EXPLORING STATE-SPACE MODELS FOR DATA-SPECIFIC NEURAL REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper studies the problem of data-specific neural representations, aiming for compact, flexible, and modality-agnostic storage of individual visual data using neural networks. Our approach considers a visual datum as a set of discrete observations of an underlying continuous signal, thus requiring models capable of capturing the inherent structure of the signal. For this purpose, we investigate state-space models (SSMs), which are well-suited for modeling latent signal dynamics. We first explore the appealing properties of SSMs for data-specific neural representation and then present a novel framework that integrates SSMs into the representation pipeline. The proposed framework achieved compact representations and strong reconstruction performance across a range of visual data formats, suggesting the potential of SSMs for data-specific neural representations.

1 INTRODUCTION

Recent years have witnessed growing interest in overfitting a neural network to a single visual datum such as image (Dupont et al., 2021; Strümpler et al., 2022), video (Chen et al., 2021; Mentzer et al., 2022), or 3D instance (Martin-Brualla et al., 2021; Zhang et al., 2020). This *data-specific neural representation* paradigm, prevalent in implicit neural representations (INRs) (Sitzmann et al., 2020) and neural compression (Ballé et al., 2016; Cheng et al., 2020), aims to directly encode a datum into an embedding or the weights of a compact neural model. Such a paradigm not only serves as an effective data compression method but also offers a standardized data format that can accommodate various modalities for future neural network training (Dupont et al., 2022), with some approaches further enabling downstream applications such as spatial/temporal super-resolution (Chen et al., 2022b), denoising (Xu et al., 2022), and in/outpainting (Skorokhodov et al., 2021; Chen et al., 2023).

The central objective of the data-specific neural representations is to represent a single datum with minimal parameter complexity and maximal reconstruction quality. One of the effective strategies to achieve this comes from the recognition that visual data are essentially arrays of pixels sampled at discrete intervals from continuous signals (Sitzmann et al., 2020; Xu et al., 2022; Tancik et al., 2020; Saragadam et al., 2023). The core idea behind this approach is to project input data onto a set of established basis functions and only save their coefficients, so that the coefficients reconstruct not only the input but also the continuous signal from which the input is sampled. Although this concept has served as a fundamental principle for effective compression and reconstruction of visual data (Cooley et al., 1969; Richardson, 2011), modern approaches to data-specific neural representation do not take it into account due to the lack of well-established neural network architectures that incorporate the concept; they have instead focused merely on coordinate-to-RGB mapping (Martin-Brualla et al., 2021; Strümpler et al., 2022), bit-level quantization (Xu et al., 2018; Gordon et al., 2023) or improving the capacity of conventional neural networks to implicitly manage input redundancies (Zhou et al., 2018; Li et al., 2018).

Recently, the rise of state-space models (SSMs) has opened a new pathway to this challenge, as SSMs provide a framework for modeling continuous signals in a way that aligns with the objectives of compact neural representations. To be specific, the hidden state of SSM was initially designed to represent the coefficients that reconstruct observed data using a set of orthogonal polynomial bases (Gu et al., 2020; 2022b), which generalizes to the traditional compression algorithms. Although the design of SSMs has become more implicit (Gu et al., 2021b; Smith et al., 2022; Gu et al., 2021a), such that their hidden states no longer explicitly represent coefficients of such continuous bases, they

still retain the desirable properties necessary for effective signal modeling (Gu et al., 2021b; Guo et al., 2025; Rao, 1987; Rao & Arun, 1992), so it is worth exploring their applications.

Driven by this motivation, we explore the potential for incorporating SSMs within data-specific neural representations. We investigate the effectiveness of SSMs in compressing input data and capturing underlying signal structures, and empirically demonstrate their benefits in enhancing the reconstruction quality. However, a naïve application of SSMs presents key challenges: (1) they primarily operate on 1D sequence inputs, necessitating unnatural scanning for multi-dimensional data, and (2) they inherently preserve input sequence length, which makes them unsuitable for an effective compression method. To address these limitations, we propose *structured state-space kernel* (S3K), which distills the expressive power of SSMs into convolutional kernels. We design the kernel parameters in a way that the convolution output matches the hidden state representation of SSMs, effectively preserving their reconstruction capability. Through seamless integration with convolution, it naturally processes multi-dimensional inputs while inherently enabling expressive downsampling. In summary, our contribution is three-fold as follows:

- We for the first time investigate the integration of state-space models (SSMs) into data-specific neural representation frameworks, providing a theoretical background that explains their potential benefits in improving both expressive power and efficiency.
- We introduce S3K, an SSM-derived convolutional kernel that inherits the expressive power of SSMs while mitigating their limitations for multi-dimensional processing and downsampling.
- Our framework shows promising results across diverse visual data reconstruction tasks—images (Kodak (Kodak, 1993), CLIC2020 (Toderici et al., 2020)), videos (Bunny (Roosendaal, 2008), UVG (Mercat et al., 2020), DAVIS (Perazzi et al., 2016)), and 3D objects (Objaverse (Deitke et al., 2023))—highlighting its potential for advancing data-specific neural representations.

2 PRELIMINARY: STATE-SPACE MODEL

SSM is a function that maps a 1D input signal $\phi(x)$ to a 1D output signal $y(x)$ of the same length through the latent state $h(x) \in \mathbb{C}^N$ based on the following linear differential equation:

$$\begin{aligned} h'(x) &= \mathbf{A}h(x) + \mathbf{B}\phi(x), \\ y(x) &= \mathbf{C}h(x), \end{aligned} \tag{1}$$

where $\mathbf{A} \in \mathbb{C}^{N\times N}$ is the state transition matrix, and $\mathbf{B} \in \mathbb{C}^N$ and $\mathbf{C} \in \mathbb{C}^N$ are projection parameters. Solving the linear differential equation (1) to explicitly express h yields [1]:

$$h(x) = \int_0^x e^{(x-\tau)\mathbf{A}}\mathbf{B}\phi(\tau)d\tau \in \mathbb{C}^N, \tag{2}$$

where each $h_k(x) \in \mathbb{C}$ corresponds to:

$$h_k(x) = \int_0^x e^{(x-\tau)\mathbf{A}_k}\mathbf{B}\phi(\tau)d\tau = \left\langle \phi(\tau), \overline{e^{(x-\tau)\mathbf{A}_k}\mathbf{B}} \right\rangle_{[0,x]} := \left\langle \phi(\tau), \xi_k(\tau, x) \right\rangle_{[0,x]}. \tag{3}$$

Here, $\langle\cdot,\cdot\rangle_{[x_1,x_2]}$ is a complex function space inner product in the given domain $[x_1, x_2]$. Intuitively, Eq. (3) tells that the k-th element of the hidden state $h_k(x)$ is a projection of the input $\phi(\tau)_{\tau\le x}$ onto the function ξ_k. Gu et al. (2020; 2021b; 2022b;a) have established that through appropriate parameterization of the matrix $\mathbf{A}$, $\{\xi_k\}$ can serve as a set of basis functions, which enable the model to effectively capture and retain key information from the entire sequence history up to the current position x. This ability to project input signals onto a learned basis makes SSMs a natural fit for compression, since such a capability to model signal representation becomes beneficial. Various ways to parametrize $\mathbf{A}$ have been explored: HiPPO (Gu et al., 2020), diagonal plus low rank (Gu et al., 2021a), and diagonal (Gupta et al., 2022; Gu et al., 2022a). Among these, using a diagonal parameterization of $\mathbf{A}$ has gained popularity for its easier formulation while maintaining expressivity (Smith et al., 2022; Gu & Dao, 2023).

