F-INR: Functional Tensor Decomposition for Implicit Neural Representations

Supplementary Material

Table of Contents

A. Introduction

This document provides comprehensive details, theoretical proofs, and extended experimental results that support our main paper, *"F-INR: A Functional Tensor Decomposition Framework for Implicit Neural Representations."* Our goal is to offer transparency into our methodology, reproducibility, and provide a deeper, more nuanced understanding of the F-INR paradigm and its performance characteristics.

- **Implementation and Architectural Details**: We provide detailed descriptions of our experimental setups, including specifics on the neural network architectures used for each backbone (e.g., ReLU [29], SIREN [40], WIRE [39]), our handling of Positional and Hash Encoding. This section also includes a discussion on our use of standard `PyTorch` [34] and `jax` [3] implementations versus highly optimized libraries like *tiny-cuda-nn*, clarifying why certain methods were chosen for specific tasks.
- **Theoretical Foundation**: We provide a formal proof that our functional tensor decomposition framework is a universal approximation. This establishes the theoretical soundness of our approach and its capacity to represent complex, high-dimensional functions.
- **Complete Experimental Results**: We present the full, unabridged quantitative and qualitative results for all tasks discussed in the main paper. This includes the comprehensive tables for image representation, and additional video representation, across all tested combinations, detailed metrics for Signed Distance Function (SDF) reconstruction, results for radiance fields, and the complete analysis of our physics-informed Navier-Stokes simulation. Please be aware that some of the tables are provided as **additional external HTML documents** due to

their large scales, as they would not have fit within this LaTeX document.

- **Ablation Studies**: We include the full suite of ablation studies that analyze the impact of each of F-INR's core components: the choice of **backend**, the decomposition **mode** (CP, TT, Tucker), and the various tensor **rank**. These studies offer critical insights into the speed-accuracy trade-offs, noise resistance, and the interplay between different design choices (first attempts at a *thumb-rule*) within our framework.

For organizational clarity, each major section of this document begins on a new page to prevent tables and figures from being awkwardly split. We believe this material provides the necessary detail to fully appreciate the robustness, flexibility, and performance of the F-INR framework.

B. Tensor Decomposition Modes and Backends

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

B.1. Tensor Decomposition 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 [19, 48, 53]. 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 [19, 48] for comprehensive information about tensor decompositions.

Let $\Phi \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ denote an N-mode tensor of order N, where I_n represents the dimensionality along the 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 1. 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 $\mathcal{O}(m^2 l n^3)$ [25].

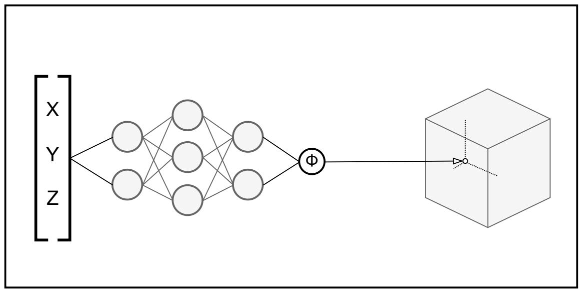


Figure 1. Classical INR of shape $X \times Y \times Z$. A single neural network predicts all the entries.

B.1.1. Canonical Polyadic (CP)

The Canonical Polyadic (CP) decomposition [15], also known as PARAFAC or CANDECOMP, approximates a tensor as a sum of rank-one tensors. Specifically, for an N-mode tensor $\Phi \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, the CP decomposition can be formulated as:

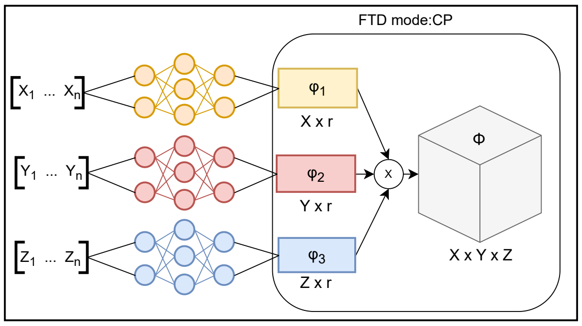


Figure 2. F-INR in CP mode for INR of shape $X \times Y \times Z$. Three individual neural networks predict the factor matrix of rank r for each corresponding dimension. The shape of the matrix/tensor is provided below.

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

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 2. For simplicity, we visualize three-dimensional INR of original 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 the forward pass involves 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 a factor matrix of shape $n \times r$ will be $\mathcal{O}(3m^2 l n r) = \mathcal{O}(m^2 l n r)$. 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 $\mathcal{O}(n^2 r^2)$. Combining them both will give the complexity of the forward pass of F-INR in CP mode as:

$$\mathcal{O}(m^2 l n r + n^2 r^2) \; . \tag{2}$$

B.1.2. Tensor Train (TT)

The Tensor Train (TT) [32] 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

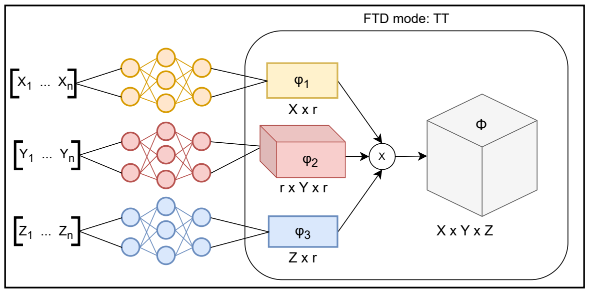


Figure 3. F-INR in TT mode for INR of shape $X \times Y \times Z$. Three individual neural networks predict the *cores* of rank r for each corresponding dimension. The shape of the matrix/tensor is provided below.

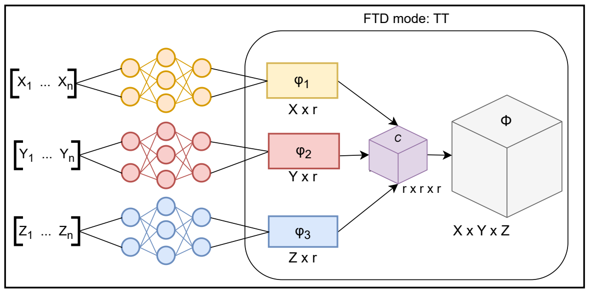


Figure 4. F-INR in Tucker mode for INR of shape $X \times Y \times Z$. C is the tucker-core tensor. Three individual neural networks predict the *factor matrices* of rank r for each corresponding dimension. The shape of the matrix/tensor is provided below.

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:

$$\Phi \approx \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \phi^{(1)}_{i_1,r_1} \phi^{(2)}_{i_2,r_1,r_2} \ldots \phi^{(N)}_{i_N,r_{N-1}} , \tag{3}$$

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 ourselves 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 $\mathcal{O}(m^2 lnr^2)$.

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 $\mathcal{O}(n^2 r^2)$. Combining these two, the total complexity of F-INR in TT mode will be:

$$\mathcal{O}(m^2 lnr^2 + n^2 r^2) . \tag{4}$$

B.1.3. Tucker (TU)

The Tucker decomposition (TU) [45] is a generalization of the CP 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:

$$\Phi \approx \mathcal{C} \times_1 \phi^{(1)} \times_2 \phi^{(2)} \ldots \times_N \phi^{(N)}, \tag{5}$$

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: $\mathcal{O}(m^2 lnr)$. 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 $\mathcal{O}(r)$ multiplications, therefore having a total complexity of $\mathcal{O}(n^3 r)$. Therefore, the forward pass complexity of an F-INR in Tucker mode is:

$$\mathcal{O}(m^2 lnr + n^3 r) . \tag{6}$$

B.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 Tanh and ReLU with Positional Encoding [29], SIREN [40], WIRE [39], FINER [23, 55], and Factor Fields [5]. 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.

B.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 [29, 41]. The inputs ν are passed through the mapping

$$\gamma(\nu) = [cos(2\pi B\nu), sin(2\pi B\nu)]^T , \tag{7}$$

where B is a random Gaussian matrix whose entries are randomly drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$, 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. Please note that we conducted experiments with σ of 0, 10, and 100. Hence, a value of zero would denote the usage of no input embedding.

B.2.2. Tanh MLP

This model is a standard MLP where each ReLU activation is replaced with Tanh. Due to its saturating nature, the Tanh activation is prone to the vanishing gradient problem, a well-known issue that often leads to training instability and lower representational fidelity compared to non-saturating alternatives. We follow the same kind of embedding experiments as for the other models.

B.2.3. SIREN

Introduced by [40], 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 [40].

Finally, SIREN has the first layer of activations as $sin(\omega_0 w_i x + b_i)$, where $\omega_0 = 30$, to ensure periodicity[40]. In our implementations, we use the same prescribed ω and initialization of weights.

B.2.4. FINER

Introduced by [23, 55], FINER follows a similar idea to SIREN. While SIREN uses fixed sinusoidal activation functions throughout the training, FINER ensures that he wave can be modulated, resulting in the potential for more adapted results.

B.2.5. WIRE

The Gabor wavelet activation function was introduced in [39] 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 [29] and SIREN [40]. The complex form of a Gabor wavelet is written as

$$\psi(x; \omega, s) = e^{j\omega x} e^{-|sx|^2} \,, \tag{8}$$

where ω is the parameter controlling the frequency and s controls 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.

B.2.6. Factor Fields

Factor Fields introduces a unified and modular framework for representing continuous signals like images and 3D scenes. The core idea is to decompose a signal's neural representation into a product of multiple distinct factors. Each factor consists of a coordinate transformation, which pre-processes input coordinates (e.g., using periodic functions or hash grids), and a field representation that maps these transformed coordinates to features (e.g., using voxel grids or a small MLP). The feature outputs from all factors are then multiplied element-wise and passed through a final projection network to produce the output signal value. This flexible design allows the framework to express many prominent neural representations like NeRF [29], TensoRF [4], and Instant-NGP [31] as special cases, thereby offering a general language for designing and understanding both existing and novel neural fields.

B.2.7. Hash Encoding as Embedding

Hash encoding was introduced in Instant Neural Graphics Primitives [31] 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 [31], 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:

$$R_l = R_0 \cdot \alpha^l, \quad l = 0, 1, \ldots, L-1 \,, \tag{9}$$

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:

$$\mathbf{i} \cdot l = \lfloor R_l \mathbf{x} \rfloor \,. \tag{10}$$

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

$$h(\mathbf{i}l) = \left(\sum j = 1^d il, jp_j\right) \mod T \ , \tag{11}$$

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:

$$\mathbf{f}(\mathbf{x}) = \sum c \in 0, 1^d w_c \cdot \mathbf{f}_h(h(\mathbf{i}_l + c)) \ , \tag{12}$$

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:

$$\mathbf{f}(\mathbf{x}) = \bigoplus_{l=0}^{L-1} \mathbf{f}_l \ . \tag{13}$$

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 [17, 31]. There are effective ways to implement hash encoding for PDEs and Eikonal geometry solving [17], but we leave this extension to the future.

B.3. Model Implementations in tiny-cuda-nn

To provide a rigorous comparison against the state-of-the-art, we benchmark against optimized backbones where applicable, such as those implemented in the *tiny-cuda-nn* framework [30] where applicable. These implementations offer significant speedups through custom CUDA kernels and drastically reduce the run time of the problem. They slightly differ from the original implementations and might yield different results, as often biases are not used in the MLP layers. Nevertheless, to show that our speed-up results are orthogonal to network design, we include these experiments.

However, integrating our native PyTorch F-INR framework with these specialized kernels is a non-trivial engineering task that we consider out of scope for this work. Furthermore, these kernels are fundamentally incompatible with our PDE-constrained experiments (e.g., SDFs with Eikonal regularization and Navier-Stokes super-resolution). They are not designed to provide stable gradients with respect to input coordinates that are essential for computing the physics-informed loss terms. Therefore, all experiments requiring such gradients are performed using standard `PyTorch` [34] and `jax` [3] backbones for all methods to ensure a fair and consistent evaluation. Either based on the provided code by the authors or the given formulas.

B.4. Notes on SOTA Methods

To ensure a fair and rigorous comparison against state-of-the-art methods, we utilized publicly available codebases whenever possible. In some cases, minor modifications were necessary to integrate these methods into our standardized evaluation protocol (e.g., adapting them to our data formats or PDE-constrained objectives) or newer software requirements (mostly `cuda` or `PyTorch` upgrades). This section documents all such changes to provide full transparency and ensure the reproducibility of both our results and our implementation changes of prior work. Therefore, if we observe slight differences from the original numbers reported in the respective papers, we assume this stems from the outlined changes in this section.

B.4.1. CoordX

CoordX [22] proposes an axis-aligned split-MLP architecture where coordinate-specific branches are progressively fused in deeper layers. In contrast, our F-INR framework enforces strict functional separability by keeping the univariate networks entirely distinct through all layers.

To ensure a fair and reproducible comparison, we addressed outdated implementation challenges. The original CoordX implementation relies on an outdated software stack (PyTorch 1.3, CUDA 10.4, `torchmeta`) incompatible with modern hardware. We, therefore, ported the official codebase to a current environment (PyTorch 2.7.1, CUDA 12.6) and replaced the deprecated `torchmeta` library with standard PyTorch equivalent procedures. To validate our port, we verified that it reproduced the results of the original CPU implementation on the provided *cameraman* image benchmark, achieving a negligible error margin ($\epsilon < 0.02$).

Despite these efforts, the original code's modules for SDF and NeRF tasks were not functional, preventing a direct port. To maintain a rigorous comparison, we therefore benchmark CoordX only on the image representation task, where its functionality could be reliably verified.

B.4.2. NeuRBF

For our comparisons involving NeuRBF [7], we adopt the official code and configurations provided by the authors. To ensure a fair and direct runtime comparison on our target hardware (NVIDIA RTX 3090), we made a single necessary adjustment to the default configuration to fit the model within the available GPU memory. All other aspects of the original implementation remain unchanged. The experiments were executed within our standardized environment (PyTorch 2.7.1, CUDA 12.6) to maintain consistency across all benchmarks.

This comparison is particularly salient on the Armadillo model, for which NeuRBF was originally designed. For this task, NeuRBF created an 80MB ply model to achieve its reported quality. In contrast, our F-INR framework reaches comparable fidelity with a ply model size of just 4MB, demonstrating a 20-fold reduction.

B.4.3. Factor Fields

Factor Fields [5] is a key related work that also explores a modular, factorization-based paradigm for INRs. For our comparison, we utilize the official codebase provided by the authors. To ensure a fair benchmark, we made minimal and necessary adjustments to its default configuration, primarily to align batch sizes with our hardware's memory constraints and to integrate it into our standardized evaluation environment (PyTorch 2.7.1, CUDA 12.6). Our goal in this process was to evaluate Factor Fields under our unified experimental protocol while preserving the core architectural and conceptual principles of the original work.

B.4.4. InstantNGP

Instant-NGP [31] sets a high bar for performance due to its hash-grid encoding and highly optimized CUDA kernels [44]. To ensure a fair comparison, our methodology distinguishes between two use cases: for baseline results, we use the official CUDA implementation. To integrate hash encoding as a component within our F-INR framework, we adapt pure PyTorch re-implementations [2, 42, 43]. This allows us to isolate the algorithmic gains of our functional decomposition from hardware-level optimizations.

C. Universal Approximation Theorem for Separable Neural Functions

We provide the theoretical foundation for our framework, showing that a rank-r functional tensor decomposition using neural networks is a universal approximation, adapted from [47]. This demonstrates that our method for mitigating the curse of dimensionality is theoretically sound.

