IoU and L_2 Error metrics to quantify the predicted values against the ground truth. All the results for Lucy are provided in Figure 24 and Table 8. For Armadillo, Figure 22 and Table 7. Finally, for the Thai statue, the results are provided in Table 9 and Figure 26 and also visualized in Figure 23. All the results show that F-INR s achieves a better result in less time. We also tested the effect of the Eikonal PDE on the solution. We vary the relative weight of the Eikonal term w.r.t to other loss terms for the best performing F-INR, i.e., Mode TT with Rank 128 and with ReLU10 backend, and the results are given in Figure 25. We observe that the Eikonal PDE significantly affects finding a better solution.

Figure 20. Visualization of vorticity for the super-resolution tasks using F-INR: **Mode Tucker with Rank 32 and WIRE backend**. We achieve good prediction closer to ground truth with a smaller rank of 32, highlighting the effectiveness of F-INR.

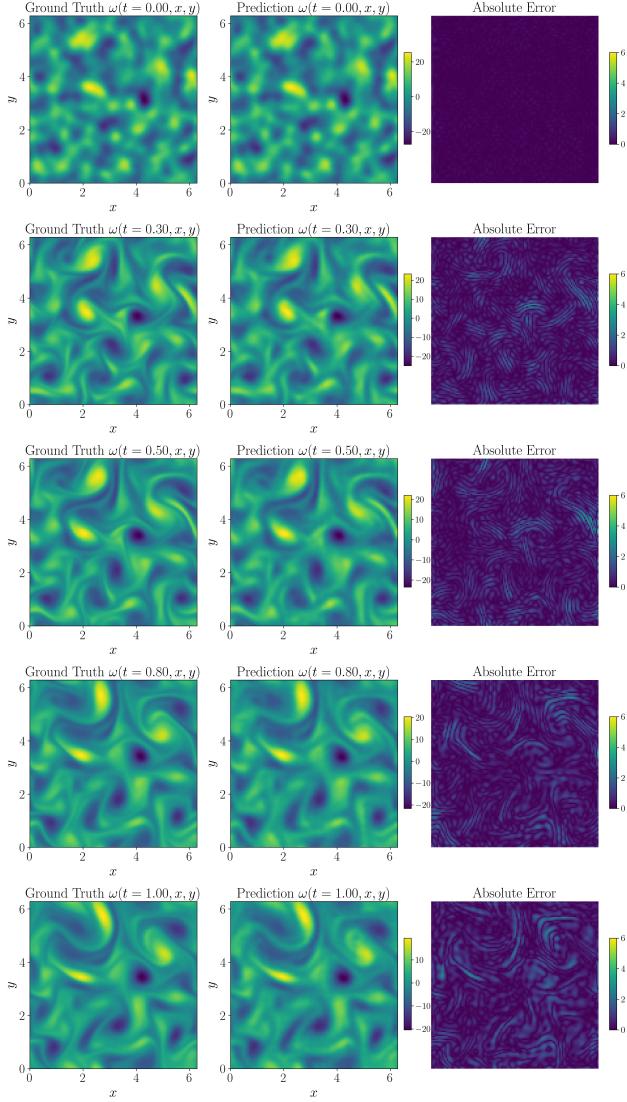| Backend	Mode	Rank	L_2 Error	Time (hh:mm)
ReLU0	-	-	0.426 ± 0.104	20:23
ReLU20	-	-	0.773 ± 0.211	20:48
ReLU10	-	-	0.097 ± 0.009	20:30
WIRE	-	-	0.073 ± 0.004	20:25
SIREN	-	-	0.184 ± 0.010	20:24
ModifiedPINN [72]	-	-	0.074 ± 0.008	28:40
CausalPINN [73]	-	-	0.070 ± 0.011	33:12
MFF Net [25]	-	-	0.048 ± 0.003	35:18
PhySR [51]	-	-	0.038 ± 0.020	27:05
WIRE	CP	16	0.320 ± 0.029	0:35
	CP	32	0.185 ± 0.021	0:37
	CP	64	0.088 ± 0.018	0:49
	CP	128	0.046 ± 0.007	1:08
	CP	256	0.043 ± 0.007	1:42
	TT	16	0.207 ± 0.068	0:35
	TT	32	0.079 ± 0.015	0:39
	TT	64	0.034 ± 0.004	0:59
	TT	128	0.035 ± 0.002	1:44
	TT	256	0.032 ± 0.002	1:56
	Tucker	16	0.215 ± 0.051	0:35
	Tucker	32	0.070 ± 0.023	0:37