[1] See Appendix A.2 for details.

3 EXPLORING SSMS FOR DATA-SPECIFIC NEURAL REPRESENTATIONS

This section delves into advantages and proper architecture designs of data-specific neural representations using SSMs. First, we examine how SSMs encode input data and highlight their effectiveness in data-specific neural representation. Then, through an extensive experimental analysis, we identify the key characteristics of network architectures incorporating SSMs for this purpose.

3.1 WHAT DO SSMS ENCODE?

To understand the operational principles of SSMs and interpret their features, we bring up a classical signal processing task, *sinusoid problem*, which aims at estimating parameters of sinusoidal signals that make up the input signal. Given an input function $\phi(t)$ that takes 1D coordinate $t \in [0, L]$, we are interested in finding $\theta_n(t)$ and c_n such that

$$\phi(t) = \sum_{n=1}^{N} c_n e^{i\theta_n(t)}, \tag{4}$$

where $e^{i\theta_n(t)}$ is the n-th sinusoidal basis and c_n is its coefficient. Note that this form generalizes to various sinusoidal transformation methods, *e.g.*, setting $\theta_n(t) = -2\pi nt/L$ leads to the choice of bases used in discrete Fourier transform (Cooley et al., 1969). Under a proper choice of $\theta_n(t)$, estimating the parameter c_n that approximates the input signal $\phi(t)$ offers an effective method for compression.

Interestingly, the SSM formulation allows the input signal to be decomposed into a sinusoidal form:

Theorem 3.1. *Let* $\mathbf{A}$ *be diagonalizable over* $\mathbb{C}$ *with non-zero distinct eigenvalues* $\{\lambda_i\}$*. Given* $\mathbf{A}$*,* $\mathbf{B}$*, and the hidden state* h *computed by Eq. (2), there exists a function* $f : (\mathbf{A}, \mathbf{B}) \mapsto \mathbf{F} \in \mathbb{C}^{N\times N}$ *with which one can decompose the input function* $\phi(t)$ *as a linear combination of complex exponentials:*

$$\phi(t) = \sum_{n=1}^{N} c_n \overline{e^{\lambda_n(L-t)}}, \tag{5}$$

where c_n *is the* n*-th element of* $f(\mathbf{A}, \mathbf{B})\overline{h}$*.*

The conclusion of the theorem implies that the SSM parameters $\mathbf{A}$ and $\mathbf{B}$ inherently capture signal characteristics of the input, as well as the hidden state h. This highlights the unique capability of SSMs: they are particularly favorable for reconstructing the input since they encode the input function $\phi(t)$ itself, unlike traditional data-specific neural representation frameworks that were originally designed to capture specific patterns or semantics.

To verify the effectiveness of SSMs in the context of compact data-specific neural representation, we conduct input reconstruction experiments where the input is compressed into an embedding and then reconstructed by a lightweight decoder, which allows us to directly assess how well the encoder captures and retains the information of the input in a compact form. To this end, we first design a simple encoder-decoder architecture that can naturally incorporate various SSMs (Fig. 1). The encoder consists of an SSM block computing the output signal by Eq. (1) and a single convolutional layer for downsampling the signal, while the decoder is composed of a deconvolutional layer (Noh et al., 2015) for upsampling. Note that the SSM block is attached directly onto the raw input and that the downsampling operation of the encoder is essential for compression due to the length-preserving nature of SSMs (Sec. 2). We consider the established SSM architectures, S4 (Gu et al., 2021a), S4D (Gu et al., 2022a), S4ND (Nguyen et al., 2022), S5 (Smith et al., 2022) and Mamba (Gu & Dao, 2023), as candidates for the SSM block.

For evaluation, we train and evaluate this baseline architecture coupled with the diverse SSMs on 1K randomly sampled images from CIFAR-100 (Krizhevsky et al., 2009). Each model is trained for 300 epochs on each image, adhering to the prevailing practice in data-specific neural representations that focuses on optimizing a lightweight model to facilitate overfitting on a single sample (Dupont et al., 2021; Chen et al., 2021). Since the SSMs are designed to operate on 1D sequences (except for S4ND (Nguyen et al., 2022)), we preprocess the input images by flattening the 32×32 pixel grid before passing it through the SSMs.

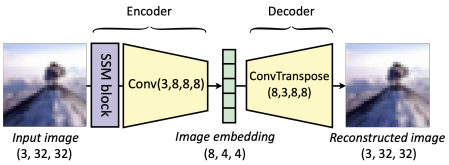
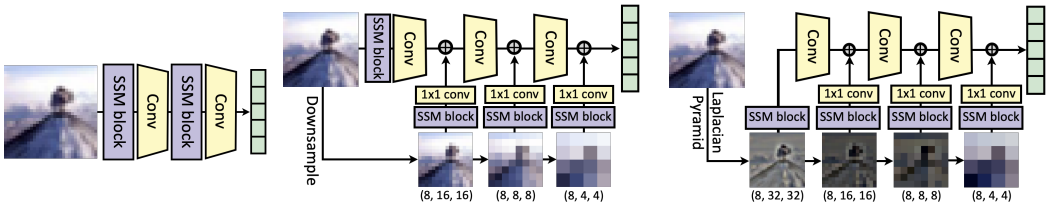


Figure 1: Baseline architecture incorporating SSMs for image reconstruction

Table 1: Image reconstruction quality in PSNR of different architectures incorporating various SSMs. (a), (b), and (c) indicate the encoder variants shown in Fig. 2.