Theorem 1 Let $f : \mathbb{K} \to \mathbb{R}$ be a continuous function on a compact set $\mathbb{K} \subset \mathbb{R}^d$. For any $\epsilon > 0$, there exists a rank-r decomposition of univariate neural networks, $\hat{f}$, such that $\|f - \hat{f}\| < \epsilon$.

Proof. It is known that any continuous multivariate function can be arbitrarily well-approximated by a sum of separable functions [8, 14, 35, 47]:

$$f(x_1, \ldots, x_d) \approx \sum_{j=1}^{r} \bigotimes_{i=1}^{d} g_i^j(x_i). \tag{14}$$

For clarity, we prove the universal approximation property for a single rank-1 term and note that it extends to the rank-r sum. Let a rank-1 function be $f(x) = \bigotimes_{i=1}^{d} g_i(x_i)$. We approximate f with $\hat{f}(x) = \bigotimes_{i=1}^{d} \widehat{g}_i(x_i, \theta_i)$, where each $\hat{g}_i$ is a neural network with parameters θ_i.

By the classical Universal Approximation Theorem [10, 16], for each continuous univariate function g_i, there exists a network $\hat{g}_i$ such that for any $\epsilon_i > 0$:

$$\|g_i(x_i) - \widehat{g}_i(x_i, \theta_i)\| < \epsilon_i, \quad \forall x_i \in \mathbb{K}_i, \tag{15}$$

where $\mathbb{K}_i$ is a compact subset of $\mathbb{R}$. The total approximation error is given by:

$$\|f - \hat{f}\| = \left\| \bigotimes_{i=1}^{d} g_i(x_i) - \bigotimes_{i=1}^{d} \widehat{g}_i(x_i, \theta_i) \right\|. \tag{16}$$

Using the identity for the difference of products [8, 14], the term inside the norm can be expanded as:

$$\bigotimes_{i=1}^{d} g_i - \bigotimes_{i=1}^{d} \widehat{g}_i = \sum_{j=1}^{d} \left(\bigotimes_{k=1}^{j-1} \widehat{g}_k \right) \otimes (g_j - \widehat{g}_j) \otimes \left(\bigotimes_{l=j+1}^{d} g_l \right). \tag{17}$$

By applying the triangle inequality and assuming a sub-multiplicative cross-norm (i.e., $\|A \otimes B\| \leq \|A\|\|B\|$) [14, 38], we can bound the total error:

$$\|f - \hat{f}\| \leq \sum_{j=1}^{d} \left\| \left(\bigotimes_{k=1}^{j-1} \widehat{g}_k \right) \otimes (g_j - \widehat{g}_j) \otimes \left(\bigotimes_{l=j+1}^{d} g_l \right) \right\| \tag{18}$$

$$\leq \sum_{j=1}^{d} \left(\prod_{k=1}^{j-1} \|\widehat{g}_k\| \right) \|g_j - \widehat{g}_j\| \left(\prod_{l=j+1}^{d} \|g_l\| \right). \tag{19}$$

Substituting the bound from Equation (15), we get:

$$\|f - \hat{f}\| \leq \sum_{j=1}^{d} \left(\prod_{k=1}^{j-1} \|\widehat{g}_k\| \right) \epsilon_j \left(\prod_{l=j+1}^{d} \|g_l\| \right). \tag{20}$$

Since all functions are continuous on a compact set, their norms ($\|g_l\|$ and $\|\hat{g}_k\|$) are bounded. By choosing each ϵ_j to be sufficiently small, the total error $\|f - \hat{f}\|$ can be made less than any desired $\epsilon > 0$. This completes the proof.

This result demonstrates that our F-INR framework, which uses neural networks as learnable factors in a tensor decomposition, retains the universal approximation property while breaking the curse of dimensionality.

D. Image Representation Details

D.1. MLP Architectrue

For our 2D image representation experiments, the F-INR framework factorizes the implicit function into two univariate MLPs, one for each spatial coordinate (x and y). The outputs of these networks are combined via an outer product to reconstruct the final image. The RGB channels are treated as the output dimension of the final projection layer, not as factorizable input dimensions.

To ensure a fair comparison between our method and standard INRs, we carefully control the model capacity:

- **F-INR Backbones**: Each of the two univariate MLPs consists of four hidden layers with 256 neurons referring to the original works.
- **Baseline Monolithic INRs**: For a comparable parameter count, each baseline model uses a single, monolithic MLP with several hidden layers with 256 neurons referring to the original works.

We specifically evaluate three configurations to analyze the effect of Positional Encoding (PE) [29] and Hash Encoding (HE) [31], here nomination is explained based on the ReLU model but holds for all other tested backlines:

- **ReLU+PE000**: A ReLU MLP without Positional Encoding.
- **ReLU+PE010**: A ReLU MLP with PE using a frequency scale of $\sigma = 10$.
- **ReLU+PE100**: A ReLU MLP with PE using a frequency scale of $\sigma = 100$.
- **ReLU+HE**: A ReLU MLP with HE using the default configuration [31].

This setup allows for a direct and principled comparison of our factorized approach against standard monolithic architectures.

Each model is trained for 50,000 iterations using the Adam optimizer with default parameters to learn the *parrot* image of DIV2K [1].

D.2. Quantitative Results

Here, we provide the PSNRs for all the models for varying ranks and the baselines in Figure 5 and the external HTML table of all tested combinations.

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 having a strong

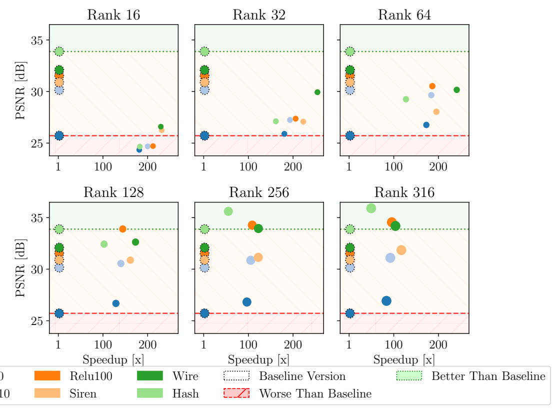


Figure 5. We display how F-INRs and baseline models compare regarding speed and PSNR value for the image encoding task. The circle radius describes the compression rate compared to the baseline. Models with a rank of 16 are similar to the worst baseline method, indicating that such models cannot fully 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 strong speedup. The performance is shown by hash encoding, followed by ReLU100 and WIRE models.

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.

D.3. Qualitative Results

More qualitative examples for all 128 combinations are provided in the accompanying zip folder for both the *parrot* and *butterfly* images. For the original image we refer to the DIV2K [1] validation dataset.

D.4. Ablations: 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 [39]. 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.

We use a *butterfly* image from DIV2K [1] for the super-resolution task and induce a $4\times$ sparsity by sub-sampling pixels before training, following the same setup as in [39]. The original dimensions of the image are 1356×2040, while the downsampled image is: 339×510. The F-INR models are trained on this sparse representation for back-

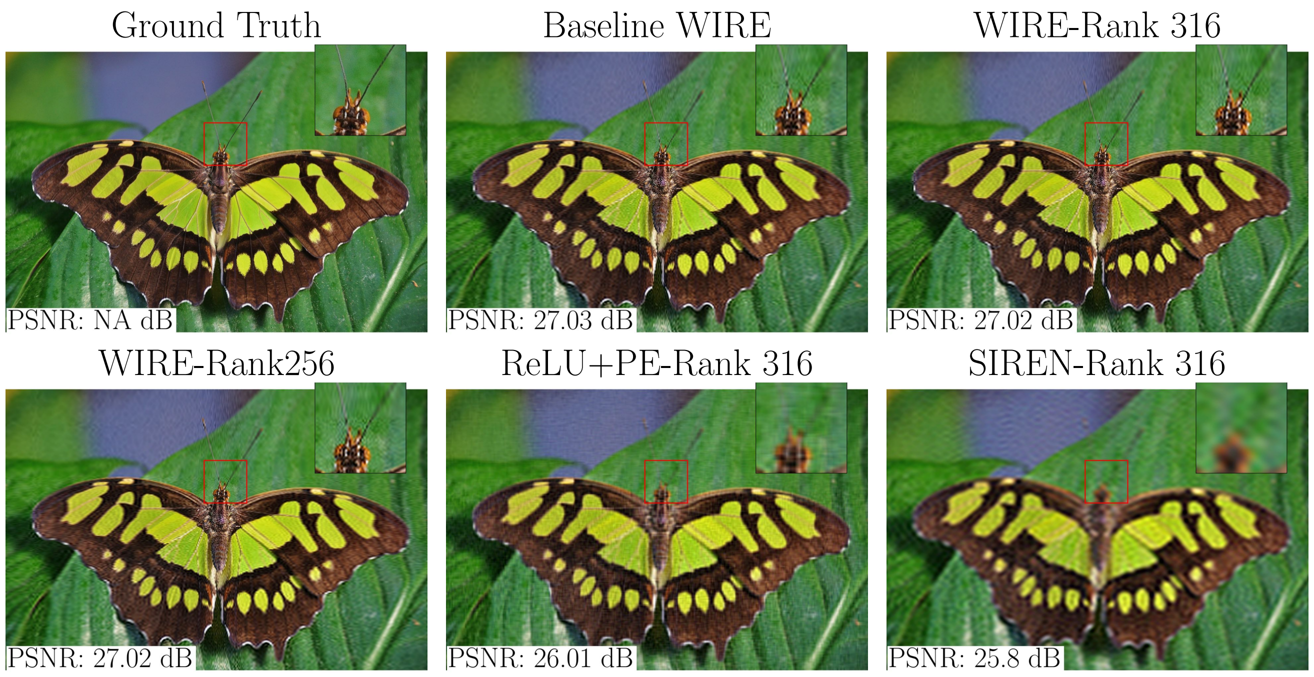


Figure 6. Single Image Super-Resolution using F-INR. Here, we observe that qualitatively F-INR performs the same as baseline WIRE. Therefore, F-INR preserves the inherent quality salient to WIRE architecture, forming better priors. Image taken from [39].

ends 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. 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 [17, 31, 41].

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 7 and Figure 6.

For the denoising task, we introduce shot Gaussian noise to the original image of the *parrot* image from DIV2K [1] from (1356 $\times$ 2040) and train the F-INR model to reconstruct the clean version, following [39]. The performance is

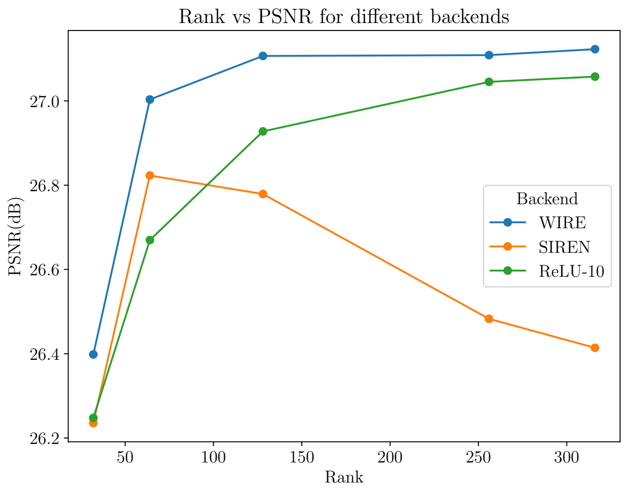


Figure 7. Evolution of Rank vs. PSNR for different backends for the Single Image Super-resolution task. We observe that WIRE performs best and is robust in ranking. SIREN degrades performance with increased rank.

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 9, and the visualizations are provided in Figure 8.

Unlike super-resolution, where rank is negligible, rank

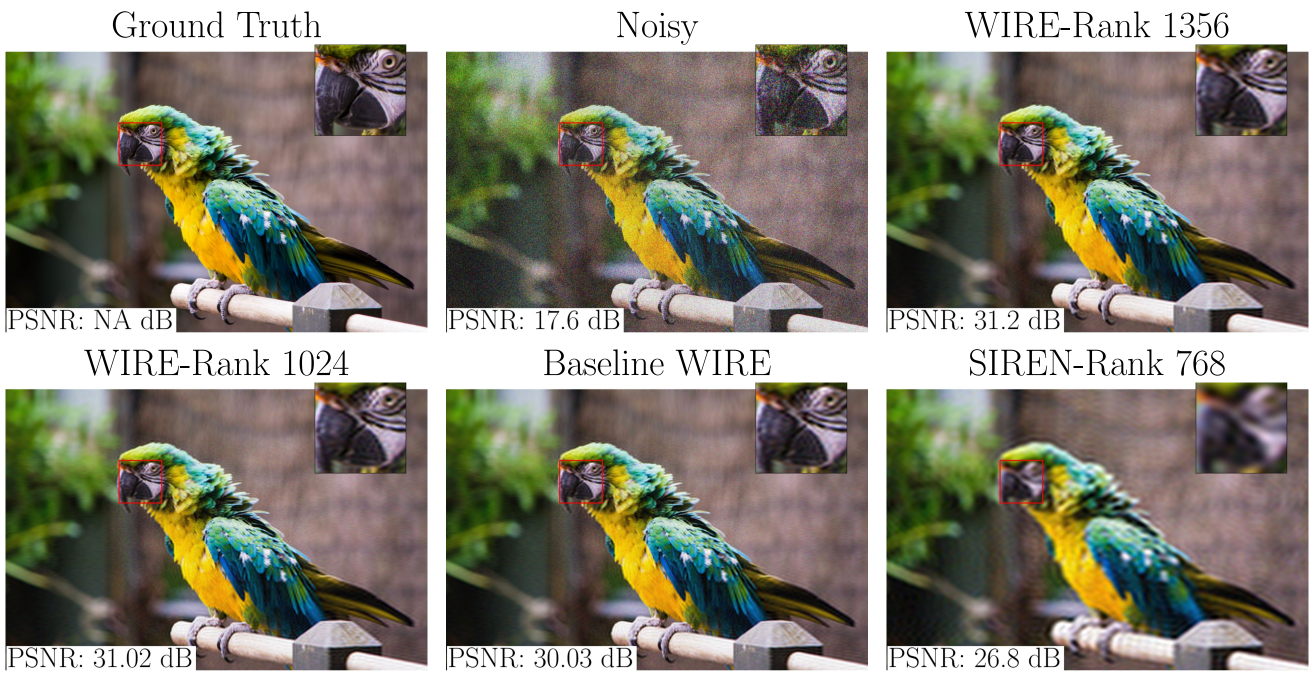


Figure 8. Visualizations for the denoising task of a parrot. Image taken from [39]. F-INR with WIRE backend retains the robustness of WIRE, emphasizing the influence of backend for the task-specific performance of F-INRs.

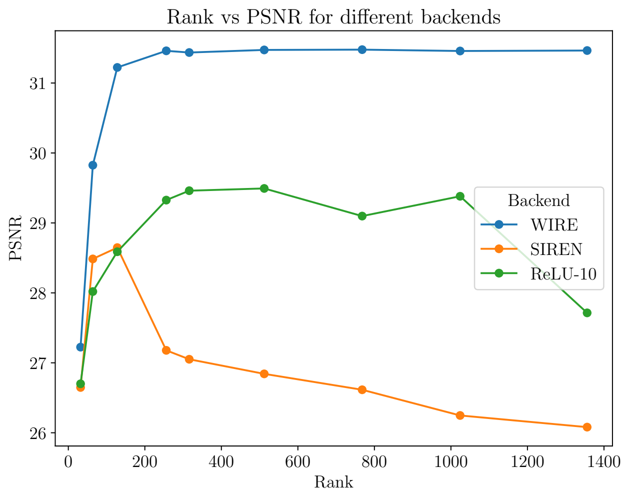


Figure 9. Rank vs PSNR for the denoising task for various backends of F-INR. We observe that the wire is robust and almost rank-independent. In contrast, SIREN and ReLU, with PE, have a deterioration with increasing rank, which might be due to overfitting of the noisy image.