	Tucker	64	0.061 ± 0.019	0:49
	Tucker	128	0.036 ± 0.003	1:08
	Tucker	256	0.034 ± 0.004	2:03
SIREN	CP	16	0.342 ± 0.053	0:35
	CP	32	0.193 ± 0.077	0:37
	CP	64	0.078 ± 0.008	0:48
	CP	128	0.044 ± 0.005	1:07
	CP	256	0.040 ± 0.010	1:52
	TT	16	0.521 ± 0.070	0:32
	TT	32	0.164 ± 0.074	0:36
	TT	64	0.038 ± 0.008	0:53
	TT	128	0.048 ± 0.005	1:31
	TT	256	0.045 ± 0.003	1:58
	Tucker	16	0.652 ± 0.087	0:31
	Tucker	32	0.238 ± 0.071	0:35
	Tucker	64	0.113 ± 0.058	0:52
	Tucker	128	0.828 ± 0.161	1:36
	Tucker	256	0.635 ± 0.089	2:00
ReLU0	CP	16	0.473 ± 0.037	0:34
	CP	32	0.379 ± 0.049	0:37
	CP	64	0.320 ± 0.078	0:48
	CP	128	0.314 ± 0.078	1:07
	TT	16	0.396 ± 0.065	0:32
	TT	32	0.345 ± 0.075	0:35
	TT	64	0.286 ± 0.061	0:52
	TT	128	0.327 ± 0.078	1:30
	Tucker	16	0.422 ± 0.059	0:30
	Tucker	32	0.408 ± 0.032	0:34
	Tucker	64	0.324 ± 0.086	0:51
	Tucker	128	0.325 ± 0.078	1:34
ReLU10	CP	16	0.344 ± 0.012	0:32
	CP	32	0.199 ± 0.009	0:35
	CP	64	0.112 ± 0.008	0:47
	CP	128	0.050 ± 0.015	1:05
	CP	256	0.044 ± 0.010	1:55
	TT	16	0.247 ± 0.049	0:30
	TT	32	0.084 ± 0.015	0:33
	TT	64	0.032 ± 0.001	0:51
	TT	128	0.030 ± 0.002	1:24
	TT	256	0.030 ± 0.002	1:59
	Tucker	16	0.264 ± 0.045	0:29
	Tucker	32	0.073 ± 0.009	0:32
	Tucker	64	0.038 ± 0.005	0:49
	Tucker	128	0.032 ± 0.004	1:34
	Tucker	256	0.032 ± 0.005	2:03
ReLU20	CP	16	0.778 ± 0.337	0:36
	CP	32	0.852 ± 0.273	0:38
	CP	64	0.817 ± 0.338	0:50
	CP	128	0.776 ± 0.085	1:10
	CP	256	0.079 ± 0.025	2:27
	TT	16	0.689 ± 0.205	0:35
	TT	32	0.744 ± 0.276	0:39
	TT	64	0.810 ± 0.134	0:58
	TT	128	0.881 ± 0.178	1:42
	TT	256	0.772 ± 0.349	2:30
	Tucker	16	0.845 ± 0.358	0:34
	Tucker	32	0.912 ± 0.418	0:37
	Tucker	64	0.773 ± 0.137	0:54
	Tucker	128	0.766 ± 0.062	1:38
	Tucker	256	0.789 ± 0.084	2:39