SSM Block	Baseline	(a)	(b)	(c)
Transformer (Vaswani et al., 2017)	24.75	24.87	23.96	24.67
S4 (Gu et al., 2021a)	25.79	24.65	24.68	25.82
S4D (Gu et al., 2022a)	25.49	24.99	25.48	26.06
S4ND (Nguyen et al., 2022)	26.00	25.25	25.75	26.61
S5 (Smith et al., 2022)	25.76	24.84	24.90	26.44
Mamba (Gu & Dao, 2023)	24.90	24.82	24.78	26.58



(a) Stacked (b) Image pyramid (c) Laplacian pyramid

Figure 2: Encoder variants incorporating SSMs for image reconstruction

We report experimental results using the baseline architecture in the 'Baseline' column of Table 1. For comparison, we adopt a Transformer (Vaswani et al., 2017) in place of the SSM block, as it is a widely used architecture for data-specific representation (Yan et al., 2024; Mentzer et al., 2022; Liu et al., 2023; Lu et al., 2021) and has demonstrated strong performance. 'Transformer' indicates a single multi-head attention layer used in this context. The results show that, under the baseline architecture, every SSM consistently outperforms the transformer in reconstruction quality. This finding is *not* trivial, particularly considering that transformers are widely recognized for their superior performance when operating on short token sequences (Gu & Dao, 2023). This suggests that for input reconstruction, the transformer's ability to compute semantic relationships between tokens is less beneficial than in other tasks, while the input function modeling property of SSMs proves to be more advantageous, as discussed in Sec. 2.

3.2 Exploring architectures incorporating SSMs

To explore encoder architectures that better leverage SSMs, we experiment with several design variants. Given our focus on evaluating how well the encoder compresses the input into an embedding, we maintain a fixed decoder architecture across all configurations. We first evaluate a stacked architecture where SSM and convolutional layers alternate to form a deep network (Fig. 2(a)), and observe consistent performance drop for SSM models (Table 1(a)). We hypothesize that this decline stems from the way SSMs encode input features (Sec. 3.1): since SSMs project the input onto implicitly parameterized basis functions and stacking them results in multiple layers of such projections, repeated projection amplifies artifacts and limits the achievable reconstruction rate, analogous to generation loss from information theory (Cover, 1999). Figure 3 supports this explanation: the feature maps from stacked SSM layers show clear block-like artifacts, and these distortions persist into the reconstructed output. The consistency between feature-level and output-level artifacts aligns with our hypothesis that stacking SSM projections accumulates errors, ultimately limiting the achievable reconstruction quality.

To address this, we introduce an 'Image pyramid' variant (Fig. 2(b)), where SSMs are applied at multiple resolutions of the input. This approach improves performance (Table 1(b)), as it enables the use of representations across different scales and increases model capacity without having to stack SSM blocks. While Mamba (Gu & Dao, 2023) shows a slight drop in this setting, the overall trend confirms the benefit of incorporating SSMs across multiple resolutions.

We further explore a 'Laplacian pyramid' variant (Fig. 2(c)), a widely used decomposition method in traditional compression techniques (Burt & Adelson, 1987; Richardson, 2011). Since a Laplacian pyramid introduces less redundancy across scales than the 'Image pyramid' variant, the separate SSM blocks can be more effectively utilized. Fig. 4 4 illustrates this claim: the outputs of SSM blocks in the 'Laplacian pyramid' variant differ meaningfully across scales, whereas those from the 'Image

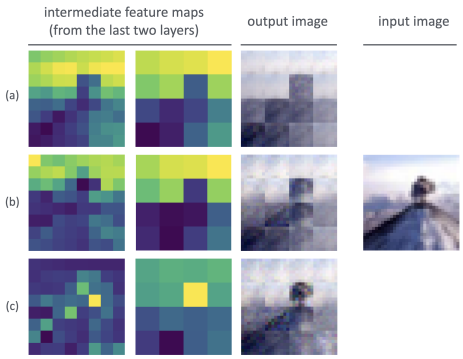


Figure 3: Feature map visualizations of three encoder variants explored in Sec 3.2. (a), (b) and (c) correspond to each variant in Fig. 2.

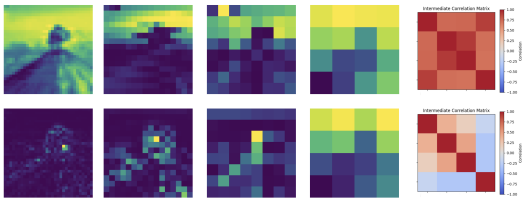


Figure 4: SSM block output of Image pyramid (top) and Laplacian pyramid (bottom) variants. The rightmost column shows correlation matrix between these maps.

pyramid' variant exhibit noticeable redundancy. Results in Table 1(c) show consistent gains, with SSM-based models benefitting the most.

From these image reconstruction experiments, we outline key insights on incorporating SSMs: (1) Stacking SSMs in the encoding process does not yield effective results, (2) attaching SSMs to downsampled images to provide intermediate multi-scale features proves advantageous, (3) employing Laplacian pyramid decomposition further enhances performance.

4 Proposed Method

This section presents our method for data-specific neural representations using SSMs. We introduce our novel module, structured state-space kernel, which addresses the two major limitations of applying SSMs to neural representations of visual data: (1) their design for one-dimensional signals, which does not align directly with visual data, and (2) their inability to compress input sequences due to their length-preserving nature, which typically necessitates additional components for downsampling. Our module overcomes these challenges by leveraging structured kernels derived from SSMs, enabling efficient encoding and reconstruction of visual data.

4.1 Structured State-Space Kernel

To implement the continuous-time dynamics of SSMs on a discrete sequence, the state update of Eq. (1) is often approximated on discretized intervals using a step size parameter Δ. For instance, one can apply the hidden state update between $h(x_i)$ and $h(x_{i-1})$ using the Euler method:

$$\begin{aligned} h(x_i) &\approx h(x_{i-1}) + \Delta h'(x_{i-1}) \\ &= h(x_{i-1}) + \Delta(\mathbf{A}h(x_{i-1}) + \mathbf{B}\phi(x_i)) \\ &:= \bar{\mathbf{A}}h(x_{i-1}) + \bar{\mathbf{B}}\phi(x_i). \end{aligned} \tag{6}$$

Depending on the choice of the discretization method and Δ, the way $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are constructed may vary. Eq. (6) can be expressed as a convolution, where the hidden states evolve according to:

$$\begin{aligned} h_{-1} &= 0, \quad h_0 = \bar{\mathbf{A}}h_{-1} + \bar{\mathbf{B}}\phi_0 = \bar{\mathbf{B}}\phi_0, \quad h_1 = \bar{\mathbf{A}}h_0 + \bar{\mathbf{B}}\phi_1 = \bar{\mathbf{A}}\bar{\mathbf{B}}\phi_0 + \bar{\mathbf{B}}\phi_1, \quad \cdots \\ h_{L-1} &= \bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}\phi_0 + \bar{\mathbf{A}}^{L-2}\bar{\mathbf{B}}\phi_1 + \cdots + \bar{\mathbf{A}}\bar{\mathbf{B}}\phi_{L-2} + \bar{\mathbf{B}}\phi_{L-1} \\ &= \left[\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}\ \bar{\mathbf{A}}^{L-2}\bar{\mathbf{B}}\ \cdots\ \bar{\mathbf{A}}\bar{\mathbf{B}}\ \bar{\mathbf{B}}\right]\left[\phi_0\ \phi_1 \cdots \phi_{L-1}\right]^\top, \end{aligned} \tag{7}$$