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. Video Representation Extension

We extend the 2D image representation framework to videos by incorporating a third factor for the temporal dimension. The video signal is thus modeled as the tensor product of three univariate neural networks, each corresponding to a single spatio-temporal axis (x, y, t):

$$\phi_1(x; \theta_1) \otimes \phi_2(y; \theta_2) \otimes \phi_3(t; \theta_3) \mapsto (r, g, b). \tag{21}$$

Here, $\phi_3(t; \theta_3)$ is the network modeling the temporal coordinate, where t corresponds to the frame index. The model architecture for each factor follows the same principles described for the image representation task.

E.1. Model Architectures

For this benchmark, we use a 300-frame video of a person at 256×256 resolution [54], visualized in Figure 11. Based on our 2D results, we evaluate the **ReLU+PE100**, **SIREN**, and **WIRE** backbones across all tensor decomposition modes. To ensure a fair comparison, each of F-INR's three univariate networks uses 3 hidden layers with 256 neurons, while monolithic baselines use a single 4-layer MLP with 256 neurons. All models are trained for 50k epochs using the Adam optimizer with the largest batch size that fits into GPU memory.

A key advantage of our framework is its modularity, which allows for assigning different encoding strategies to different input dimensions. This adaptability is showcased in our video experiments, where the best-performing configuration combines **hash encoding** for the spatial dimensions (ϕ_1, ϕ_2) with **Fourier features** for the temporal dimension (ϕ_3). This hybrid approach leverages the distinct strengths of each encoding type for their respective domains. This flexibility also extends to the decomposition modes; for the Tensor-Train (TT) decomposition, we align its 3-way tensor core with the temporal dimension, which empirically provided the most stable training.

For clarity, we note that our comparison does not include dedicated neural video compression algorithms like NeV-D or COIN++ [6, 11, 26]. These methods operate on a different paradigm of frame-wise feature learning followed by quantization, which is distinct from our goal of learning a single, continuous spatio-temporal representation.

E.2. Results

The open source video of the face of the girl, the results for various ranks and backends are given in Figure 10 and Table 1. 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 the 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 as external files.

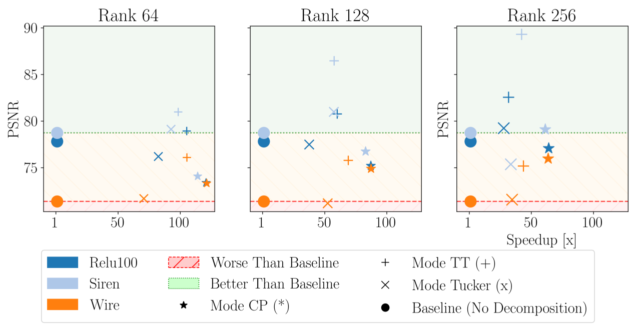


Figure 10. We visualize PSNR values given in the Table 1. Here, different markers are assigned for each mode, and different colors are assigned for each backend. We only use rank ≥ 64 and three backends, WIRE, SIREN, and ReLU100. Many configurations surpass the baseline implementation (above green), and of all the modes, TT has consistently the best results.

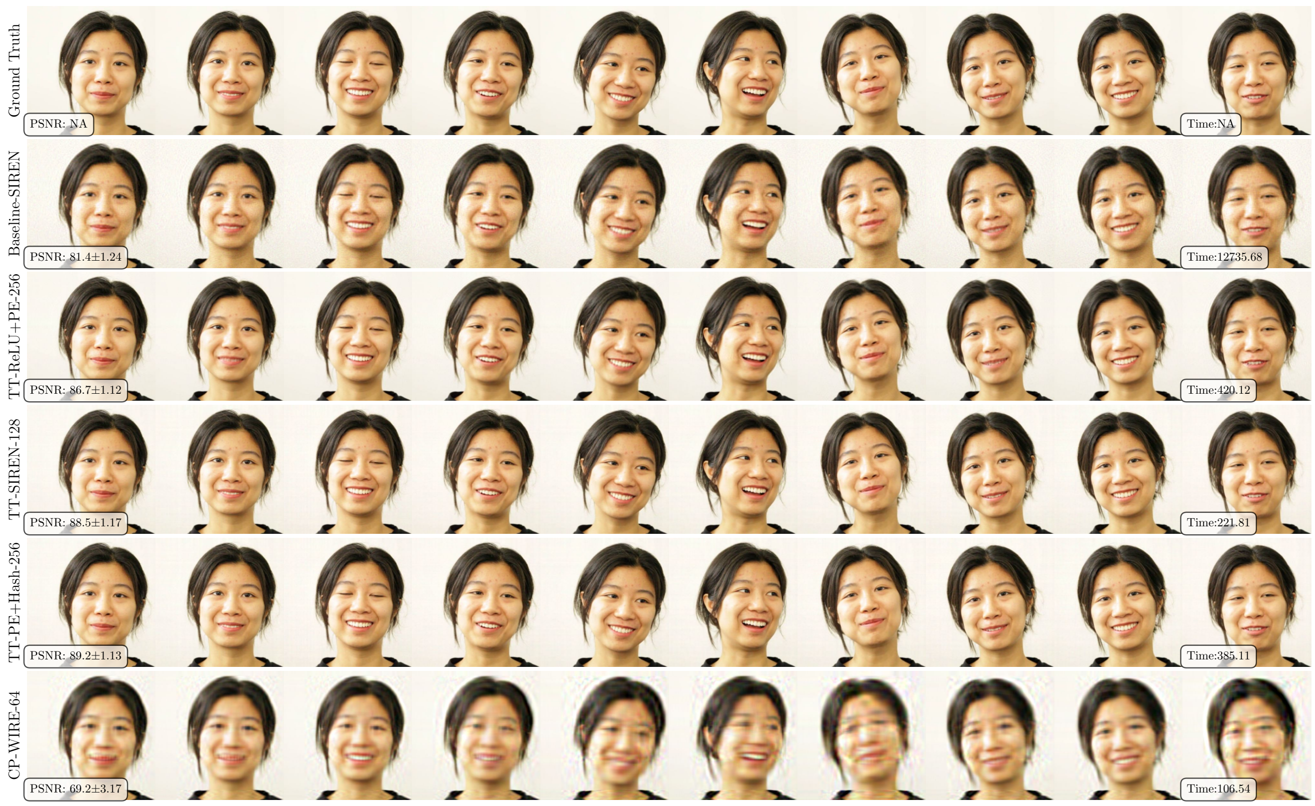


Figure 11. **Encoding video with nuanced facial features [54] (publicly available) using F-INR**: The mean PSNR (dB) and model are shown in the first column; training time is in the last. SIREN [40] and ReLU [27] with positional encoding [29] outperform their baseline, capturing facial details and maintaining temporal consistency, while WIRE [39] performs worse. The best performance (fifth row) was achieved using a combination of hash [31] and positional encoding for spatial and frame dimensions, respectively, highlighting the modularity of F-INR. Additional results are in the supplementary.

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
ReLU100	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
	TT	128	80.74 ± 0.17	0.93	221
	TT	**256**	**82.55 ± 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
SIREN	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
	TT	**128**	**86.46 ± 0.19**	**0.97**	**220**
	TT	**256**	**88.28 ± 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
WIRE	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
	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**	**89.2 ± 0.13**	**0.98**	**385**
ReLU100 + Hash	Tucker	256	80.35 ± 2.63	0.74	427

Table 1. 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 10.

F. Geometry Learning via Signed Distance Functions

For our geometry encoding task, we adopt a physics-informed approach, learning Signed Distance Functions (SDFs) by enforcing the Eikonal equation as a PDE regularization. This principled objective ensures the learned function represents a valid metric space, a critical property often overlooked by simpler data-fitting criteria. Consequently, this rigorous paradigm precludes methods whose architectures are not fully differentiable with respect to their input coordinates and thus cannot support the required gradient-based regularization.

Our experimental protocol is as follows. We evaluate four intricate models from the Stanford 3D Scan Repository [9, 12, 21, 46]: the *Armadillo*, *Dragon*, *Lucy*, and *Thai statues*. To generate the ground-truth SDF data, we convert the original .obj meshes using the `mesh2sdf` library [49]. The final, continuous SDFs learned by our models are visualized by extracting the zero-level set using the standard Marching Cubes algorithm [24], following common practice [5].

F.1. Architecture and Training

For the SDF task, our F-INR framework factorizes the 3D function into three univariate neural networks, one for each spatial axis. We evaluate all three tensor decomposition modes (CP, TT, Tucker). To ensure a fair comparison of model capacity, we define our architectures as follows:

- F-INR Models: Use three separate 4-layer MLPs, each with 256 neurons.
- Monolithic INR Baselines: Use a single 5-layer MLP with 256 neurons.

We also benchmark against state-of-the-art methods like DeepSDF [33] and IGR [13]. While these methods are typically trained on point clouds, we have adapted their frameworks to train directly on our voxel grid data. This ensures all methods are evaluated on identical input data and under the same Eikonal PDE objective. All models are trained for 50k iterations using the Adam optimizer. The qualitative visualizations presented in the paper consistently showcase the best-performing model configuration for each approach.

F.2. Ablations

We use IoU and MSE metrics to quantify the predicted values against the ground truth. All the results for *Lucy* are provided in Figure 22 and Table 2. For *Armadillo*, Figure 13 and the tables are provided as an additional HTML file. Finally, for the Thai statue, the results are provided in Table 3 and Figure 24 and also visualized in Figure 21. All the results show that F-INR models achieve 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 with

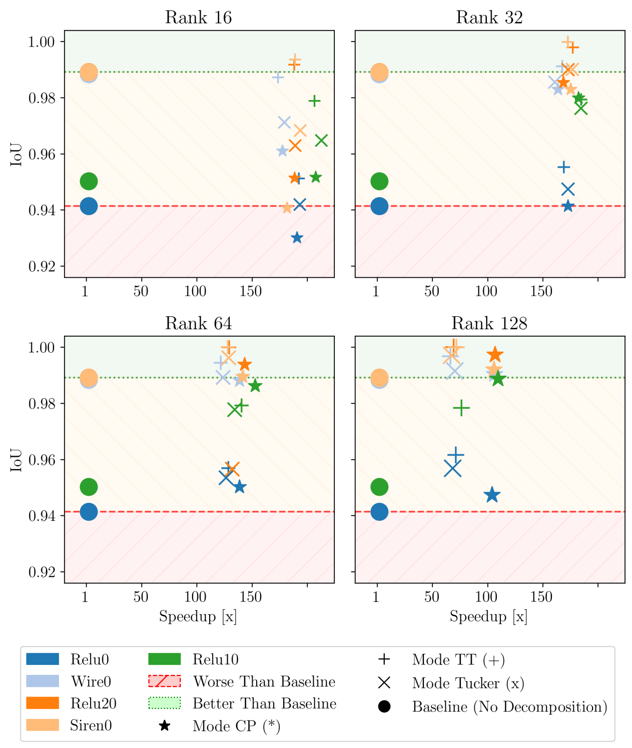


Figure 12. Quantitative results IoU vs Speedup, for model Armadillo. The general trend is that a higher rank leads to a better IoU. Tensor-Train [32] models perform the best in several models compared to the other two models.

respect to other loss terms for the best performing F-INR, i.e., mode TT with rank 128 and with ReLU+P010 backend, and the results are given in Figure 23. We observe that the Eikonal PDE significantly affects finding a better solution.

F.3. Qualiative Results

We provide more examples here in the document of the *Armadillo* model as well as **all** combinations as additional external image files. The best `.ply` models are also provided in the accompanying zip folder. Please note that all exported models are normalized in size for easier rendering. Please note that for *Lucy* and *Thai statuette*, a simpler rendering pipeline was employed during the ablation studies without a texture material, resulting in a flatter visual appearance.

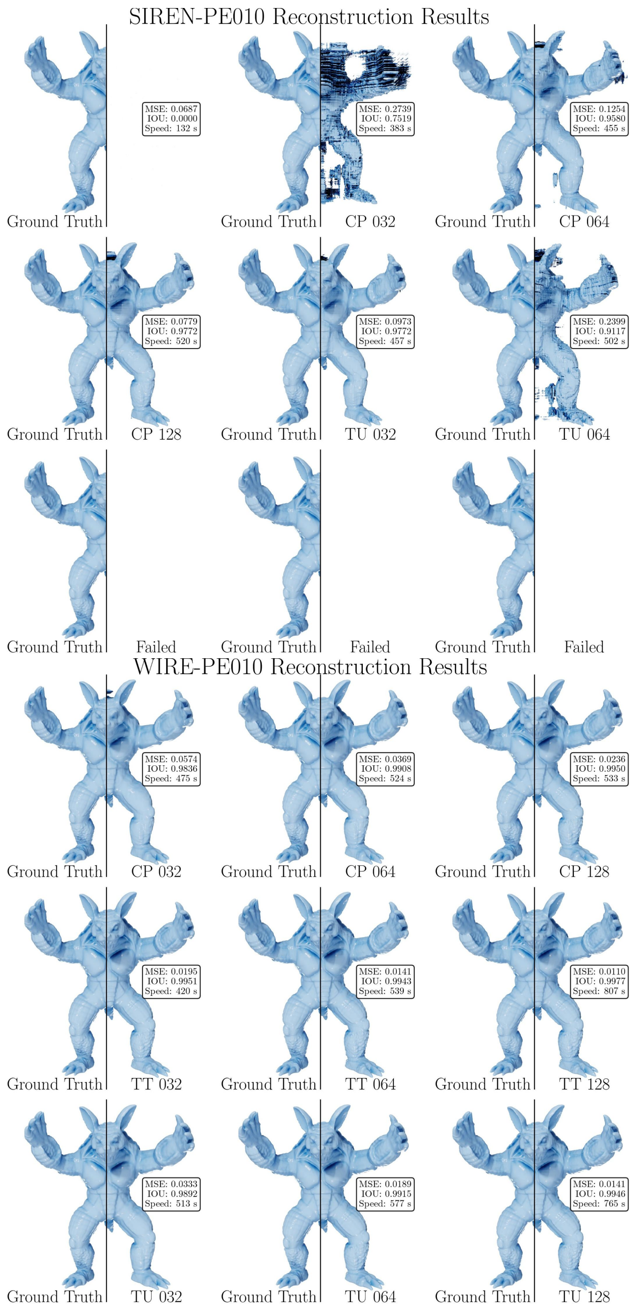

SIREN-PE010 Reconstruction Results
MSE: 0.0687 IOU: 0.0000 Speed: 132 s
MSE: 0.2739 IOU: 0.7519 Speed: 383 s
MSE: 0.1254 IOU: 0.9580 Speed: 455 s
Ground Truth
CP 032
CP 064
MSE: 0.0779 IOU: 0.9772 Speed: 520 s
MSE: 0.0973 IOU: 0.9772 Speed: 457 s
MSE: 0.2399 IOU: 0.9117 Speed: 502 s
CP 128
TU 032
TU 064
Failed
WIRE-PE010 Reconstruction Results
MSE: 0.0574 IOU: 0.9836 Speed: 475 s
MSE: 0.0369 IOU: 0.9908 Speed: 524 s
MSE: 0.0236 IOU: 0.9950 Speed: 533 s
MSE: 0.0195 IOU: 0.9951 Speed: 420 s
MSE: 0.0141 IOU: 0.9943 Speed: 539 s
MSE: 0.0110 IOU: 0.9977 Speed: 807 s
TT 032
TT 064
TT 128
MSE: 0.0333 IOU: 0.9892 Speed: 513 s
MSE: 0.0189 IOU: 0.9915 Speed: 577 s
MSE: 0.0141 IOU: 0.9946 Speed: 765 s
TU 128

Figure 13. **Qualitative Armadillo results.** This figure illustrates how the choice of INR backbone and tensor rank critically affects reconstruction quality. While certain configurations, such as a low-rank (CP-32) SIREN model, can fail to converge and produce severe artifacts, others demonstrate high robustness. Notably, the WIRE backbone consistently yields high-fidelity reconstructions across all tested ranks, highlighting the importance of the interplay between a backbone's inductive bias and the capacity afforded by the tensor decomposition.