Table 6. Comparison of L_2 Errors for F-INR s and baseline implementations for super-resolution of decaying vorticity simulation using Navier Stokes equation. We tabulate all the combinations of ranks, modes, and backends. F-INR s consistently outperforms, having lesser L_2 error and convergence time than baseline implementations.

Figure 21. Visualization of vorticity for the super-resolution tasks using F-INR: **Mode TT with Rank 64 and ReLU10 backend**. We achieve good prediction closer to ground truth with a rank of 64; this solution surpasses the baseline implementations, highlighting the effectiveness of F-INR.

Figure 22. Quantitative results IoU vs Speedup, for model Armadillo. The general trend is that a higher rank leads to a better IoU. Tensor-Train [13] models perform the best in several models compared to the other two models.

Figure 23. Here, we visualize the qualitative differences in complex Thai statuette 3D scan reconstructions. All models have been obtained from SDF using the marching cubes algorithm with the additional step of Laplacian smoothing to reduce introduced artifacts. We compare the results to the obtained ground truth SDF, the worst and best-performing baseline methods, and the best-decomposed version. Additionally, we show how a too low rank ($r = 16$) does not yield a successful reconstruction.

Figure 24. Quantitative results IoU vs Speedup, for model Lucy. The general trend is that a larger rank leads to a better IoU. Tensor-Train models perform the best when compared to the other two models, and in several models, they are better than baselines.

Figure 25. The effect of Eikonal term on the learned SDF for Lucy model. We varied the relative weight of the Eikonal term for the best performing F-INR: Mode TT Rank 128 and ReLU10 Backend. We observe that the presence of the Eikonal term significantly affects the solution, and after a relative weight of around 0.5, its effect is constant.

Table 7.

Backend	Mode	Rank	IoU	L_2 Error	Time (s)
ReLU0	-	-	0.941 ± 0.002	0.295 ± 0.002	15057
ReLU10	-	-	0.950 ± 0.001	0.233 ± 0.002	15428
ReLU20	-	-	0.989 ± 0.005	0.135 ± 0.003	15929
WIRE	-	-	0.988 ± 0.003	0.154 ± 0.004	15134
SIREN	-	-	0.989 ± 0.004	0.127 ± 0.006	15165
DeepSDF [47]	-	0	0.989 ± 0.002	0.126 ± 0.005	18932
IGR [15]	-	0	0.990 ± 0.004	0.127 ± 0.004	17533
	CP	16	0.930 ± 0.005	0.338 ± 0.015	157
	CP	32	0.941 ± 0.004	0.301 ± 0.009	174
	CP	64	0.950 ± 0.006	0.291 ± 0.008	217
	CP	128	0.947 ± 0.002	0.295 ± 0.009	289
	TT	16	0.951 ± 0.004	0.285 ± 0.007	156
ReLU0	TT	32	0.955 ± 0.002	0.266 ± 0.008	178
	TT	64	0.957 ± 0.003	0.257 ± 0.011	233
	TT	128	0.962 ± 0.003	0.239 ± 0.002	423
	Tucker	16	0.942 ± 0.009	0.316 ± 0.009	155
	Tucker	32	0.947 ± 0.003	0.295 ± 0.014	174
	Tucker	64	0.954 ± 0.002	0.267 ± 0.007	238
	Tucker	128	0.957 ± 0.004	0.250 ± 0.014	441
	CP	16	0.961 ± 0.013	0.279 ± 0.011	170
	CP	32	0.983 ± 0.001	0.201 ± 0.003	184
	CP	64	0.988 ± 0.000	0.150 ± 0.003	218
	CP	128	0.990 ± 0.000	0.128 ± 0.002	284
	TT	16	0.987 ± 0.001	0.155 ± 0.003	174
WIRE	TT	32	0.991 ± 0.000	0.119 ± 0.002	180
	TT	64	0.994 ± 0.000	0.094 ± 0.001	248
	TT	128	0.997 ± 0.000	0.078 ± 0.002	455
	Tucker	16	0.971 ± 0.002	0.243 ± 0.004	168
	Tucker	32	0.986 ± 0.001	0.175 ± 0.004	188
	Tucker	64	0.989 ± 0.002	0.137 ± 0.004	244
	Tucker	128	0.992 ± 0.000	0.121 ± 0.002	431
	CP	16	0.951 ± 0.012	0.280 ± 0.011	169
	CP	32	0.985 ± 0.002	0.187 ± 0.003	188
	CP	64	0.994 ± 0.000	0.128 ± 0.002	222
	CP	128	0.997 ± 0.000	0.092 ± 0.002	298
	TT	16	0.992 ± 0.000	0.138 ± 0.004	169
ReLU20	TT	32	0.998 ± 0.001	0.088 ± 0.002	179
	TT	64	1.000 ± 0.000	0.064 ± 0.003	246
	TT	128	1.000 ± 0.000	0.055 ± 0.012	460
	Tucker	16	0.963 ± 0.008	0.247 ± 0.006	168
	Tucker	32	0.990 ± 0.039	0.165 ± 0.030	184
	Tucker	64	0.957 ± 0.165	0.189 ± 0.059	241
	Tucker	128	0.816 ± 0.189	0.230 ± 0.059	428
	CP	16	0.941 ± 0.005	0.295 ± 0.006	166
	CP	32	0.983 ± 0.001	0.200 ± 0.005	173
	CP	64	0.990 ± 0.000	0.151 ± 0.001	214
	CP	128	0.992 ± 0.000	0.128 ± 0.001	286
	TT	16	0.994 ± 0.001	0.125 ± 0.002	160
SIREN	TT	32	1.000 ± 0.000	0.066 ± 0.005	175
	TT	64	1.000 ± 0.000	0.053 ± 0.020	236
	TT	128	1.000 ± 0.000	0.095 ± 0.014	422
	Tucker	16	0.968 ± 0.005	0.233 ± 0.006	156
	Tucker	32	0.990 ± 0.000	0.158 ± 0.004	171
	Tucker	64	0.996 ± 0.000	0.114 ± 0.005	235
	Tucker	128	0.997 ± 0.388	0.138 ± 0.368	451
	CP	16	0.952 ± 0.009	0.290 ± 0.008	148
	CP	32	0.980 ± 0.003	0.214 ± 0.007	169
	CP	64	0.986 ± 0.000	0.160 ± 0.003	201
	CP	128	0.989 ± 0.000	0.133 ± 0.001	282
	TT	16	0.979 ± 0.000	0.187 ± 0.002	149
ReLU10	TT	32	0.979 ± 0.000	0.169 ± 0.001	166
	TT	64	0.979 ± 0.000	0.166 ± 0.001	219
	TT	128	0.978 ± 0.001	0.170 ± 0.003	404
	Tucker	16	0.965 ± 0.001	0.242 ± 0.003	145
	Tucker	32	0.976 ± 0.001	0.202 ± 0.004	167
	Tucker	64	0.978 ± 0.006	0.184 ± 0.019	229
	Tucker	128	0.892 ± 0.270	0.394 ± 0.229	440