where we denote $h(x_i)$ and $\phi(x_i)$ as h_i and ϕ_i, respectively, for brevity. Given that the last hidden state is the projection of the entire sequence onto the basis functions defined by $\mathbf{A}$ and $\mathbf{B}$ (Sec. 2), some previous work (Gu et al., 2020; 2022b) have demonstrated that the input signal can be reconstructed solely from the last hidden state. Inspired by this idea, we employ h_{L-1} for downsampling, as it effectively compresses the input into a compact representation while allowing for an efficient

operation by skipping the computation of intermediate hidden states, *i.e.*, $\{h_0, \ldots, h_{L-2}\}$. Once we construct the convolutional kernel $\mathbf{K}$ so that

$$\mathbf{K} = [\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}} \quad \bar{\mathbf{A}}^{L-2}\bar{\mathbf{B}} \quad \cdots \quad \bar{\mathbf{A}}\bar{\mathbf{B}} \quad \bar{\mathbf{B}}] \in \mathbb{C}^{N\times L\times C}, \tag{8}$$

we can convolve this kernel to obtain the compressed input. We refer to this kernel as *structured state-space kernel* (S3K), as it is factorized by leveraging the state transition matrix $\bar{\mathbf{A}}$ and the projection matrix $\bar{\mathbf{B}}$. In practice, we adopt the diagonal parameterization of $\mathbf{A}$ (Gu et al., 2022a; Gupta et al., 2022) to ease the computation of the power terms and adopt multi-input multi-output (MIMO) framework (Smith et al., 2022) by letting $\bar{\mathbf{B}} \in \mathbb{C}^{N\times C}$ to handle C channels of input sequence simultaneously. The convolution using this structured kernel acts as a lossy compression mechanism, theoretically allowing the reconstruction of the original input signal using the learned parameters.

Theorem 4.1. *Let $\mathbf{A}$ be diagonalizable over $\mathbb{C}$ with non-zero distinct eigenvalues $\{\lambda_i\}$, and Δ be the step size used for the discretization of $\mathbf{A}$. Given the final hidden state $h \in \mathbb{C}^N$ after applying S3K, there exists a function $R : (\mathbf{A}, \mathbf{B}, h) \mapsto \mathbf{H} \in \mathbb{C}^{1\times N}$ with which one can reconstruct the input sequence as:*

$$R(\mathbf{A}, \mathbf{B}, h)\left[e^{\lambda_i(L\Delta - k\Delta)}\right]_{\substack{i=1,2,\ldots,N\\ k=1,2,\ldots,L}}. \tag{9}$$

This finding further supports the interpretation of S3K as a lossy compression mechanism, where the transformed representation retains sufficient information to reconstruct the original input signal through the learned state-space parameters.

4.2 Extension to multi-dimensional S3K

We extend S3K to multiple dimensions via outer products of multiple independent 1D S3Ks. This follows naturally from the definition of nD basis functions as outer products of 1D basis functions in continuous space (Cheney, 1986; Nguyen et al., 2022). The resulting nD kernel $\mathbf{K}$ now has dimensions $(L^{(1)}, L^{(2)}, \cdots, L^{(n)}, N, C)$, where $\{L^{(i)}\}_{i=1,\cdots,n}$ are spatial dimensions of the kernel, and N and C represent output and input channel dimensions, respectively. This formulation enables nD convolution operations on multi-dimensional inputs using structured kernels, akin to the traditional convolution.

4.3 Enhancing expressivity

While the structured kernel theoretically finds effective basis functions and their corresponding coefficients that represent the input data, practical implementation reveals limited expressivity due to the small number of learnable parameters. To address this, we introduce several modifications to enhance the power of our model as follows.

- Input-adaptive $\mathbf{B}$: Instead of using a fixed kernel, we adopt an adaptive mechanism (Chen et al., 2020; Gu & Dao, 2023) where the kernel parameters depend on the input, allowing dynamic adjustments to diverse signals.
- Real-valued SSM parameters: To improve numerical stability and expressivity, we follow the real parameterization of $\mathbf{A}$ and $\mathbf{B}$, which has shown strong empirical performance in Mamba (Gu & Dao, 2023).
- Subsequent 1×1 convolution layer: We integrate a 1×1 convolution layer to further enhance the representation capacity while allowing the state size N to differ from the output channel size, providing greater flexibility in model architecture design.

These modifications refine S3K into a flexible neural network module with stronger representation capacity, allowing S3K to be effectively integrated into data-specific neural representation frameworks. Additional implementation details can be found in Appendix A.7.

4.4 Model architecture

We illustrate the architecture of our final model in Fig. 5. We follow the design choices of the last encoder variant in Sec. 3.2 using multi-scale signals decomposed by Laplacian pyramid (Fig. 2(c)),

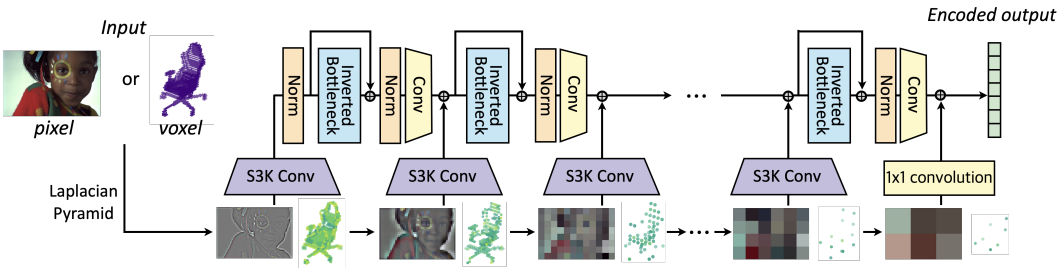


Figure 5: Structure and operation of the proposed LPNet-S3K architecture

and attach an S3K convolution layer, instead of an SSM block and the following convolutional layer, to each level of the Laplacian pyramid. We make two key modifications to complete our model: (1) replacing the intermediate MLP blocks with *inverted bottleneck* layers (Liu et al., 2022), a more advanced module that has shown superior performance across various domains (Woo et al., 2023; Chen et al., 2023; Zhao et al., 2024), and (2) using SiLU (Elfwing et al., 2018) activation and RMSNorm (Zhang & Sennrich, 2019) that have been frequently used in SSMs (Gu & Dao, 2023; Nguyen et al., 2022; Smith et al., 2022). We call this network *Laplacian Pyramid Network with S3K* (LPNet-S3K) for ease of reference.

5 Experiments

To evaluate the effectiveness of the proposed LPNet-S3K architecture, we overfit the network to individual visual inputs across diverse data formats, including image, video, and 3D objects. We first validate the efficacy of the LPNet and S3K architectures in the context of data-specific neural representations by evaluating their performance on images and 3D objects. Then, we evaluate our model on video INR benchmarks (NeRV), demonstrating its efficacy on the standard literature benchmarks.