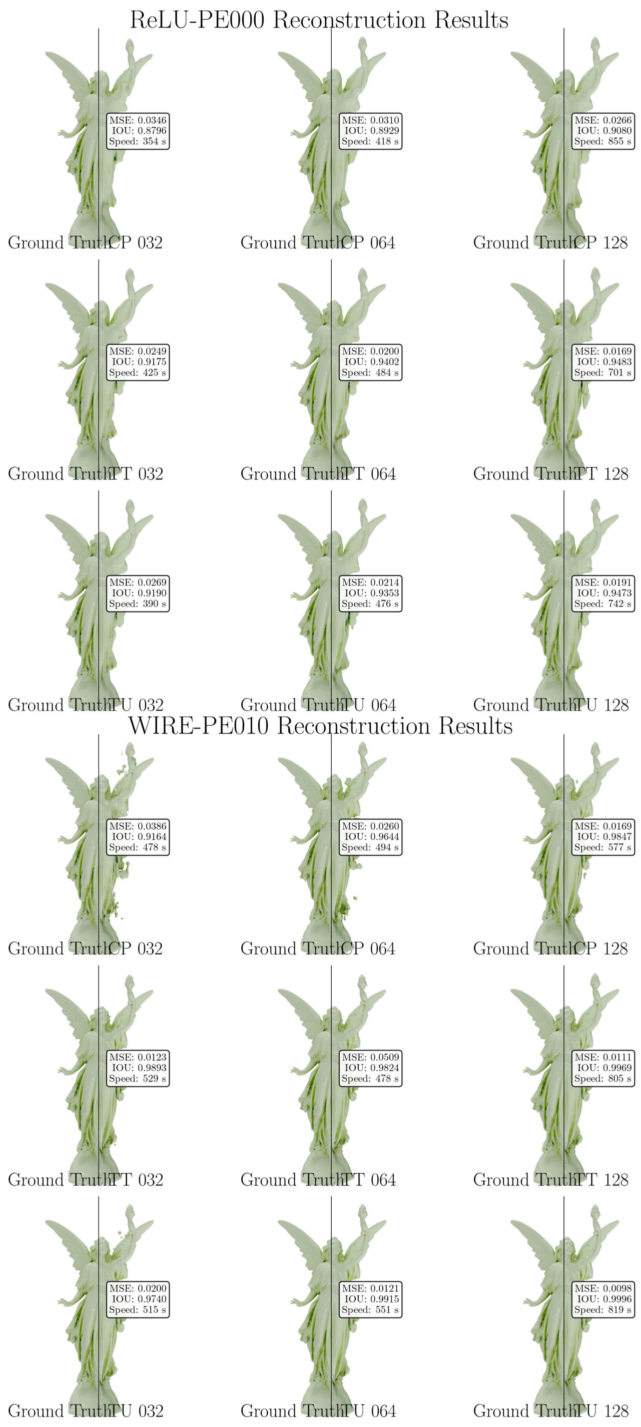
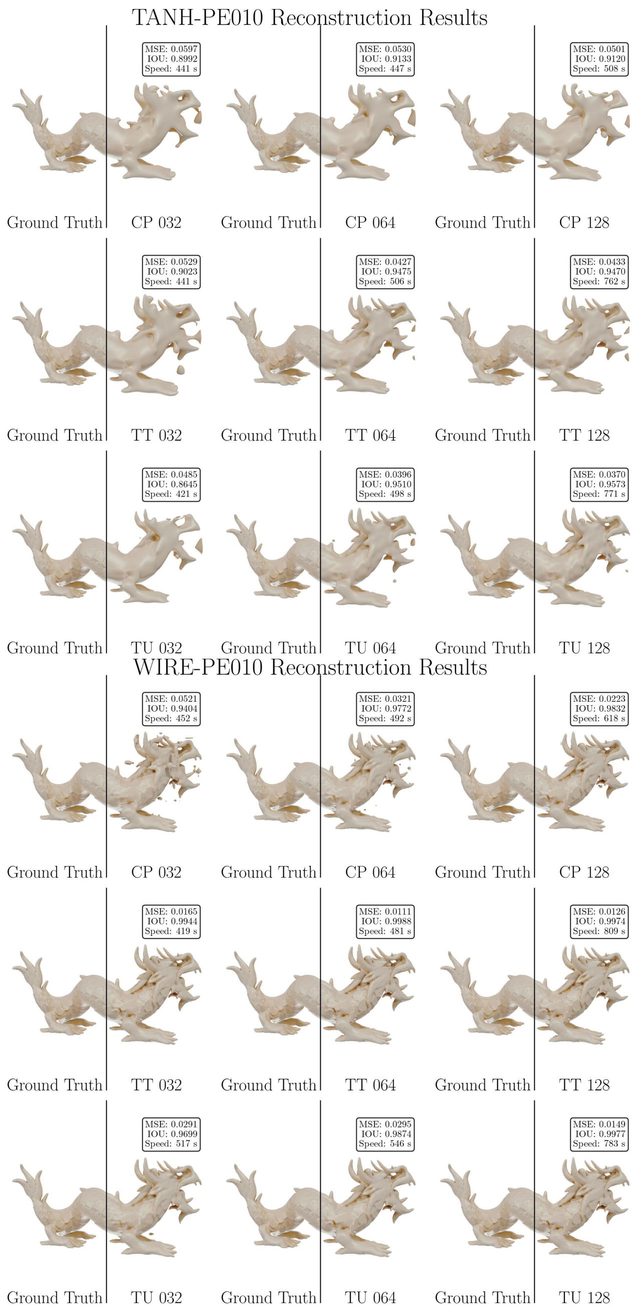
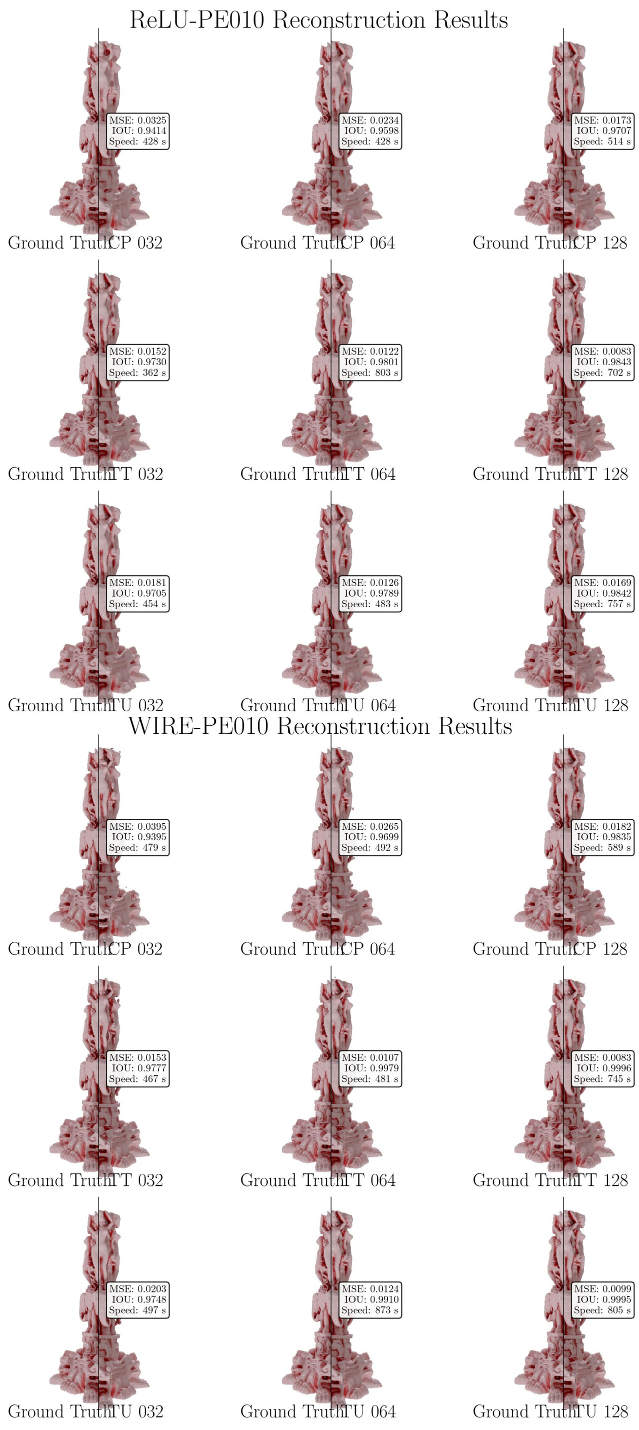

ReLU-PE000 Reconstruction Results
MSE: 0.0346 IOU: 0.8796 Speed: 354 s
MSE: 0.0310 IOU: 0.8929 Speed: 418 s
MSE: 0.0266 IOU: 0.9080 Speed: 855 s
Ground Truth
CP 032
CP 064
CP 128
MSE: 0.0249 IOU: 0.9175 Speed: 425 s
MSE: 0.0200 IOU: 0.9402 Speed: 484 s
MSE: 0.0169 IOU: 0.9483 Speed: 701 s
TT 032
TT 064
TT 128
MSE: 0.0269 IOU: 0.9190 Speed: 390 s
MSE: 0.0214 IOU: 0.9353 Speed: 476 s
MSE: 0.0191 IOU: 0.9473 Speed: 742 s
TU 032
TU 064
TU 128
WIRE-PE010 Reconstruction Results
MSE: 0.0386 IOU: 0.9164 Speed: 478 s
MSE: 0.0260 IOU: 0.9644 Speed: 494 s
MSE: 0.0169 IOU: 0.9847 Speed: 577 s
MSE: 0.0123 IOU: 0.9893 Speed: 529 s
MSE: 0.0509 IOU: 0.9824 Speed: 478 s
MSE: 0.0111 IOU: 0.9969 Speed: 805 s
MSE: 0.0200 IOU: 0.9740 Speed: 515 s
MSE: 0.0121 IOU: 0.9915 Speed: 551 s
MSE: 0.0098 IOU: 0.9996 Speed: 819 s

Figure 14. **Qualitative Lucy results.**



TANH-PE010 Reconstruction Results
MSE: 0.0597 IOU: 0.8992 Speed: 441 s
MSE: 0.0530 IOU: 0.9133 Speed: 447 s
MSE: 0.0501 IOU: 0.9120 Speed: 508 s
Ground Truth
CP 032
Ground Truth
CP 064
Ground Truth
CP 128
MSE: 0.0529 IOU: 0.9023 Speed: 441 s
MSE: 0.0427 IOU: 0.9475 Speed: 506 s
MSE: 0.0433 IOU: 0.9470 Speed: 762 s
Ground Truth
TT 032
Ground Truth
TT 064
Ground Truth
TT 128
MSE: 0.0485 IOU: 0.8645 Speed: 421 s
MSE: 0.0396 IOU: 0.9510 Speed: 498 s
MSE: 0.0370 IOU: 0.9573 Speed: 771 s
Ground Truth
TU 032
Ground Truth
TU 064
Ground Truth
TU 128
WIRE-PE010 Reconstruction Results
MSE: 0.0521 IOU: 0.9404 Speed: 452 s
MSE: 0.0321 IOU: 0.9772 Speed: 492 s
MSE: 0.0223 IOU: 0.9832 Speed: 618 s
Ground Truth
CP 032
Ground Truth
CP 064
Ground Truth
CP 128
MSE: 0.0165 IOU: 0.9944 Speed: 419 s
MSE: 0.0111 IOU: 0.9988 Speed: 481 s
MSE: 0.0126 IOU: 0.9974 Speed: 809 s
Ground Truth
TT 032
Ground Truth
TT 064
Ground Truth
TT 128
MSE: 0.0291 IOU: 0.9699 Speed: 517 s
MSE: 0.0295 IOU: 0.9874 Speed: 546 s
MSE: 0.0149 IOU: 0.9977 Speed: 783 s
Ground Truth
TU 032
Ground Truth
TU 064
Ground Truth
TU 128

Figure 15. **Qualitative Lucy results.**



ReLU-PE010 Reconstruction Results
MSE: 0.0325 IOU: 0.9414 Speed: 428 s
MSE: 0.0234 IOU: 0.9598 Speed: 428 s
MSE: 0.0173 IOU: 0.9707 Speed: 514 s
Ground Truth
CP 032
Ground Truth
CP 064
Ground Truth
CP 128
MSE: 0.0152 IOU: 0.9730 Speed: 362 s
MSE: 0.0122 IOU: 0.9801 Speed: 803 s
MSE: 0.0083 IOU: 0.9843 Speed: 702 s
Ground Truth
TT 032
Ground Truth
TT 064
Ground Truth
TT 128
MSE: 0.0181 IOU: 0.9705 Speed: 454 s
MSE: 0.0126 IOU: 0.9789 Speed: 483 s
MSE: 0.0169 IOU: 0.9842 Speed: 757 s
Ground Truth
TU 032
Ground Truth
TU 064
Ground Truth
TU 128
WIRE-PE010 Reconstruction Results
MSE: 0.0395 IOU: 0.9395 Speed: 479 s
MSE: 0.0265 IOU: 0.9699 Speed: 492 s
MSE: 0.0182 IOU: 0.9835 Speed: 589 s
Ground Truth
CP 032
Ground Truth
CP 064
Ground Truth
CP 128
MSE: 0.0153 IOU: 0.9777 Speed: 467 s
MSE: 0.0107 IOU: 0.9979 Speed: 481 s
MSE: 0.0083 IOU: 0.9996 Speed: 745 s
Ground Truth
TT 032
Ground Truth
TT 064
Ground Truth
TT 128
MSE: 0.0203 IOU: 0.9748 Speed: 497 s
MSE: 0.0124 IOU: 0.9910 Speed: 873 s
MSE: 0.0099 IOU: 0.9995 Speed: 805 s
Ground Truth
TU 032
Ground Truth
TU 064
Ground Truth
TU 128