Table 7. Results for geometry encoding task: Here we provide both IoU and L_2 Error between the predicted SDFs and Ground truth, for Armadillo model, taken from [65]. A visualization of IoU values is provided in Figure 22.

Table 8.

Backend	Mode	Rank	IoU	L_2 Error	Time(s)
ReLU0	-	-	0.884 ± 0.007	0.213 ± 0.003	15963
ReLU10	-	-	0.908 ± 0.009	0.188 ± 0.004	16428
ReLU20	-	-	0.973 ± 0.006	0.163 ± 0.004	16923
SIREN	-	-	0.971 ± 0.005	0.126 ± 0.010	16138
WIRE	-	-	0.975 ± 0.009	0.062 ± 0.064	16157
DeepSDF [47]	-	0	0.978 ± 0.010	0.062 ± 0.015	18375
IGR [15]	-	0	0.985 ± 0.004	0.067 ± 0.014	16912
	CP	16	0.852 ± 0.007	0.225 ± 0.005	178
	CP	32	0.869 ± 0.015	0.213 ± 0.007	194
	CP	64	0.883 ± 0.007	0.213 ± 0.003	228
	CP	128	0.885 ± 0.006	0.211 ± 0.004	304
	TT	16	0.884 ± 0.009	0.207 ± 0.007	177
ReLU0	TT	32	0.896 ± 0.009	0.198 ± 0.005	195
	TT	64	0.907 ± 0.003	0.191 ± 0.002	250
	TT	128	0.923 ± 0.003	0.181 ± 0.002	448
	Tucker	16	0.864 ± 0.006	0.220 ± 0.008	170
	Tucker	32	0.880 ± 0.003	0.209 ± 0.005	192
	Tucker	64	0.904 ± 0.006	0.198 ± 0.003	256
	Tucker	128	0.922 ± 0.005	0.186 ± 0.004	465
	CP	16	0.869 ± 0.019	0.217 ± 0.009	190
	CP	32	0.933 ± 0.013	0.159 ± 0.010	204
	CP	64	0.961 ± 0.002	0.121 ± 0.003	238
	CP	128	0.973 ± 0.001	0.100 ± 0.002	314
	TT	16	0.941 ± 0.007	0.133 ± 0.015	194
WIRE	TT	32	0.976 ± 0.002	0.089 ± 0.002	210
	TT	64	0.987 ± 0.001	0.069 ± 0.003	266
	TT	128	0.991 ± 0.001	0.058 ± 0.003	475
	Tucker	16	0.909 ± 0.014	0.198 ± 0.005	188
	Tucker	32	0.953 ± 0.004	0.137 ± 0.003	208
	Tucker	64	0.971 ± 0.001	0.111 ± 0.002	264
	Tucker	128	0.977 ± 0.001	0.092 ± 0.005	452
	CP	16	0.829 ± 0.021	0.217 ± 0.034	191
	CP	32	0.912 ± 0.012	0.176 ± 0.018	205
	CP	64	0.962 ± 0.004	0.118 ± 0.002	239
	CP	128	0.989 ± 0.001	0.076 ± 0.002	315
	TT	16	0.928 ± 0.010	0.142 ± 0.014	195
ReLU20	TT	32	0.988 ± 0.001	0.070 ± 0.003	211
	TT	64	0.999 ± 0.000	0.055 ± 0.005	267
	TT	128	0.999 ± 0.000	0.045 ± 0.003	477
	Tucker	16	0.892 ± 0.043	0.185 ± 0.007	189
	Tucker	32	0.948 ± 0.021	0.146 ± 0.005	209
	Tucker	64	0.984 ± 0.039	0.104 ± 0.028	265
	Tucker	128	0.860 ± 0.035	0.146 ± 0.043	450
	CP	16	0.882 ± 0.022	0.235 ± 0.010	181
	CP	32	0.928 ± 0.016	0.193 ± 0.008	199
	CP	64	0.967 ± 0.003	0.141 ± 0.004	231
	CP	128	0.979 ± 0.001	0.119 ± 0.001	307
	TT	16	0.894 ± 0.010	0.136 ± 0.002	179
SIREN	TT	32	0.983 ± 0.002	0.066 ± 0.002	197
	TT	64	0.998 ± 0.000	0.063 ± 0.011	252
	TT	128	0.994 ± 0.002	0.105 ± 0.010	454
	Tucker	16	0.899 ± 0.028	0.182 ± 0.007	172
	Tucker	32	0.919 ± 0.015	0.141 ± 0.004	195
	Tucker	64	0.977 ± 0.018	0.098 ± 0.020	259
	Tucker	128	0.980 ± 0.030	0.121 ± 0.374	469
	CP	16	0.885 ± 0.042	0.231 ± 0.018	170
	CP	32	0.903 ± 0.010	0.209 ± 0.004	187
	CP	64	0.947 ± 0.005	0.159 ± 0.004	221
	CP	128	0.956 ± 0.002	0.135 ± 0.006	302
	TT	16	0.932 ± 0.011	0.179 ± 0.001	169
ReLU10	TT	32	0.943 ± 0.007	0.162 ± 0.002	186
	TT	64	0.945 ± 0.006	0.159 ± 0.002	239
	TT	128	0.937 ± 0.007	0.164 ± 0.008	424
	Tucker	16	0.911 ± 0.013	0.223 ± 0.006	165
	Tucker	32	0.932 ± 0.004	0.190 ± 0.001	187
	Tucker	64	0.930 ± 0.032	0.168 ± 0.244	249
	Tucker	128	0.851 ± 0.038	0.458 ± 0.216	460