Datasets. We evaluate our models on standard image and video reconstruction benchmarks. For images, we use Kodak (Kodak, 1993) and CLIC2020 (Toderici et al., 2020), consisting of high-resolution photographs. For videos, we follow common NeRV benchmarks, including Bunny (Roosendaal, 2008), UVG (Mercat et al., 2020), and DAVIS (Pont-Tuset et al., 2017). For 3D data, we randomly sample 1K furniture objects from Objaverse (Deitke et al., 2023). To enable compatibility with our framework, we voxelize each point cloud into a binary voxel grid, forming a cube shaped tensor that encodes the object's geometry. We describe details of each dataset in Appendix A.8.1.

Evaluation metrics. We adhere to the standard evaluation protocols, reporting Peak Signal-to-Noise Ratio (PSNR, in dB) and/or Multi-Scale Structural Similarity Index Measure (MS-SSIM) (Wang et al., 2003) as fidelity metrics across all reconstruction tasks.

Implementation details. For images and 3D objects, we adopt a simple setup by attaching multiple 2D or 3D deconvolutional layers (Noh et al., 2015) on top of LPNet-S3K to reconstruct the input. For videos, we replace the convolutional encoders of HNeRV (Chen et al., 2023), SNeRV (Kim et al., 2024b), and PNeRV-L (Zhao et al., 2024) with LPNet-S3K, denoted as 'Ours-H', 'Ours-S' and 'Ours-P', respectively. Additional implementation details are provided in Appendix A.8.2.

5.1 Quantitative comparisons

Images and 3D objects. Quantitative results are reported in Table 2. To validate the effectiveness of LPNet (Sec. 4.4), the proposed baseline architecture, we construct a ConvNeXt (Liu et al., 2022) variant that follows the same configuration of LPNet, as ConvNeXt is a widely adopted encoder in the literature on data-specific neural representations (Chen et al., 2023; Zhao et al., 2023; 2024; Kim et al., 2024b). For a fair comparison with ConvNeXt, we integrate standard convolutional layers into LPNet, denoted as LPNet-Conv. The performance gap between ConvNeXt and LPNet-Conv

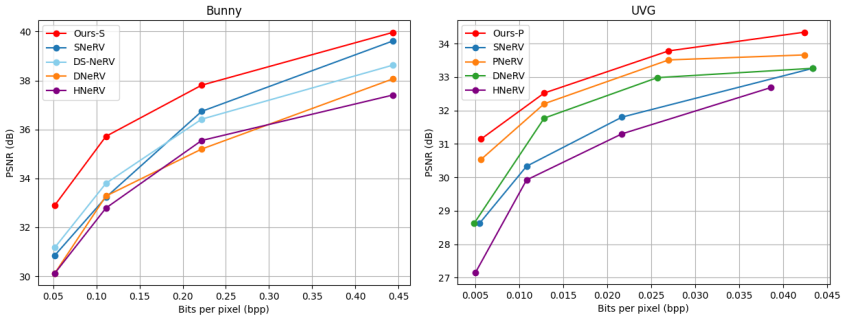


Figure 6: Compression results on Bunny (left) and UVG (right).

Table 2: Comparison between different architectures on Kodak, CLIC2020, and Objaverse. Results reported in PSNR/MS-SSIM.

Method	2D images		3D points
	Kodak	CLIC2020	Objaverse
ConvNeXt (Liu et al., 2022)	25.99/0.8830	24.39/0.8280	17.17/0.7536
LPNet-Conv	27.44/0.9132	25.41/0.8505	17.67/0.7815
LPNet-Mamba	27.51/0.9227	26.16/**0.8694**	17.74/0.7732
LPNet-S3K (Ours)	**28.09/0.9331**	**26.33**/0.8692	**18.34/0.8492**

Table 3: Comparison with the existing NeRV methods on Bunny with different model sizes. Results reported in PSNR.

Model size	0.35M	0.75M	1.5M	3.0M
HNeRV (Chen et al., 2023)	30.15	32.81	35.57	37.43
DNeRV (Zhao et al., 2023)	30.15	33.30	35.22	38.09
DS-NeRV (Yan et al., 2024)	31.20	33.82	36.44	38.65
SNeRV (Kim et al., 2024b)	30.88	33.25	36.76	39.64
Ours-S	**32.93**	**35.74**	**37.83**	**39.99**

Table 4: Comparison with the existing NeRV methods on UVG. '*' indicates results reproduced by official codebases.

Method	Size	Beauty	Bosp.	Honey.	Jockey	Ready.	Shake.	Yacht	Avg.
HNeRV (Chen et al., 2023)	3.0M	33.58	34.73	38.96	32.04	25.74	34.57	29.26	32.70
*DNeRV (Zhao et al., 2023)	3.4M	34.12	35.65	39.22	33.72	28.22	34.80	29.74	33.64
*PNeRV (Zhao et al., 2024)	3.3M	34.18	35.56	**39.80**	31.51	25.94	35.30	30.27	33.22
DS-NeRV (Yan et al., 2024)	3.0M	33.97	35.22	39.56	32.86	27.10	35.04	29.40	33.31
*SNeRV (Kim et al., 2024b)	3.0M	33.76	35.66	38.44	33.78	26.57	35.11	29.65	33.28
Ours-S	3.0M	34.04	36.32	39.51	31.80	27.92	35.54	30.47	33.66
Ours-P	3.3M	**34.22**	**36.54**	38.71	**35.40**	**29.31**	**35.85**	**30.50**	**34.36**

Table 5: Decoding speed comparison on UVG

Method	PSNR	Decoding Speed sec/vid (↓)	FPS (↑)
HNeRV	32.70	1.74	344.83
Ours-H	**32.79**	1.74	344.83
PNeRV	33.22	1.99	301.10
Ours-P	**34.35**	1.99	301.10
SNeRV	33.28	10.08	59.52
Ours-S	**33.66**	10.08	59.52

Table 6: Comparison with the existing NeRV methods on DAVIS

Method	Size	Bike-packing	Blackswan	BMX-trees	Breakdance	Camel	Car-rndabt	Car-shdw	Cows	Dance-twirl	Dog	Avg.
HNeRV (Chen et al., 2023)	3.0M	30.55	30.35	29.98	30.45	26.71	28.61	31.11	24.60	28.60	31.04	29.20
DNeRV (Zhao et al., 2023)	3.4M	30.24	30.92	29.63	30.88	27.38	29.35	31.95	24.88	29.13	31.32	29.57
*PNeRV (Zhao et al., 2024)	3.3M	28.57	29.17	28.77	29.67	27.89	28.76	31.02	24.39	28.16	30.95	28.74
DS-NeRV (Yan et al., 2024)	3.0M	-	32.55	29.76	32.21	27.26	29.48	35.88	25.08	28.79	33.29	-
SNeRV (Kim et al., 2024b)	3.0M	33.29	33.83	31.65	31.40	28.68	31.27	35.79	25.14	30.41	**34.11**	31.56
Ours-S	3.0M	**34.33**	**34.58**	31.72	**33.86**	**30.04**	32.35	**36.69**	**26.47**	30.25	33.21	**33.86**
Ours-P	3.3M	32.15	34.17	**32.48**	33.15	29.04	**32.38**	32.35	25.94	**30.94**	33.97	32.25