Figure 16. **Qualitative Lucy results.**

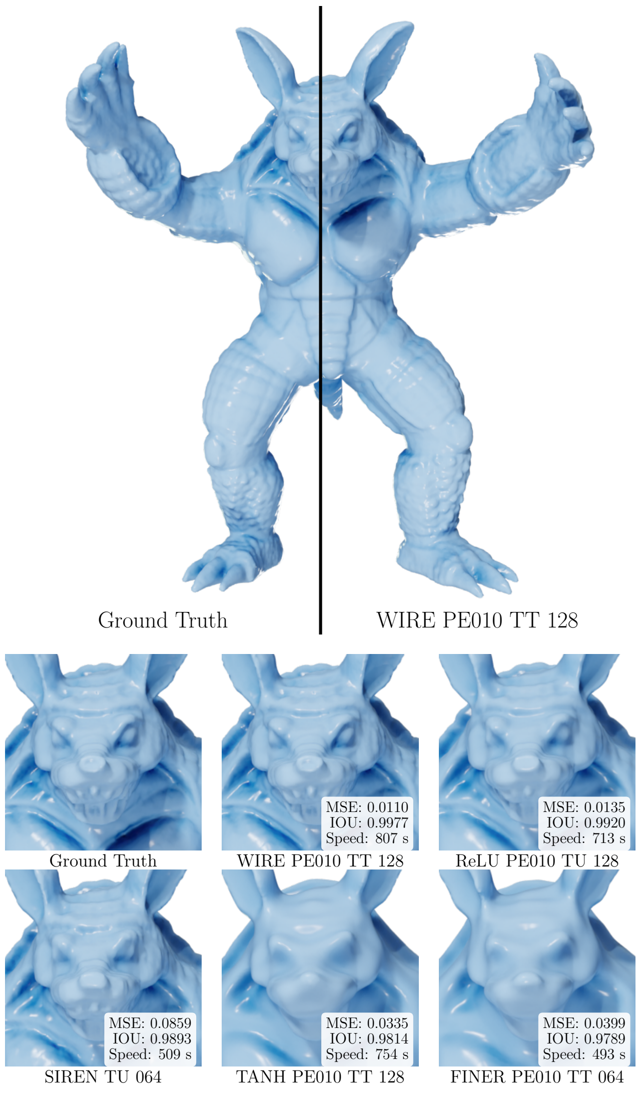


Figure 17. **Qualitative Armadillo comparison.**

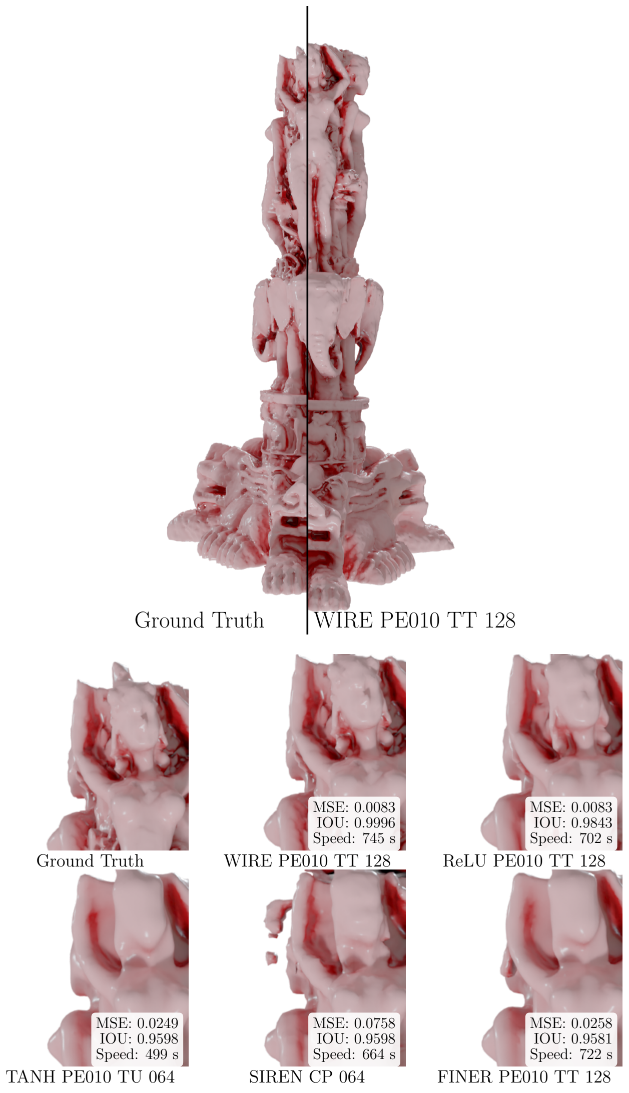


Figure 18. **Qualitative Statuette comparison.**

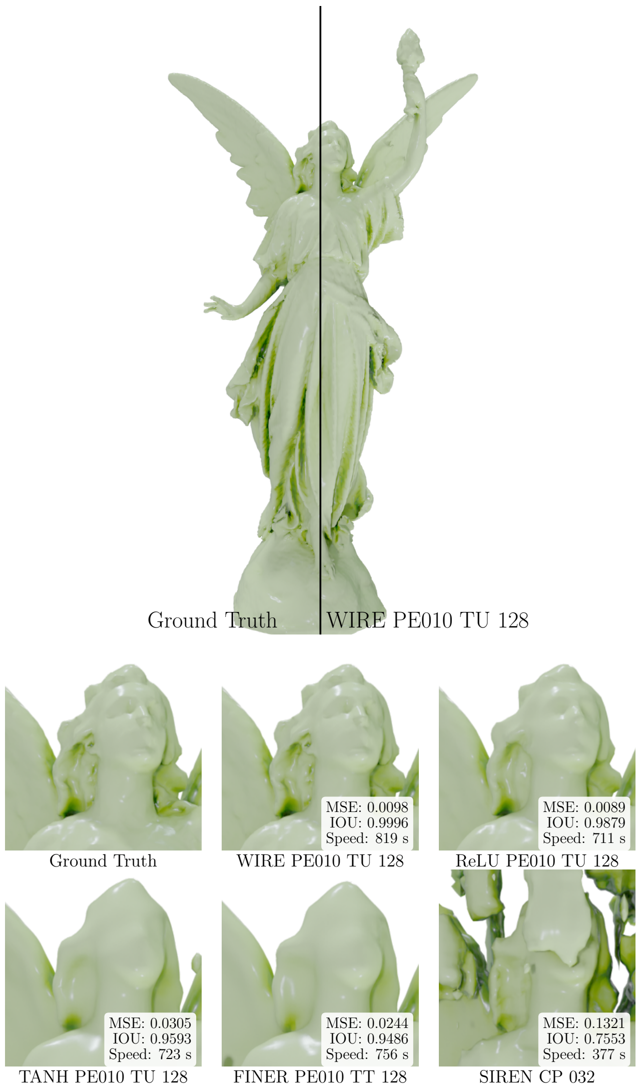


Figure 19. **Qualitative Lucy comparison.**

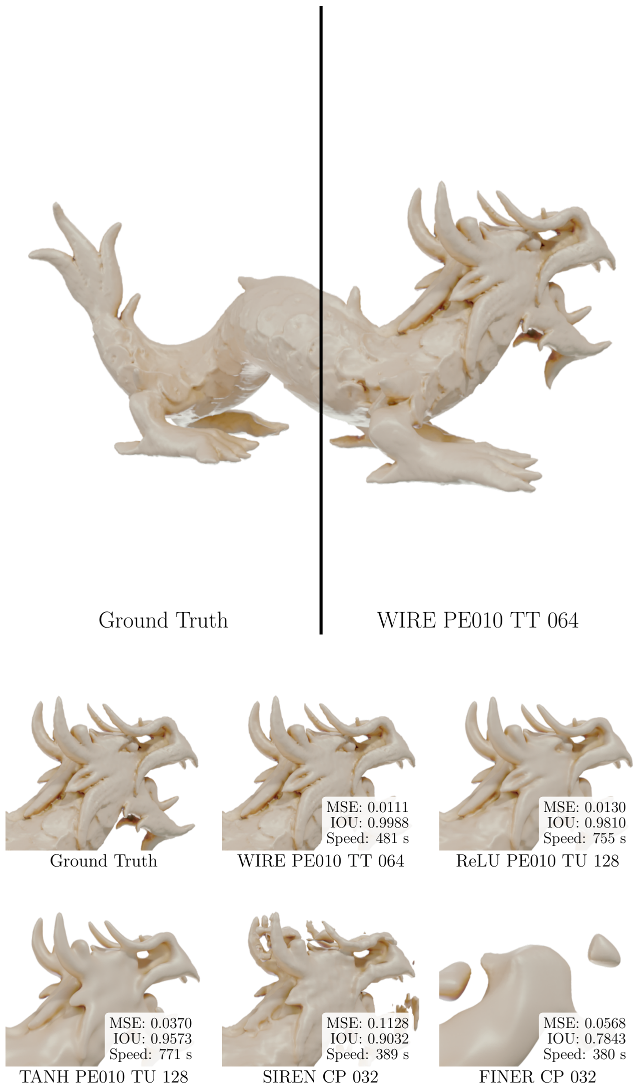


Figure 20. **Qualitative Dragon comparison.**

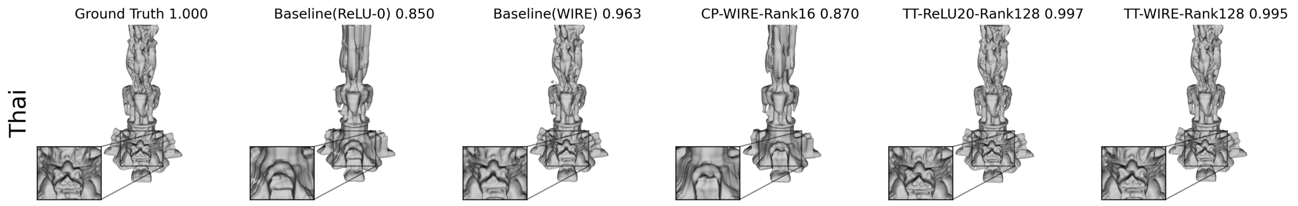


Figure 21. Here, we visualize the qualitative differences in complex Thai statuette 3D scan reconstructions. All models have been obtained from SDF using the marching cubes algorithm with the additional step of Laplacian smoothing to reduce introduced artifacts. We compare the results to the obtained 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.

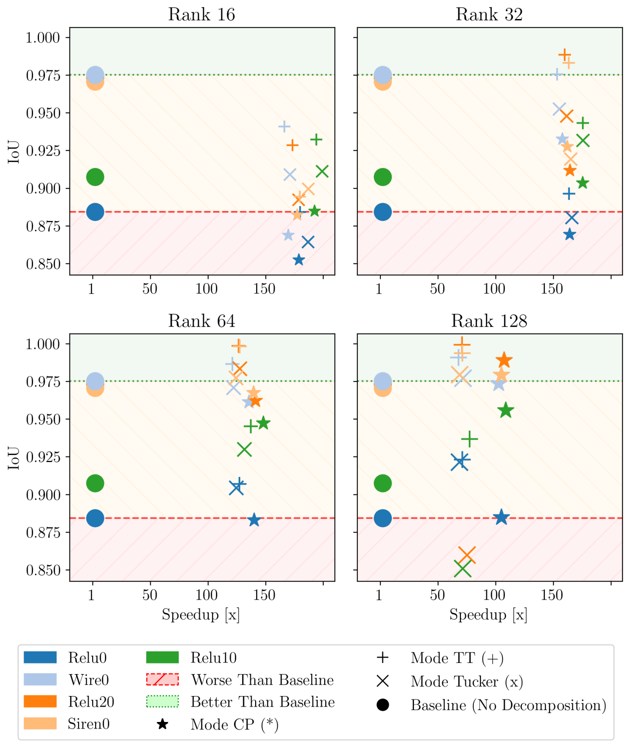


Figure 22. Quantitative results IoU vs Speedup, for model Lucy. The general trend is that a larger rank leads to a better IoU. Tensor-Train models perform the best when compared to the other two models, and in several models, they are better than baselines.

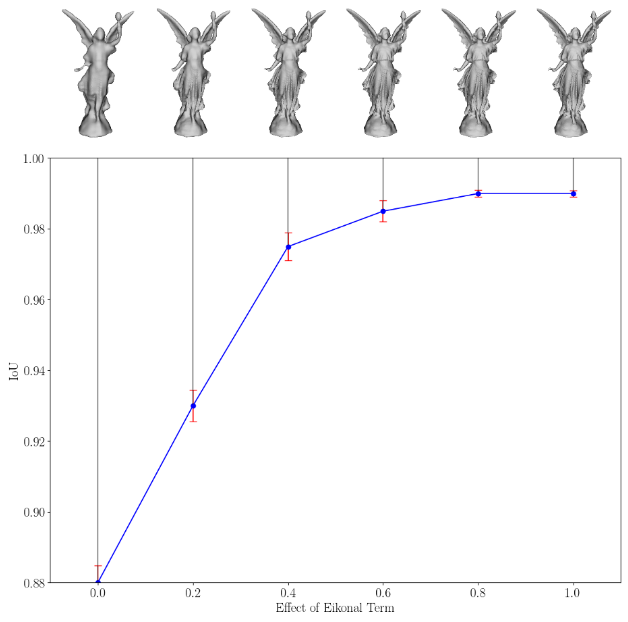


Figure 23. The effect of the Eikonal term on the learned SDF for the Lucy model. We varied the relative weight of the Eikonal term for the best performing F-INR: Mode TT Rank 128 and ReLU10 Backend. We observe that the presence of the Eikonal term significantly affects the solution, and after a relative weight of around 0.5, its effect is constant.

Backend	Mode	Rank	IoU	L2 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 [33]	-	-	0.978 ± 0.010	0.062 ± 0.015	18375
IGR [13]	-	-	0.985 ± 0.004	0.067 ± 0.014	16912
Factor Fields [5]	-	-	0.968 ± 0.01	N/A	280
	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
	TU	16	0.864 ± 0.006	0.220 ± 0.008	170
	TU	32	0.880 ± 0.003	0.209 ± 0.005	192
	TU	64	0.904 ± 0.006	0.198 ± 0.003	256
	TU	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
	TU	16	0.909 ± 0.014	0.198 ± 0.005	188
	TU	32	0.953 ± 0.004	0.137 ± 0.003	208
	TU	64	0.971 ± 0.001	0.111 ± 0.002	264
	TU	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
	TU	16	0.892 ± 0.043	0.185 ± 0.007	189
	TU	32	0.948 ± 0.021	0.146 ± 0.005	209
	TU	64	0.984 ± 0.039	0.104 ± 0.028	265
	TU	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
	TU	16	0.899 ± 0.028	0.182 ± 0.007	172
	TU	32	0.919 ± 0.015	0.141 ± 0.006	195
	TU	64	0.977 ± 0.018	0.098 ± 0.020	259
	TU	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
	TU	16	0.911 ± 0.013	0.223 ± 0.006	165
	TU	32	0.932 ± 0.004	0.190 ± 0.001	187
	TU	64	0.930 ± 0.032	0.168 ± 0.244	249
	TU	128	0.851 ± 0.038	0.458 ± 0.216	460

Table 2. **Results for geometry encoding task via SDFs.** Here we provide both IoU and L2 Error between the predicted SDFs and the Ground truth, for the Lucy model, taken from [9, 12, 21, 46]. A visualization of IoU values is provided in Figure 22.