Table 8. Results for Geometry encoding task: Here we provide both IoU and L2 Error between the predicted SDFs and Ground truth, for Lucy model, taken from [12]. A visualization of IoU values is provided in Figure 24.
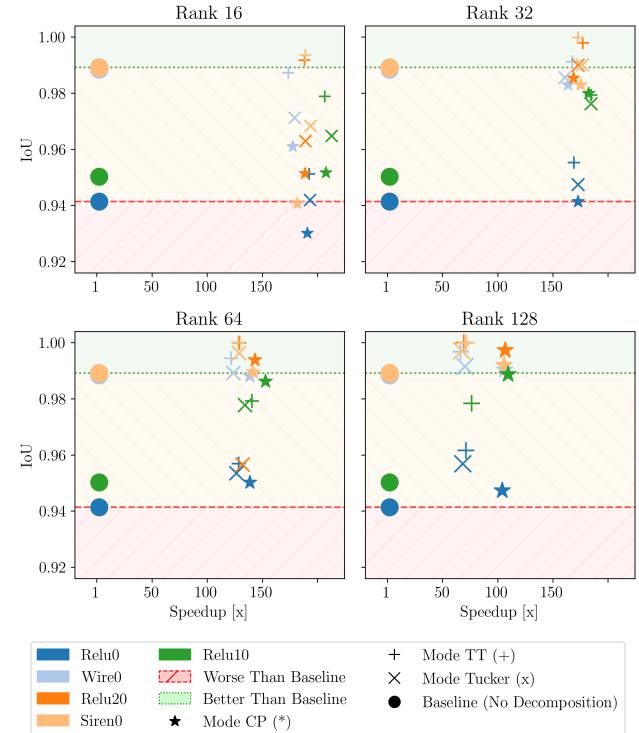
Figure 26. Quantitative results IoU vs. Speedup, for model Thai Statue. The general trend is that a larger rank leads to a better