in Table 2 highlights the superiority of the LPNet architecture. We then compare LPNet-Conv and LPNet-S3K to assess the contribution of S3K convolution; LPNet-S3K outperformed LPNet-Conv across all benchmarks as shown in the table, underscoring the benefits of SSMs for data-specific neural representations. To assess the distinct advantage of S3K, we also experiment with Mamba (Gu & Dao, 2023) as an alternative SSM (LPNet-Mamba). Specifically, we switch the S3K convolutions with Mamba followed by a standard convolutional layer. When comparing the two, LPNet-S3K consistently outperforms LPNet-Mamba, showing its effectiveness as a compression-specialized SSM. Due to space constraints, additional ablation studies are provided in Appendix A.8.3. It is worth noting that the approach introduced by LPNet and S3K is orthogonal to prevalent techniques in data-specific neural representations, such as bit quantization (Kim et al., 2024a; Ladune et al., 2023; Damodaran et al., 2023) or learning image priors from large-scale datasets (Ballé et al., 2016; 2018; Cheng et al., 2020; Strümpler et al., 2022; Catania & Allegra, 2023). This distinction highlights the potential complementarity of our framework: we believe that integrating LPNet-S3K with these existing techniques could further enhance performance and offer new insights into the design of compact and effective data-specific neural representations.

NeRV benchmarks. We evaluate our method on standard NeRV benchmarks, with results shown in Table 3 (Bunny), Table 4 and 5 (UVG), and Table 6 (DAVIS). We include convolution-based NeRV

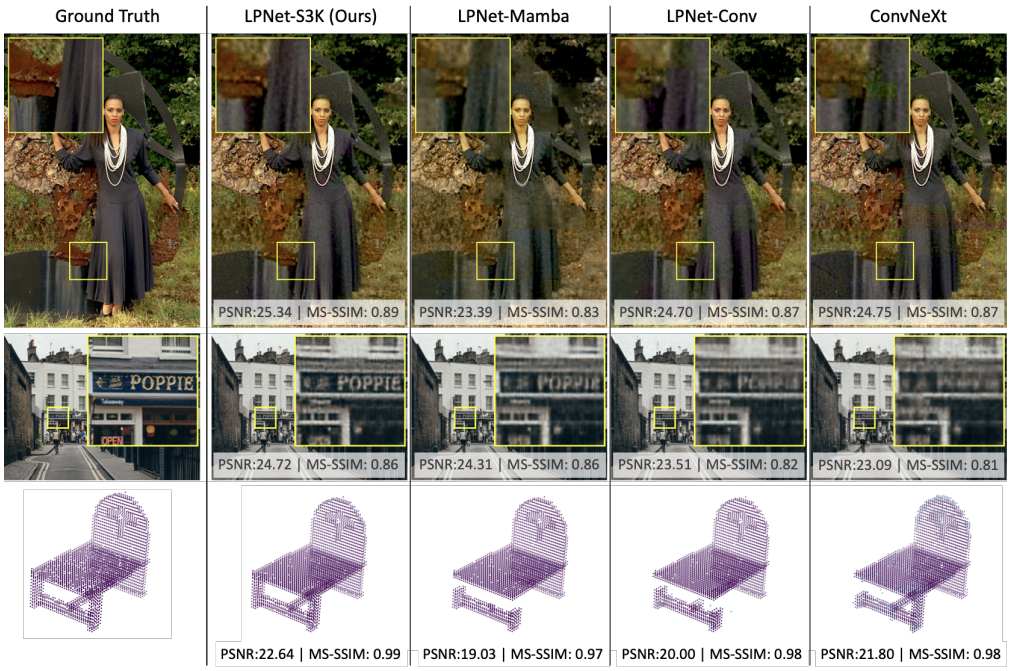


Figure 7: Reconstruction results on images (Kodak, CLIC2020) and voxelized points (Objaverse)

Figure 8: Reconstruction results on a DAVIS video

models as our baselines, following Yan et al. (2024) and Kim et al. (2024b). Additional comparisons to MLP-based methods are provided in Appendix A.8.4. On Bunny, our method ranks the best across all model sizes. On UVG, our method outperforms previous arts and even surpasses models of larger sizes: Ours-S achieves 33.66 PSNR with only 3.0M parameters, exceeding the performance of bigger models like DNeRV and PNeRV. It is noteworthy that our method enhances the performance *while leaving the decoder part unchanged*, ensuring the inference cost remains the same (Table 5). This aspect offers a meaningful advantage in NeRV, since video decoding speed is critical for its real-time streaming applications. On DAVIS, our model ranks either first or second across various videos, surpassing most prior methods. All results are obtained by employing existing decoders, indicating that the performance gains are entirely from our SSM-based encoder; this also implies even greater potential of our method with a dedicated decoder design.

5.2 Qualitative Analysis

We present qualitative comparisons on images and 3D objects in Fig. 7, showing reconstruction results on Kodak (Kodak, 1993), CLIC2020 (Toderici et al., 2020), and Objaverse (Deitke et al., 2023). Across all datasets, our model consistently preserves finer details, such as high-frequency textures in the background (first row) or legible text on signage (second row), and underlying structure

of 3D geometry (third row). We provide video reconstruction results in Fig. 8. Our model shows superior performance despite its smaller model size compared to PNeRV and DNeRV, and effectively preserves fine details, such as a person's hand or typographies on a t-shirt. Additional qualitative results are provided in Appendix Sec. A.8.5.

6 Conclusion and future directions

In this paper, we present the first attempt to link SSMs to data-specific neural representations. To this end, we explore network architectures for effective SSM integration and analyze how different architectures suit the characteristics of SSMs. As a result, we propose S3K, which harnesses the expressiveness of SSMs while enabling natural multi-dimensional data processing and downsampling. These results are theoretically supported and together lead to a novel and powerful data-specific neural representation framework. Our framework achieves strong performance across diverse visual data formats, including images, videos and 3D objects, and remains superior on challenging NeRV benchmarks despite not being designed for NeRV.

In the following, we outline potential directions to improve and extend our framework.