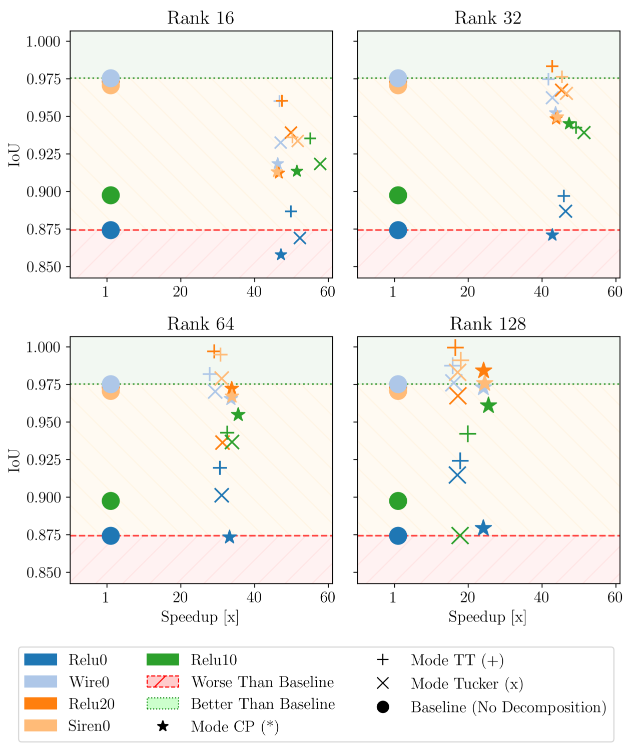


Figure 24. Quantitative results IoU vs. Speedup, for model Thai Statue. The general trend is that a larger rank leads to a better IoU. Tensor-Train models perform the best when compared to the other two models, and in several models, they are better than baselines. Even though all values seem closer, the visualizations shown in 21 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
Factor Fields [5]	-	-	0.954 ± 0.001	N/A	275
	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
	TU	16	0.869 ± 0.006	0.219 ± 0.006	308
	TU	32	0.887 ± 0.004	0.212 ± 0.005	347
	TU	64	0.901 ± 0.004	0.196 ± 0.004	519
	TU	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
	TU	16	0.933 ± 0.004	0.207 ± 0.006	351
	TU	32	0.962 ± 0.002	0.154 ± 0.004	387
	TU	64	0.970 ± 0.001	0.120 ± 0.003	565
	TU	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
	TU	16	0.939 ± 0.005	0.195 ± 0.005	341
	TU	32	0.967 ± 0.016	0.144 ± 0.008	376
	TU	64	0.936 ± 0.102	0.142 ± 0.022	543
	TU	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
	TU	16	0.934 ± 0.017	0.195 ± 0.015	318
	TU	32	0.965 ± 0.004	0.144 ± 0.001	352
	TU	64	0.979 ± 0.007	0.102 ± 0.033	528
	TU	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
	TU	16	0.918 ± 0.009	0.221 ± 0.005	290
	TU	32	0.939 ± 0.004	0.191 ± 0.003	326
	TU	64	0.937 ± 0.046	0.182 ± 0.110	494
	TU	128	0.874 ± 0.337	0.279 ± 0.304	943

Table 3. 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 [9]. A visualization of IoU values is provided in Figure 24 and the SDFs are visualized in Figure 21.

G. Neural Radiance Fields

Neural Radiance Fields (NeRFs) have emerged as one of the most prominent and widely adopted applications of Implicit Neural Representations (INRs). At their core, NeRFs learn a function

$$f(x,y,z,\theta,\phi) = (R,G,B,\sigma) \tag{22}$$

that maps 3D spatial coordinates and 2D viewing directions to color and density values. This function is typically optimized using a large collection of posed images, where training proceeds by casting rays from the camera centers, sampling points along each ray, querying the network, and compositing the results via volume rendering. This process, ray marching, translates the discrete supervision from images into a continuous coordinate-based learning problem.

A critical distinction between NeRF and other INR tasks is that the sampling is unstructured: we do not have a regular voxel grid, image plane, or temporal volume. Instead, the network must learn from arbitrary 5D input queries along thousands of rays per scene. In contrast, prior F-INR tasks like image representation or SDF modeling involved structured domains (e.g., 2D pixel grids or 3D voxel volumes), where low-rank patches could be more easily extracted and decomposed.

Previous efforts like TensoRF [4] and Plenoxels [52] introduced low-rank factorization or voxel-level grids to speed up NeRF training. However, these still rely on trilinear interpolation and do not use per-axis neural networks. They maintain a monolithic representation where the interpolation layers handle separability rather than the model itself.

G.1. Architecture

In F-INR, we reformulate the NeRF learning problem by decomposing the input coordinates axis-wise. Each of the five input dimensions, three spatial and two angular, is handled by a dedicated univariate MLP, and the outputs are combined via a tensor decomposition (CP, TT, or Tucker). This not only restructures the representation but also reduces redundancy and enables fine control over the rank and mode of the decomposition.

Given the massive variety of NeRF backbones and ray-marching optimizations in the literature, we do not aim to exhaustively benchmark across all variants. Instead, we focus on a representative and highly optimized baseline: InstantNGP [31]. This model relies heavily on a 3D multiresolution hash encoding, which forms the bulk of its computation.

Our adaptation replaces this 3D hash grid with three 1D hash grids, one per spatial axis, leaving all other components (ray tracing, batching, rendering, loss computation) identical. This effectively transforms InstantNGP into its F-INR counterpart. While we retain the same resolution and feature dimensions per level, the overall representation becomes more compact and structured.

Instead of a 3D hash grid, we use three independent 1D hash encodings for each axis.

$$\gamma_x(x) = \text{concat}_{l=1}^{L}\gamma_{x,l}(x) \in \mathbb{R}^{L\cdot r} \tag{23}$$

$$\gamma_y(y) = \text{concat}_{l=1}^{L}\gamma_{y,l}(y) \in \mathbb{R}^{L\cdot r} \tag{24}$$

$$\gamma_z(z) = \text{concat}_{l=1}^{L}\gamma_{z,l}(z) \in \mathbb{R}^{L\cdot r} \tag{25}$$

Each 1D hash grid uses linear interpolation:

$$\gamma_{x,l}(x) = (1-w_x)\cdot \mathbf{f}_{x,l,i} + w_x \cdot \mathbf{f}_{x,l,i+1} \tag{26}$$

The features are then combined via tensor decomposition, as usual in a F-INR manner:

- **CP Decomposition**:

$$\gamma_{\text{CP}}(x,y,z) = \gamma_x(x)\circ\gamma_y(y)\circ\gamma_z(z) \tag{27}$$

- **Tensor-Train (TT) Decomposition:**

$$\gamma_{\text{TT}}(x,y,z) = G_1[\gamma_x(x)]\cdot G_2[\gamma_y(y)]\cdot G_3[\gamma_z(z)] \tag{28}$$

- **Tucker Decomposition:**

$$\gamma_{\text{Tucker}}(x,y,z) = \mathcal{G}\times_1\gamma_x(x)\times_2\gamma_y(y)\times_3\gamma_z(z) \tag{29}$$

  where $\times_n$ denotes the mode-n Hadamard product.

Unsurprisingly, larger tensor ranks improve PSNR at the cost of slower training, especially since hash-based methods are already computationally intensive.

G.2. Quantitative Results

In the main paper, we have provided the results for drums and *Lego* models in the NeRF synthetic dataset. Here, we provide PSNR values for all the remaining models in the dataset, namely *chair*, *ficus*, *hotdog*, *ship*, *material*, and *mic*.

Backend	Mode	Rank	chair	ficus	materials	ship	hotdog	mic
ReLU+HE	CP	16	32.45	31.56	32.18	30.82	35.57	36.79
ReLU+HE	TT	16	35.34	34.10	32.40	31.68	37.90	37.50

Table 4. Quantitative Results for F-INR with ReLU + HE backend for additional synthetic NeRF datasets.

G.3. Qualitative Results

Here, we provide additional visualizations for all the models present in the NeRF synthetic dataset for the best performing F-INR (ReLU + HE - TT - Rank 16). The visualizations are provided in Figure 25. For the *hotdog* model, we also show how the learning is taking place over iterations in F-INR. At the start (a few hundred rays), the images rendered are not crisp but still capture the global shapes. More details (mustard on a hot dog, for example) are crisply rendered as the learning increases. We plot every 1000 iterations.

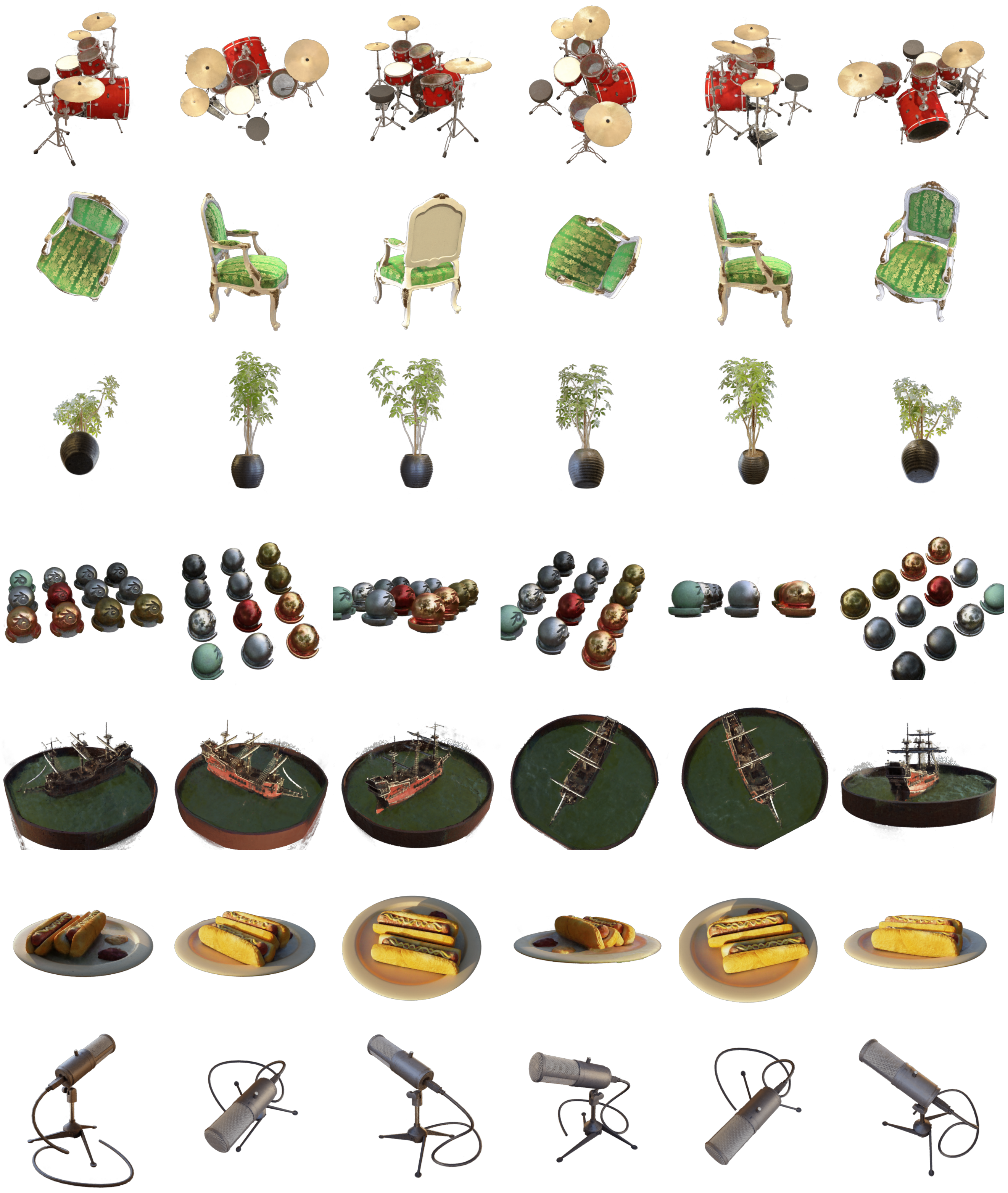


Figure 25. Novel Views generated for all models in Nerf Synthetic dataset [29] using F-INR ReLU + HE -TT- Rank 16.

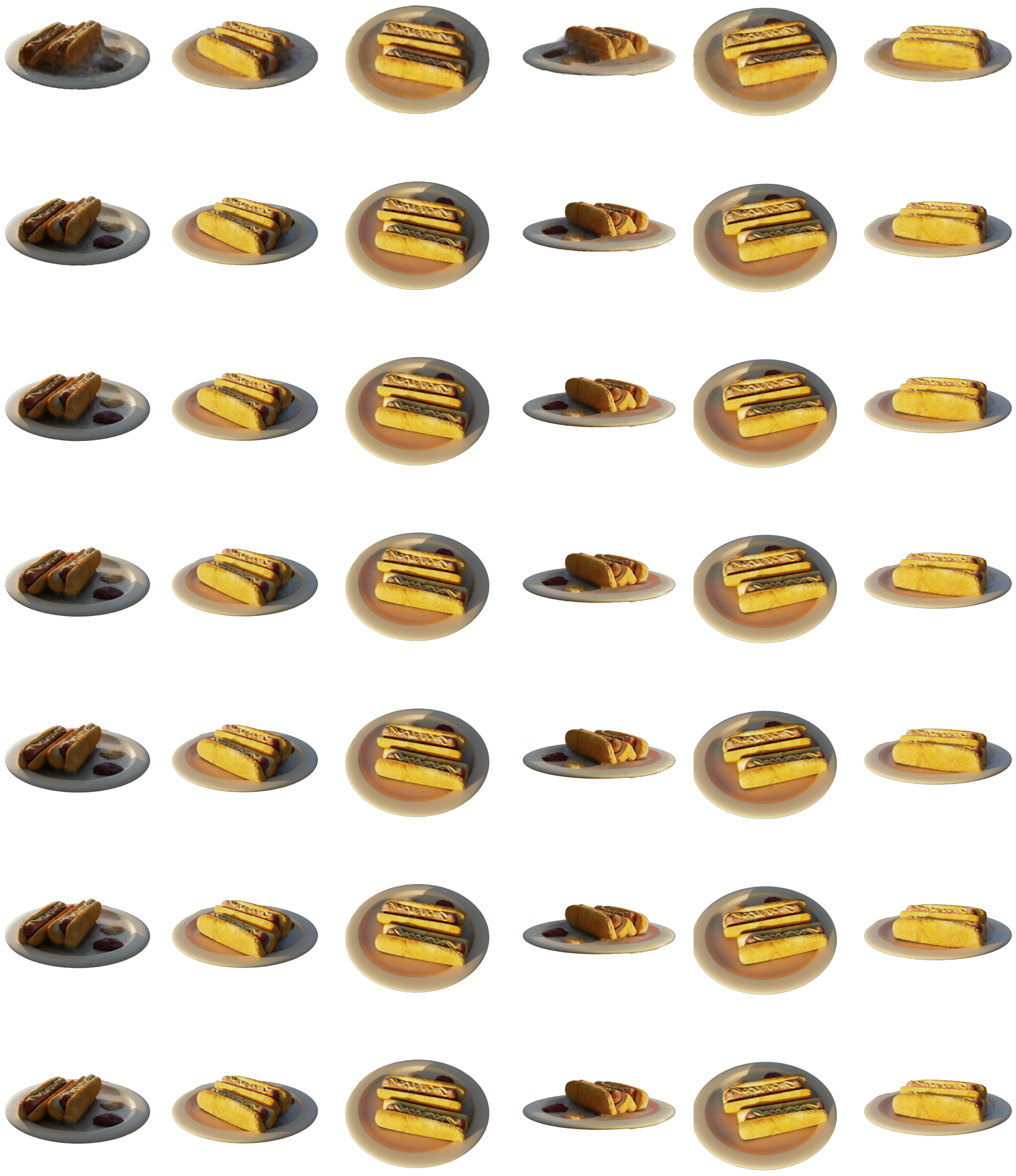


Figure 26. Evolution of rendering test images shown for the Hotdog model, for every 1000 iterations.

H. Navier-Stokes PDE Super Resolution

Super-resolution setup is: We have sparse, discrete observations, and we train an INR with these sparse observations and physics-informed loss [36] 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 [37] and MeshFreeFlowNet [18], and did not include them because they do not come under the family of INRs. There is no F-INR- equivalent of such methods. We include them for completeness, and we observe that for most backends, F-INR shows competitive performance.