IoU. Tensor-Train models perform the best when compared to the other two models, and in several mo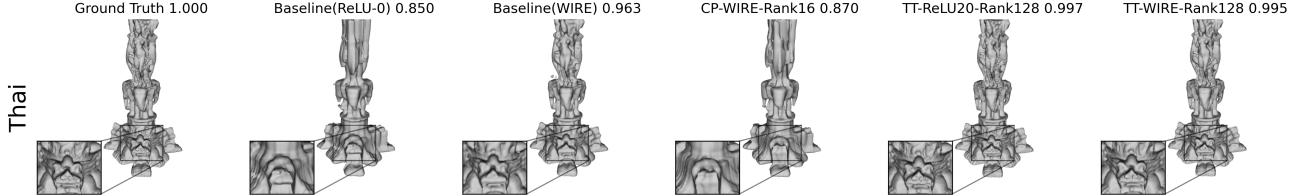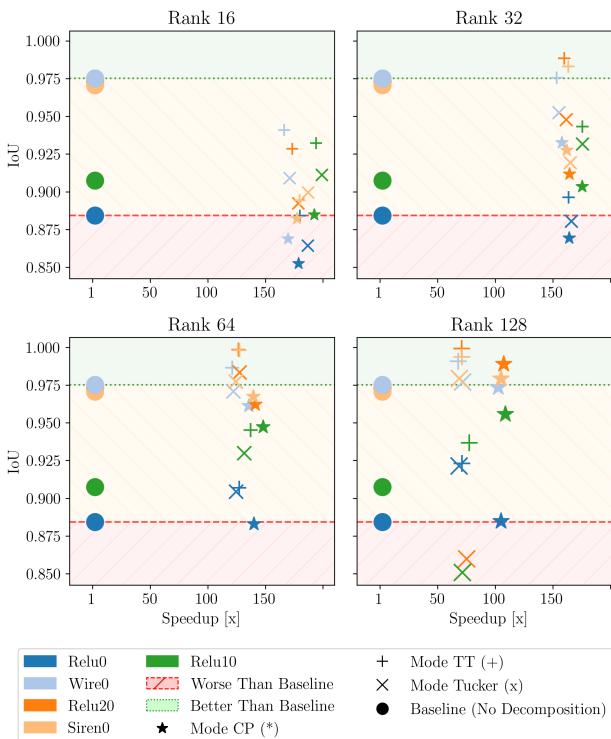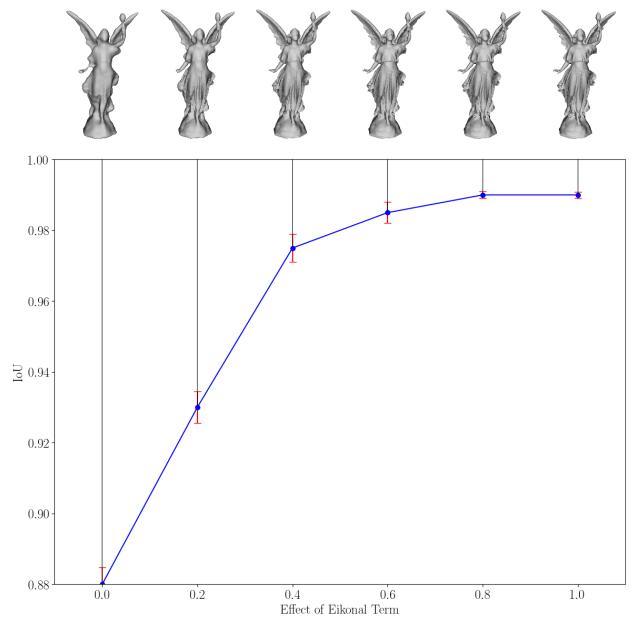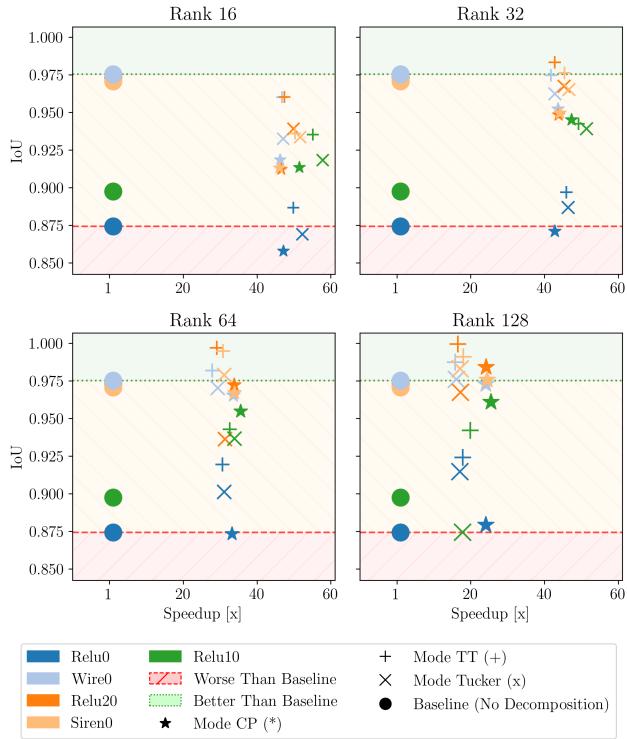dels, they are better than baselines. Even though all values seem closer, the visualizations shown in 23 show the qualitative differences.