1. *Designing a dedicated decoder*: This work focuses primarily on designing the SSM-based encoder, while employing a simple upsampling decoder or decoders from other methods. While our results demonstrate that the encoder alone significantly contributes to the performance improvements, a decoder tailored to the characteristics of the SSM encoded features may further improve the performance of our model.
2. *Reducing encoding complexity*: Constructing an input-sized kernel using state-space model parameters leads to substantial computational overhead: about $20\times$ more memory and $4\times$ more FLOPs than a plain convolution. While effective, this approach may limit scalability. More efficient alternatives—such as avoiding explicit kernel construction through mathematically equivalent formulations (Nguyen et al., 2022; Gu et al., 2021a), or employing hardware-optimized implementations (Gu & Dao, 2023)—could alleviate this burden.
3. *Application to autoencoders*: Although our method is proposed for data-specific neural networks, its compressive property can be exploited to produce compressive representation beyond individual inputs. Note that modern autoencoders used for generative modeling often rely on convolution-based architectures (Black Forest Labs, 2023; Rombach et al., 2022) or signal processing methods (Agarwal et al., 2025). S3K, which aligns closely with both convolution and signal processing principles, could enable compressive autoencoders that encode inputs using fewer tokens.

References

Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, et al. Cosmos world foundation model platform for physical ai. *arXiv preprint arXiv:2501.03575*, 2025. 10

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016. 1, 8

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018. 8, 1

Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2023. 10

Peter J Burt and Edward H Adelson. The laplacian pyramid as a compact image code. In *Readings in computer vision*, pp. 671–679. Elsevier, 1987. 4

Lorenzo Catania and Dario Allegra. Nif: A fast implicit image compression with bottleneck layers and modulated sinusoidal activations. In *Proceedings of the 31st ACM International Conference on Multimedia*, pp. 9022–9031, 2023. 8

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European conference on computer vision*, pp. 333–350. Springer, 2022a. 1

Guo Chen, Yifei Huang, Jilan Xu, Baoqi Pei, Zhe Chen, Zhiqi Li, Jiahao Wang, Kunchang Li, Tong Lu, and Limin Wang. Video mamba suite: State space model as a versatile alternative for video understanding. *arXiv preprint arXiv:2403.09626*, 2024. 1

Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser Nam Lim, and Abhinav Shrivastava. Nerv: Neural representations for videos. *Advances in Neural Information Processing Systems*, 34:21557–21568, 2021. 1, 3, 10

Hao Chen, Matthew Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. Hnerv: A hybrid neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10270–10279, 2023. 1, 7, 8, 9, 10

Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11030–11039, 2020. 6

Zeyuan Chen, Yinbo Chen, Jingwen Liu, Xingqian Xu, Vidit Goel, Zhangyang Wang, Humphrey Shi, and Xiaolong Wang. Videoinr: Learning video implicit neural representation for continuous space-time super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2047–2057, 2022b. 1

E Ward Cheney. *Multivariate approximation theory: Selected topics*. SIAM, 1986. 6

Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7939–7948, 2020. 1, 8

J Cooley, P Lewis, and P Welch. The finite fourier transform. *IEEE Transactions on audio and electroacoustics*, 17(2):77–85, 1969. 1, 3

Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999. 4

Bharath Bhushan Damodaran, Muhammet Balcilar, Franck Galpin, and Pierre Hellier. Rqat-inr: Improved implicit neural image compression. In *2023 Data Compression Conference (DCC)*, pp. 208–217. IEEE, 2023. 8

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13142–13153, 2023. 2, 7, 9

Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 1, 3

Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022. 1

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018. 7

Cameron Gordon, Shin-Fang Chng, Lachlan MacDonald, and Simon Lucey. On quantizing implicit neural representations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 341–350, 2023. 1

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 2, 3, 4, 6, 7, 8, 10, 1

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020. 1, 2, 5

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021a. 1, 2, 3, 4, 10, 6

Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021b. 1, 2

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022a. 2, 3, 4, 6, 1, 7

Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré. How to train your hippo: State space models with generalized orthogonal basis projections. *arXiv preprint arXiv:2206.12037*, 2022b. 1, 2, 5

Hang Guo, Jinmin Li, Tao Dai, Zhihao Ouyang, Xudong Ren, and Shu-Tao Xia. Mambair: A simple baseline for image restoration with state-space model. In *European Conference on Computer Vision*, pp. 222–241. Springer, 2025. 2

Zongyu Guo, Gergely Flamich, Jiajun He, Zhibo Chen, and José Miguel Hernández-Lobato. Compression with bayesian implicit neural representations. *Advances in Neural Information Processing Systems*, 36:1938–1956, 2023. 1

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022. 2, 6

Vincent Tao Hu, Stefan Andreas Bauma, Ming Gui, Olga Grebenkova, Pingchuan Ma, Johannes Fischer, and Bjorn Ommer. Zigma: Zigzag mamba diffusion model. *arXiv preprint arXiv:2403.13802*, 2024. 1

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017. 7

Hyunjik Kim, Matthias Bauer, Lucas Theis, Jonathan Richard Schwarz, and Emilien Dupont. C3: High-performance and low-complexity neural compression from a single image or video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9347–9358, 2024a. 8, 10

Jina Kim, Jihoo Lee, and Je-Won Kang. Snerv: Spectra-preserving neural representation for video. In *European Conference on Computer Vision*, pp. 332–348. Springer, 2024b. 7, 8, 9, 10

Subin Kim, Sihyun Yu, Jaeho Lee, and Jinwoo Shin. Scalable neural video representations with learnable positional features. *Advances in Neural Information Processing Systems*, 35:12718–12731, 2022. 9, 10

Eastman Kodak. Kodak lossless true color image suite (photocd pcd0992). *URL http://r0k.us/graphics/kodak*, 6:2, 1993. 2, 7, 9

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 3

Ho Man Kwan, Ge Gao, Fan Zhang, Andrew Gower, and David Bull. Hinerv: Video compression with hierarchical encoding-based neural representation. *Advances in Neural Information Processing Systems*, 36, 2024. 9, 10

Théo Ladune, Pierrick Philippe, Félix Henry, Gordon Clare, and Thomas Leguay. Cool-chic: Coordinate-based low complexity hierarchical image codec. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 13515–13522, 2023. 8

Serge Lang. *Algebra*, volume 211. Springer Science & Business Media, 2012. 3

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Learning to generalize: Meta-learning for domain generalization. In *AAAI*, 2018. 1

Kunchang Li, Xinhao Li, Yi Wang, Yinan He, Yali Wang, Limin Wang, and Yu Qiao. Videomamba: State space model for efficient video understanding. In *European Conference on Computer Vision*, pp. 237–255. Springer, 2025. 1

Shufan Li, Harkanwar Singh, and Aditya Grover. Mamba-nd: Selective state space modeling for multi-dimensional data. In *European Conference on Computer Vision*, pp. 75–92. Springer, 2024. 1

Dingkang Liang, Xin Zhou, Wei Xu, Xingkui Zhu, Zhikang Zou, Xiaoqing Ye, Xiao Tan, and Xiang Bai. Pointmamba: A simple state space model for point cloud analysis. *arXiv preprint arXiv:2402.10739*, 2024. 1

Jinming Liu, Heming Sun, and Jiro Katto. Learned image compression with mixed transformer-cnn architectures. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14388–14397, 2023. 4

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*, pp. 2713–2722, 2024. 9, 10