H.1. Architecture

The training strategy is similar to that of [8]. The input for each network is the coordinates of respective dimensions, and the output is a velocity vector in the x and y directions. The Navier-Stokes equation is in its vorticity form, so this velocity is converted into vorticity, which is used to enforce the PDE term and compressibility constraint. The sparse observations are given as a 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. We uniformly sample 100 points per dimension as collocation points to enforce the PDE loss. We use WIRE and SIREN backends and ReLU+PE010 and ReLU+PE020 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 [18] and PhySR [37] 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 $10\times64\times64$, uniformly sampled from the original resolution dataset $101\times128\times128$. Thereby inducing a 40x sparsity. Each model trains for 50k Adam iterations, with resampling of collocation points for every 1000 iterations. We test the models by predicting the simulation at full resolution, thereby doing the super-resolution. We quantify the MSE error between the ground truth and the prediction.

H.2. Results

The results are given in Figure 27 and Table 5. 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 29. In the accompanying zip folder, we provide a video of the Navier-Stokes results over all time steps.

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 [50]	-	-	0.074 ± 0.008	28:40
CausalPINN [51]	-	-	0.070 ± 0.011	33:12
MFF Net [18]	-	-	0.048 ± 0.003	35:18
PhySR [37]	-	-	0.038 ± 0.020	27:05
	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
WIRE	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
	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
SIREN	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
	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
ReLU0	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
	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
ReLU10	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
	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
ReLU20	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 5. 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.

H.3. 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 28. 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.

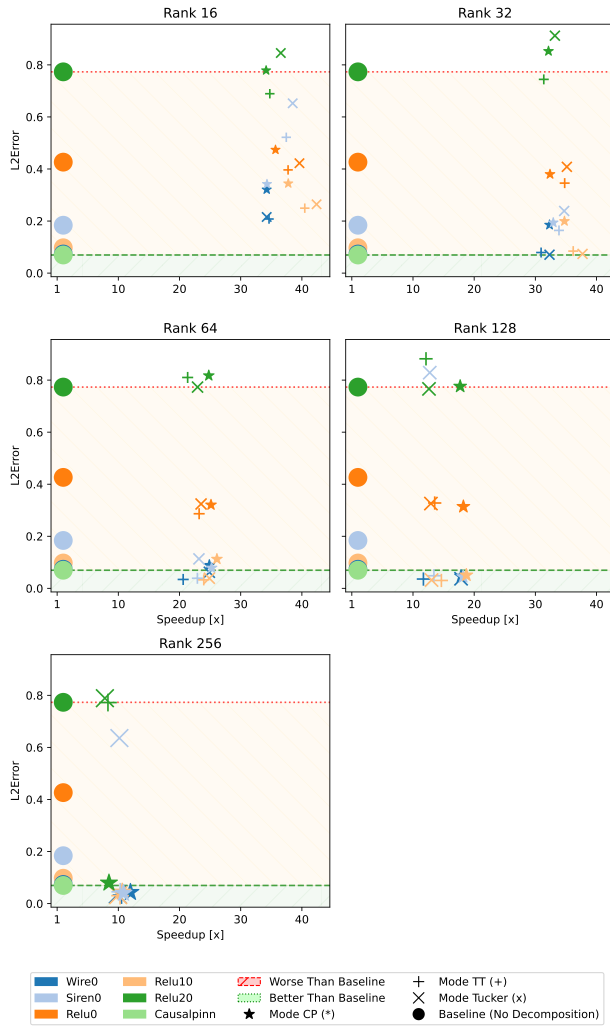


Figure 27. L_2 Error vs. Speedup plots for Super-Resolution of simulation task. This is a visualization of Table 5. This demonstrates that the proposed method consistently outperforms baseline methods in terms of speed and solution quality. Notably, the ReLU20 backend exhibits poor performance, whereas 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 results across all modes, underscoring its effectiveness in this context.

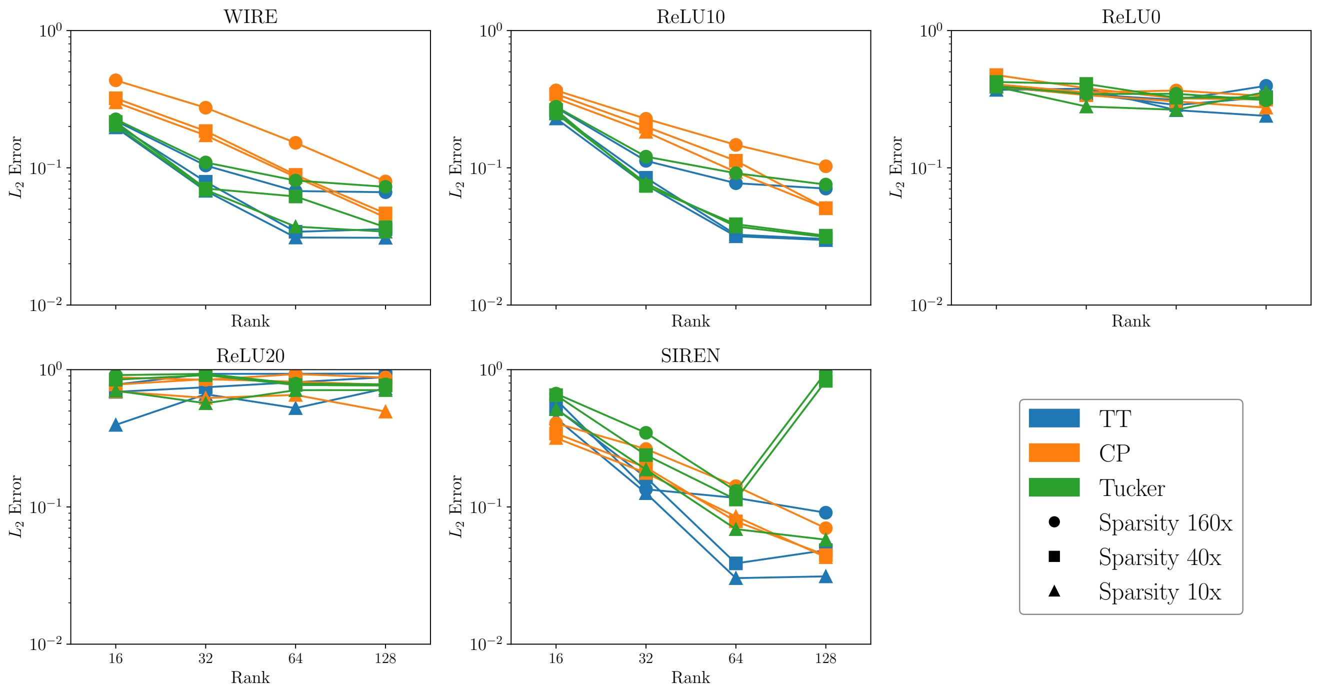


Figure 28. L_2 Errors for varying sparsity levels for super-resolution using F-INRs. Three levels of sparsity are tested for all the modes and backends. We observe that ReLU20 performs badly, and mode TT performs the best. Less sparsity leads to a better solution because more data is available. Nevertheless, F-INR s achieves competitive results even for higher sparsities.

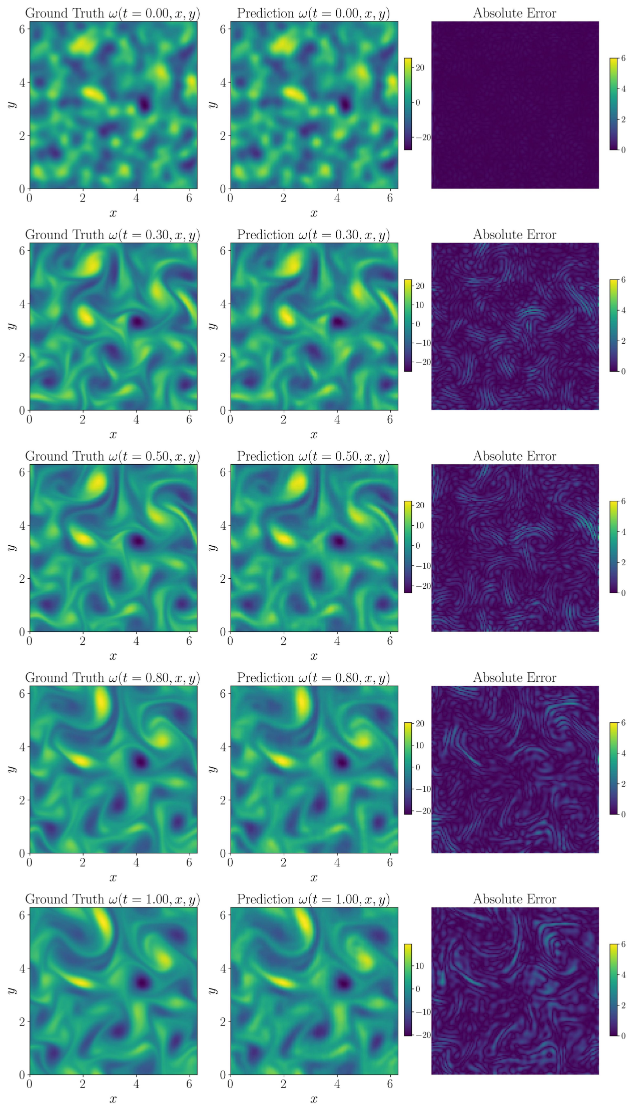


Figure 29. Visualization of vorticity for the super-resolution tasks using F-INR: **Mode Tucker with Rank 32 and WIRE backend**. We achieve a good prediction closer to the ground truth with a smaller rank of 32, highlighting the effectiveness of F-INR.

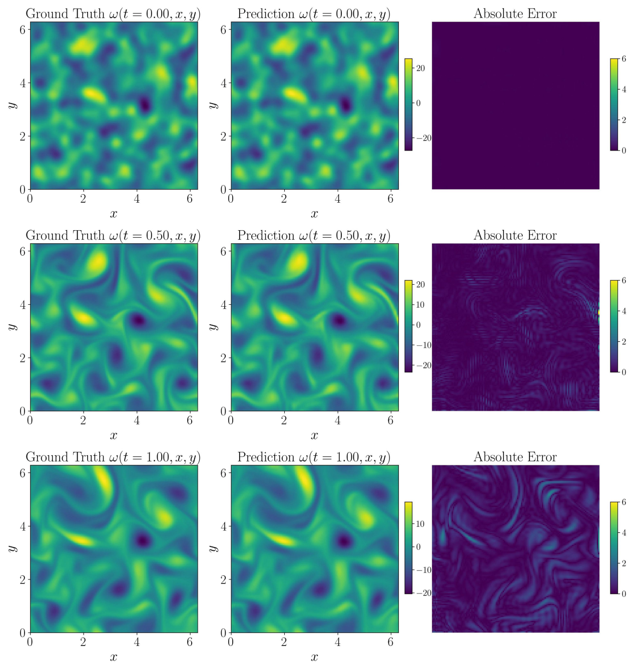


Figure 30. Visualization of vorticity for the super-resolution tasks using F-INR: **Mode TT with Rank 64 and ReLU10 backend**. We achieve good prediction closer to ground truth with a rank of 64; this solution surpasses the baseline implementations, highlighting the effectiveness of F-INR.

I. Failure Modes at Low Rank

Our main experiments focused on rank ranges that yield high-fidelity reconstructions. However, the tensor rank directly controls the model's capacity. Below a certain threshold, the parameter count becomes insufficient to capture the complexity of the target signal, inevitably leading to reconstruction failure or strong artifacts. For this ablation study, we deliberately probe this lower bound by evaluating extremely low ranks (e.g., 2, 4) to analyze the resulting failure modes and understand how different backbones degrade under severe capacity constraints.

What Rank Controls. In F-INR, the rank r of the tensor decomposition controls the capacity to model *cross-dimensional interactions*. With CP/TT/Tucker, the predictor is a sum (or chain) of separable factors. If the target function has an effective, yet unknown, tensor rank $R^\star$ (with respect to the chosen mode), any choice $r < R^\star$ forces a structural approximation error that optimization cannot remove. However, still a minimal error is aimed for.

Discrete View (Intuition) [20]. Let us consider the tensor $X \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ formed by evaluating the signal on a grid. The best rank-$(r_1, \ldots, r_d)$ Tucker approximation $X_{(r_1,\ldots,r_d)}$ satisfies the classical tail-energy bound

$$\|X - X_{(r_1,\ldots,r_d)}\|_F^2 \leq \sum_{k=1}^{d} \sum_{i>r_k} \left(\sigma_i^{(k)}\right)^2, \qquad (30)$$

where $\sigma_i^{(k)}$ are the singular values of the mode-k unfolding of X. Thus, picking r_k below the intrinsic rank of mode k leaves a residual that cannot be fit away by training the model, simply lacks the degrees of freedom to couple modes. This is a very active area of research, to find the optimum rank of a tensor decomposition, and we suggest these works for further reading [19, 20, 28].

Empirical Signatures of Under-ranked Models. Across tasks, we consistently observe:

- **Images.** For the extreme low cases no suitable reconstruction could be reached, as clearly visible in Figure 31. Over-smoothing and loss of high-frequency details; edges appear "soft" and textures collapse into axis-aligned patterns. Artifacts from the backend, such as hash encoding, also show up.
- **NeRF (e.g., *Lego*).** With very low ranks $r \in \{1, 2, 4\}$, geometry is under-reconstructed (with artifacts from hash encoding), view-dependent effects degenerate to diffuse color, and PSNR saturates well below monolithic baselines, as seen in Figs. 32 and 33.

Why Optimization Cannot Fix it. For TT, the smallest internal rank is a hard bottleneck: it is the upper lower bound the information that can flow across any bipartition of dimensions. For Tucker, small factor ranks cap the dimension of each per-mode subspace, and the core cannot create interactions not spanned by those subspaces. Consequently, gradient-based optimization can reduce noise within the chosen subspaces but cannot synthesize missing cross-mode structure.

Insights for Rank Selection. We would treat r as the primary capacity control for *interactions* between the decomposition components. Very low ranks predictably underfit, as seen in Fig. 31. We would start from a modest $r = 8$ and increase only if (i) loss plateaus without overfitting signals and (ii) qualitative artifacts match the under-ranked signatures. Large ranks ($r = 256$) rarely help beyond small, diminishing improvements and can introduce optimization instability (e.g., poorly conditioned Tucker cores). As we have seen in Fig. 9 (denoising ablations), large ranks also enable that noise patterns are learned within the components.