Backend	Mode	Rank	IoU	L_2 Error	Time (s)
ReLU0	-	-	0.874 ± 0.007	0.229 ± 0.003	16148
ReLU10	-	-	0.898 ± 0.009	0.192 ± 0.004	16753
ReLU20	-	-	0.973 ± 0.006	0.163 ± 0.004	17023
SIREN	-	-	0.971 ± 0.005	0.126 ± 0.010	16438
WIRE	-	-	0.975 ± 0.009	0.062 ± 0.064	16557
	CP	16	0.858 ± 0.002	0.229 ± 0.002	342
	CP	32	0.871 ± 0.003	0.227 ± 0.001	377
	CP	64	0.873 ± 0.002	0.217 ± 0.004	486
	CP	128	0.879 ± 0.002	0.205 ± 0.005	670
	TT	16	0.887 ± 0.005	0.209 ± 0.006	324
ReLU0	TT	32	0.897 ± 0.002	0.187 ± 0.006	351
	TT	64	0.919 ± 0.007	0.179 ± 0.006	526
	TT	128	0.924 ± 0.001	0.167 ± 0.003	902
	Tucker	16	0.869 ± 0.006	0.219 ± 0.006	308
	Tucker	32	0.887 ± 0.004	0.212 ± 0.005	347
	Tucker	64	0.901 ± 0.004	0.196 ± 0.004	519
	Tucker	128	0.915 ± 0.005	0.186 ± 0.004	945
	CP	16	0.918 ± 0.008	0.227 ± 0.004	357
	CP	32	0.952 ± 0.003	0.176 ± 0.007	379
	CP	64	0.965 ± 0.001	0.133 ± 0.004	492
	CP	128	0.973 ± 0.001	0.107 ± 0.003	686
	TT	16	0.960 ± 0.003	0.156 ± 0.005	353
WIRE	TT	32	0.975 ± 0.001	0.097 ± 0.003	396
	TT	64	0.982 ± 0.000	0.076 ± 0.002	595
	TT	128	0.987 ± 0.001	0.069 ± 0.001	1047
	Tucker	16	0.933 ± 0.004	0.207 ± 0.006	351
	Tucker	32	0.962 ± 0.002	0.154 ± 0.004	387
	Tucker	64	0.970 ± 0.001	0.120 ± 0.003	565
	Tucker	128	0.976 ± 0.001	0.101 ± 0.003	1028
	CP	16	0.912 ± 0.010	0.217 ± 0.010	365
	CP	32	0.948 ± 0.002	0.170 ± 0.017	388
	CP	64	0.972 ± 0.002	0.130 ± 0.006	503
	CP	128	0.984 ± 0.001	0.089 ± 0.003	704
	TT	16	0.960 ± 0.005	0.165 ± 0.010	359
ReLU20	TT	32	0.983 ± 0.002	0.080 ± 0.005	397
	TT	64	0.997 ± 0.001	0.059 ± 0.003	585
	TT	128	0.999 ± 0.000	0.051 ± 0.001	1029
	Tucker	16	0.939 ± 0.005	0.195 ± 0.005	341
	Tucker	32	0.967 ± 0.016	0.144 ± 0.008	376
	Tucker	64	0.936 ± 0.102	0.142 ± 0.022	543
	Tucker	128	0.967 ± 0.129	0.109 ± 0.061	989
	CP	16	0.913 ± 0.007	0.241 ± 0.006	357
	CP	32	0.950 ± 0.004	0.192 ± 0.005	372
	CP	64	0.967 ± 0.001	0.148 ± 0.003	485
	CP	128	0.976 ± 0.001	0.122 ± 0.001	673
	TT	16	0.936 ± 0.004	0.145 ± 0.004	327
SIREN	TT	32	0.976 ± 0.002	0.075 ± 0.001	361
	TT	64	0.995 ± 0.000	0.060 ± 0.024	534
	TT	128	0.991 ± 0.008	0.097 ± 0.020	912
	Tucker	16	0.934 ± 0.017	0.195 ± 0.015	318
	Tucker	32	0.965 ± 0.004	0.144 ± 0.001	352
	Tucker	64	0.979 ± 0.007	0.102 ± 0.033	528
	Tucker	128	0.983 ± 0.493	0.143 ± 0.446	961
	CP	16	0.913 ± 0.009	0.236 ± 0.007	325
	CP	32	0.945 ± 0.007	0.201 ± 0.006	353
	CP	64	0.955 ± 0.001	0.162 ± 0.002	471
	CP	128	0.961 ± 0.001	0.138 ± 0.003	656
	TT	16	0.935 ± 0.002	0.186 ± 0.004	303
ReLU10	TT	32	0.942 ± 0.002	0.161 ± 0.002	339
	TT	64	0.943 ± 0.001	0.157 ± 0.003	513
	TT	128	0.942 ± 0.003	0.157 ± 0.003	840
	Tucker	16	0.918 ± 0.009	0.221 ± 0.005	290
	Tucker	32	0.939 ± 0.004	0.191 ± 0.003	326
	Tucker	64	0.937 ± 0.046	0.182 ± 0.110	494
	Tucker	128	0.874 ± 0.337	0.279 ± 0.304	943

Table 9. Results for Geometry encoding task: Here, we provide both IoU and L2 Error between the predicted SDFs and Ground truth for the Thai model, taken from [12]. A visualization of IoU values is provided in Figure 26 and the SDFs are visualized in Figure 23.

References - Supplementary

[Sup1] Junwoo Cho, Seungtae Nam, Hyunmo Yang, Seok-Bae Yun, Youngjoon Hong, and Eunbyung Park. Separable physics-informed neural networks. *Advances in Neural Information Processing Systems*, 2023.

[Sup2] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 303–312. ACM, 1996.

[Sup3] Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. Linear light source reflectometry. In *ACM SIGGRAPH 2003 Papers*, pages 749–758, New York, NY, USA, 2003. Association for Computing Machinery.

[Sup4] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of the 37th International Conference on Machine Learning*. JMLR.org, 2020.

[Sup5] Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[Sup6] Xinquan Huang and Tariq Alkhalifah. Efficient physics-informed neural networks using hash encoding. *Journal of Computational Physics*, 501:112760, 2024.

[Sup7] Chiyu "Max" Jiang, Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A. Tchelepi, Philip Marcus, Prabhat, and Anima Anandkumar. Meshfreeflownet: a physics-constrained deep continuous space-time super-resolution framework. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2020.

[Sup8] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[Sup9] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery.

[Sup10] Yisi Luo, Xile Zhao, Zhemin Li, Michael K Ng, and Deyu Meng. Low-rank tensor function representation for multi-dimensional data recovery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[Sup11] 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.

[Sup12] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41 (4):102:1–102:15, 2022.

[Sup13] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[Sup14] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, Long Beach, CA, USA, 2019. IEEE.

[Sup15] 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.

[Sup16] Pu Ren, Chengping Rao, Yang Liu, Zihan Ma, Qi Wang, Jian-Xun Wang, and Hao Sun. Physr: Physics-informed deep super-resolution for spatiotemporal data. *Journal of Computational Physics*, 492:112438, 2023.

[Sup17] 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.

[Sup18] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[Sup19] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[Sup20] Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[Sup21] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994, Orlando, FL, USA, July 24-29, 1994*, pages 311–318. ACM, 1994.

[Sup22] Maolin Wang, Yu Pan, Zenglin Xu, Xiangli Yang, Guangxi Li, and Andrzej Cichocki. Tensor networks meet neural networks: A survey and future perspectives. *CoRR*, abs/2302.09019, 2023.

[Sup23] Peng-Shuai Wang, Yang Liu, and Xin Tong. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Transactions on Graphics (SIGGRAPH)*, 41(4), 2022.