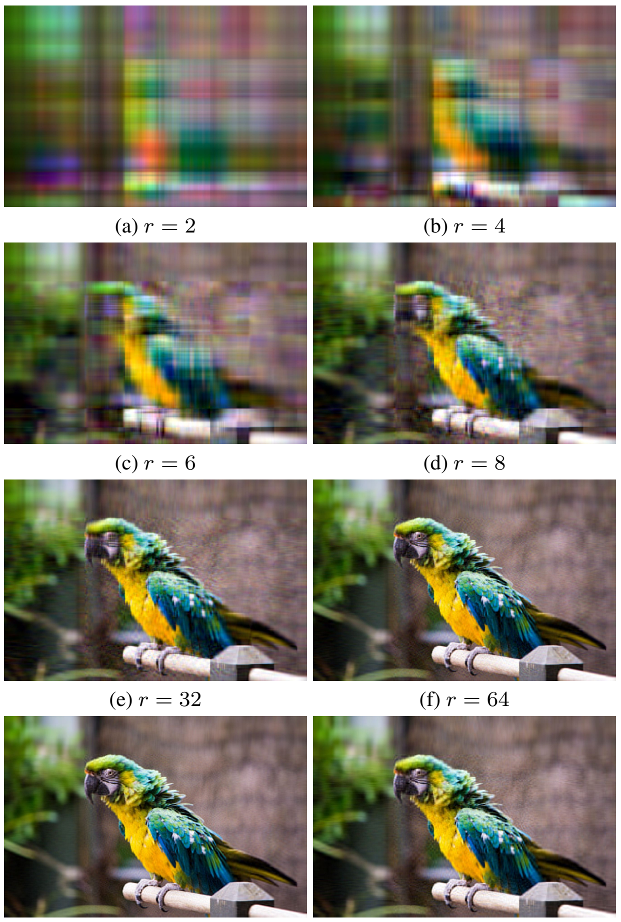
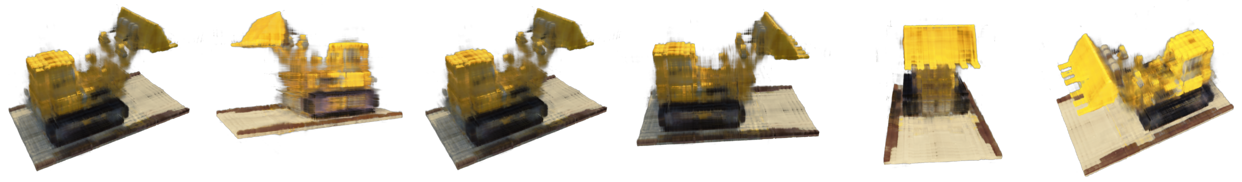
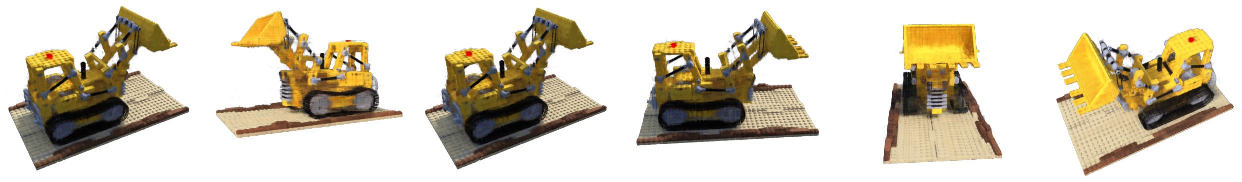

(a) $r = 2$ (b) $r = 4$ (c) $r = 6$ (d) $r = 8$ (e) $r = 32$ (f) $r = 64$ (g) $r = 128$ (h) $r = 316$

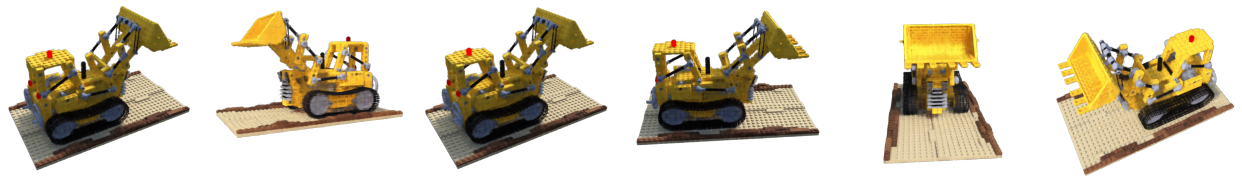
Figure 31. **Low Rank Ablations - Images.** Performance of F-INR (ReLU + Hash) on Image task for increasing rank $r = [2, 4, 8, 16, 32, 64, 128, 315]$ without changing any other hyperparameter. Extremely low ranks cannot reconstruct the image, leading to artifacts and smoothening. As the rank increases, the fidelity of the image increases.



(a) $r = 1$

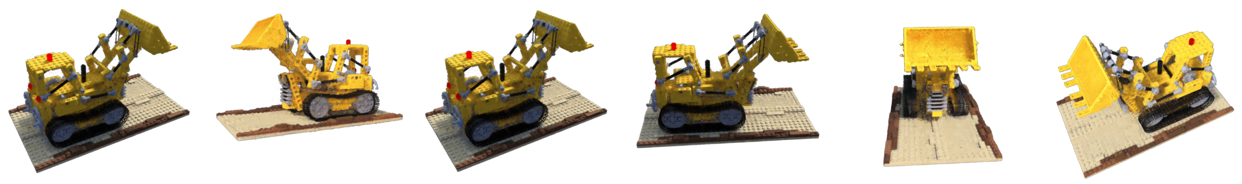


(b) $r = 2$

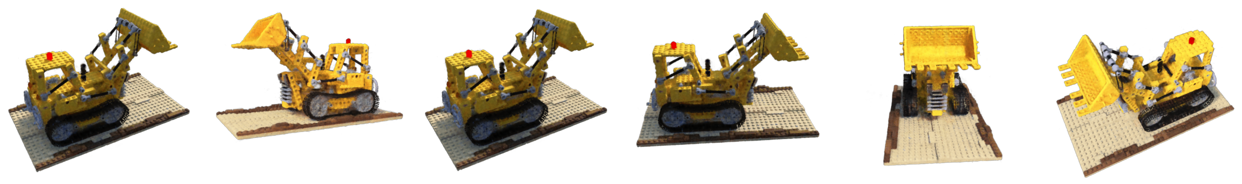


(c) $r = 4$

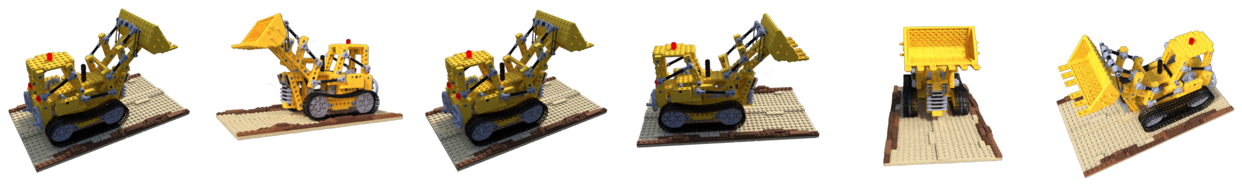
Figure 32. **Low Rank Ablations CP - NeRF Lego.** Performance on NeRF task for low ranks of 1, 2, and 4. For extremely low ranks, we see artifacts and the inability of the F-INR to learn fine features, though the backend neural network is unchanged.



(a) $r = 1$



(b) $r = 2$



(c) $r = 4$

Figure 33. **Low Rank Ablations TT - NeRF Lego.** Performance on NeRF task for low ranks of 1, 2, and 4, respectively. However, these visual performance is better than their CP counterparts, owing to the larger capacity of TT.

J. Practitioner Guide: Backend, Mode, Rank

This section synthesizes our experimental findings into practical recommendations for applying the F-INR framework. We provide task-specific guidance on selecting the optimal combination of (i) INR backend (e.g., SIREN, WIRE, ReLU+PE), (ii) decomposition mode (CP, TT, Tucker), and (iii) tensor rank. These guidelines are derived from our evaluations across diverse applications, including image and video representation, SDF reconstruction, NeRF, and physics-informed super-resolution. Our aim is to provide a clear, empirically-grounded starting point for practitioners and future research leveraging our functional decomposition paradigm.

J.1. Investigating Task Constraints

Do you need gradients w.r.t. input (e.g., Eikonal/PDE terms)? Use a smooth, fully differentiable backend (ReLU+PE, WIRE, or SIREN). *Avoid hash encodings*, which are piecewise non-smooth and incompatible with Eikonal-style regularizers.

Is your INR dense/grid-like or sparse/unstructured? Both work with F-INR, but sparse ray sampling (NeRF) benefits from encoders with strong locality (Hash) when gradients w.r.t. input are not required. When dealing with grid structures, utilizing the outer products will help in reducing the forward pass complexity.

J.2. Backends: Recommend Task Defaults

- **PDE/SDF (Input-gradients Required):** ReLU+PE or WIRE. In our SDF benchmark, WIRE+PE with TT gave the lowest MSE/highest IoU; hash was excluded by design. For Navier-Stokes, ReLU+PE and WIRE with TT/Tucker dominated across ranks.
- **NeRF (No Input-gradient Regularization):** ReLU+Hash, with TT mode. TT at $r = 16$ outperformed CP/Tucker on Lego/Drums and stayed strong across the remaining scenes.
- **Images/Video:** All backends benefit from factorization. For single images, CP at moderate-to-high ranks achieved large PSNR gains and speedups across SIREN/WIRE/ReLU+PE; for videos, TT tended to win, while WIRE under default hyperparameters performed slightly worse.

J.3. Modes: Recommendations

- **TT** is a robust first choice when cross-axis couplings matter (PDEs, NeRF, video). It consistently achieved the best or tied-best metrics, and remained stable as r grew.
- **Tucker** is competitive on PDE/SDF and sometimes within a few accuracy/metric points of TT. Use when you want per-mode rank control and a compact core.
- **CP** is sufficient for 2D images at modest cost, but shows slight performance degradations for PDEs at low rank.

J.4. Rank: Capacity Knob for Interactions

Starting Points. Use the smallest rank that *does not*:
- Images: $r \in [64, 256]$ (CP/TT)
- NeRF: $r = 16$ (TT)
- PDE/SDF: $r \in [64, 256]$ (TT/Tucker)

In SDF (Armadillo and others), WIRE+PE with TT at r=128 set the best MSE/IoU. In Navier-Stokes, error decreased monotonically up to $r \in [128, 256]$ across TT/Tucker, with large speedups vs. monolithic baselines. In NeRF, TT r=16 with ReLU+Hash was the best among modes at comparable budgets.

Increasing the Rank. Increase r if: (i) validation loss (or physics residual) plateaus over several epochs, (ii) edges/textures (images/NeRF) or gradient constraints (SDF/PDE) remain underfit, and (iii) larger r gives diminishing returns. The rank-error trends and plateau behavior are visible in the ablation sections.

J.5. Decide Mode & Rank

1. **Choose backend by constraint:** if gradients are needed, use ReLU+PE or WIRE; else prefer ReLU+Hash for 3D/5D fields.
2. **Run short probes (1-2k steps) at a small rank r_0:** TT, Tucker, CP. Pick the mode with the largest Δloss-per-minute. Our ablations consistently rank TT$\geq$Tucker$\gg$CP for Navier-Stokes; TT¿CP/Tucker for NeRF; CP is strong for 2D images at comparable cost.
3. **Grow r cautiously:** double r only when the probe stalls; stop when Δmetric is $<$1–2% over a full probe window. (Our tables show clear diminishing returns beyond $r \in [128, 256]$ for PDE; NeRF often saturates by r=16.)

J.6. PE/Encoder sensitivity (Practical Note)

For PE/encoder frequencies, overly aggressive settings can hurt PDE stability; in Navier-Stokes, ReLU+PE with a higher frequency performed worse (ReLU20), whereas a milder setting (ReLU10) achieved substantially lower error. Use conservative PE at first, then tune up only if needed [29, 41]. For WIRE, we adopt the default ω=30, s=30 used throughout unless stated otherwise [39].

J.7. Ready-To-Use Defaults

These defaults match the best or near-best settings in our tables while keeping training time modest.

Task	Backend	Mode	Rank r
2D Image (Grid)	SIREN or WIRE (+PE)	CP or TT	128-256
SDF (Eikonal PDE)	WIRE or ReLU+PE (no Hash)	TT ($\approx$ Tucker)	128 (64-256)
NeRF (Synthetic)	ReLU+Hash	TT	16 (8-32)
Video (Per-frame Grid)	ReLU+PE / ReLU+Hash	TT	128-256
Navier-Stokes SR	ReLU+PE or WIRE (no Hash)	TT ($\geq$ Tucker $\gg$ CP)	128-256

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. 9, 10

[2] Yash Bhalgat. Hashnerf-pytorch. `https://github.com/yashbhalgat/HashNeRF-pytorch/`, 2022. 7

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 1, 6

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European conference on computer vision*, pages 333–350. Springer, 2022. 5, 23

[5] Anpei Chen, Zexiang Xu, Xinyue Wei, Siyu Tang, Hao Su, and Andreas Geiger. Factor fields: A unified framework for neural fields and beyond. *arXiv preprint arXiv:2302.01226*, 2023. 4, 7, 15, 21, 22

[6] 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. 12

[7] Zhang Chen, Zhong Li, Liangchen Song, Lele Chen, Jingyi Yu, Junsong Yuan, and Yi Xu. Neurbf: A neural fields representation with adaptive radial basis functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4182–4194, 2023. 6

[8] 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. 8, 26

[9] 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. 15, 21, 22

[10] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989. 8

[11] 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. 12

[12] 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. 15, 21

[13] 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. 15, 21

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

[15] 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. 3

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. 8

[17] Xinquan Huang and Tariq Alkhalifah. Efficient physics-informed neural networks using hash encoding. *Journal of Computational Physics*, 501:112760, 2024. 6, 10

[18] 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. 26

[19] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. 3, 30

[20] 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. 30

[21] 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. 15, 21

[22] Ruofan Liang, Hongyi Sun, and Nandita Vijaykumar. Coordx: Accelerating implicit neural representation with a split mlp architecture. *ArXiv*, abs/2201.12425, 2022. 6

[23] Zhen Liu, Hao Zhu, Qi Zhang, Jingde Fu, Weibing Deng, Zhan Ma, Yanwen Guo, and Xun Cao. Finer: Flexible spectral-bias tuning in implicit neural representation by variable-periodic activation functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2713–2722, 2024. 4, 5

[24] 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. 15

[25] 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. 3

[26] 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. 12

[27] 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. 13

[28] M. Messiter and Y. Shamash. Product and sum separable functions. *IEEE Transactions on Automatic Control*, 30(7): 694–697, 1985. 30

[29] 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, 4, 5, 9, 13, 24, 32

[30] Thomas Müller. tiny-cuda-nn, 2021. 6

[31] 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. 5, 6, 7, 9, 10, 13, 23

[32] I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. 3, 15

[33] 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. 15, 21

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 1, 6

[35] M.D. Raisinghania. *Ordinary and Partial Differential Equations*. S. Chand Publishing, 1991. 8

[36] 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. 26

[37] 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. 26

[38] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 1964. 8

[39] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18507–18516, 2023. 1, 4, 5, 9, 10, 11, 13, 32

[40] 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, 4, 5, 13

[41] 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. 4, 10, 32

[42] Jiaxiang Tang. Torch-ngp: a pytorch implementation of instant-ngp, 2022. https://github.com/ashawkey/torch-ngp. 7

[43] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable nerf via rank-residual decomposition. *arXiv preprint arXiv:2205.14870*, 2022. 7

[44] Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Honghao Lu, Ethan Yan, Jack Kosaian, Mark Hoemmen, Haicheng Wu, Andrew Kerr, Matt Nicely, Duane Merrill, Dustyn Blasig, Fengqi Qiao, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, and Manish Gupta. CUTLASS, 2023. 7

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

[46] 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. 15, 21

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

[48] 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. 3

[49] 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. 15

[50] 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. 26

[51] 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. 26

[52] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2(3):6, 2021. 23

[53] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016. 3

[54] 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. 12, 13

[55] Hao Zhu, Zhen Liu, Qi Zhang, Jingde Fu, Weibing Deng, Zhan Ma, Yanwen Guo, and Xun Cao. Finer++: Building a family of variable-periodic functions for activating implicit neural representation. *arXiv preprint arXiv:2407.19434*, 2024. 4, 5