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11976–11986, 2022. 7, 8

Ming Lu, Peiyao Guo, Huiqing Shi, Chuntong Cao, and Zhan Ma. Transformer-based image compression. *arXiv preprint arXiv:2111.06707*, 2021. 4

Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7210–7219, 2021. 1

Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4394–4402, 2018. 1

Fabian Mentzer, George Toderici, David Minnen, Sung-Jin Hwang, Sergi Caelles, Mario Lucic, and Eirikur Agustsson. Vct: A video compression transformer. *arXiv preprint arXiv:2206.07307*, 2022. 1, 4

Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pp. 297–302, 2020. 2, 7

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. *Communications of the ACM*, 65(1):99–106, 2021. 1

Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4nd: Modeling images and videos as multidimensional signals with state spaces. *Advances in neural information processing systems*, 35:2846–2861, 2022. 3, 4, 6, 7, 10, 1

Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 3, 7, 8

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

Federico Perazzi, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 724–732, 2016. 2

Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv preprint arXiv:1704.00675*, 2017. 7

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10318–10327, 2021. 1

B Rao. Sensitivity analysis of state space methods in spectrum estimation. In *ICASSP'87. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 12, pp. 1517–1520. IEEE, 1987. 2

Bhaskar D Rao and KS Arun. Model based processing of signals: A state space approach. *Proceedings of the IEEE*, 80(2):283–309, 1992. 2

Iain E Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011. 1, 4

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022. 10

Ton Roosendaal. Big buck bunny. In *ACM SIGGRAPH ASIA 2008 computer animation festival*, pp. 62–62. 2008. 2, 7

Jiacheng Ruan and Suncheng Xiang. Vm-unet: Vision mamba unet for medical image segmentation. *arXiv preprint arXiv:2402.02491*, 2024. 1

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*, pp. 18507–18516, 2023. 1

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016. 8

Seungjun Shin, Suji Kim, and Dokwan Oh. Efficient neural video representation with temporally coherent modulation. In *European Conference on Computer Vision*. Springer, 2024. 9

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 1, 9, 10

Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10753–10764, 2021. 1

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022. 1, 2, 3, 4, 6, 7, 8

Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. In *European Conference on Computer Vision*, pp. 74–91. Springer, 2022. 1, 8

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

George Toderici, Wenzhe Shi, Radu Timofte, Lucas Theis, Johannes Balle, Eirikur Agustsson, Nick Johnston, and Fabian Mentzer. Workshop and challenge on learned image compression (clic2020). In *CVPR*, 2020. 2, 7, 9

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 4, 6

Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pp. 1398–1402. Ieee, 2003. 7

Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16133–16142, 2023. 7

Zhaohu Xing, Tian Ye, Yijun Yang, Guang Liu, and Lei Zhu. Segmamba: Long-range sequential modeling mamba for 3d medical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 578–588. Springer, 2024. 1

Dejia Xu, Peihao Wang, Yifan Jiang, Zhiwen Fan, and Zhangyang Wang. Signal processing for implicit neural representations. *Advances in Neural Information Processing Systems*, 35:13404–13418, 2022. 1

Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 1

Hao Yan, Zhihui Ke, Xiaobo Zhou, Tie Qiu, Xidong Shi, and Dadong Jiang. Ds-nerv: Implicit neural video representation with decomposed static and dynamic codes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23019–23029, 2024. 4, 8, 9, 10

Yibo Yang, Robert Bamler, and Stephan Mandt. Improving inference for neural image compression. *Advances in Neural Information Processing Systems*, 33:573–584, 2020a. 1

Yibo Yang, Robert Bamler, and Stephan Mandt. Variational bayesian quantization. In *International Conference on Machine Learning*, pp. 10670–10680. PMLR, 2020b. 1

Yijun Yang, Zhaohu Xing, and Lei Zhu. Vivim: a video vision mamba for medical video object segmentation. *arXiv preprint arXiv:2401.14168*, 2024. 1

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019. 7

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 1

Yunfan Zhang, Ties Van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. Implicit neural video compression. *arXiv preprint arXiv:2112.11312*, 2021. 1

Qi Zhao, M Salman Asif, and Zhan Ma. Dnerv: Modeling inherent dynamics via difference neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2031–2040, 2023. 7, 8, 10

Qi Zhao, M Salman Asif, and Zhan Ma. Pnerv: Enhancing spatial consistency via pyramidal neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19103–19112, 2024. 7, 8, 10

Lei Zhou, Chunlei Cai, Yue Gao, Sanbao Su, and Junmin Wu. Variational autoencoder for low bit-rate image compression. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 2617–2620, 2018. 1

Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024a. 1

Qinfeng Zhu, Yuanzhi Cai, Yuan Fang, Yihan Yang, Cheng Chen, Lei Fan, and Anh Nguyen. Samba: Semantic segmentation of remotely sensed images with state space model. *Heliyon*, 10(19), 2024b. 1

A APPENDIX

This material provides proofs for the theorems in the main paper (*i.e.*, Theorem 4.1 and Theorem 5.1), and additional details omitted in the manuscript due to the space constraint.

A.1 RELATED WORK

State-space models. SSMs are a family of sequence-to-sequence models that embeds historical data in state-space representation, using differential equations that involve hidden states and sequential inputs. HiPPO (Gu et al., 2020), an early state-space model, treats an input sequence as samples taken from a continuous function. This function is then approximated using a predefined set of orthogonal polynomials, with their coefficients being dynamically updated by the incoming sequential inputs. LSSL (Gu et al., 2021b) generalizes HiPPO by replacing HiPPO parameters into learnable ones, while still retaining its ability to continuously remember and store the history of observed tokens. It has developed into S4 (Gu et al., 2021a; 2022a; Nguyen et al., 2022) and S5 (Smith et al., 2022), which addresses technical inefficiencies of the previous work. Recently, Mamba (Gu & Dao, 2023) has been introduced as a state-space model that adapts its parameters based on the input sequence. Although originally designed for sequential inputs, this model has inspired various adaptations across different visual perception tasks, including images (Zhu et al., 2024a; Hu et al., 2024; Ruan & Xiang, 2024; Zhu et al., 2024b; Nguyen et al., 2022), videos (Li et al., 2025; Yang et al., 2024; Chen et al., 2024; Li et al., 2024), and 3D scenes (Liang et al., 2024; Xing et al., 2024). Although these efforts extend SSMs to multi-dimensional inputs, they primarily target sequence modeling tasks such as classification or sequence-to-sequence translation. In contrast, our work investigates SSMs as compact representations through compression of input data, a perspective that has received comparatively less attention.

Implicit neural representations. INRs aim at constructing a model that effectively captures continuous signals, including 3D scenes (Park et al., 2019; Mildenhall et al., 2021), images (Strümpler et al., 2022; Guo et al., 2023) and videos (Zhang et al., 2021; Chen et al., 2021). INR typically represents a continuous signal by parameterizing a field, which involves mapping between the coordinate space and the signal space. The emergence of INRs has rapidly advanced the field of data-specific neural representations, offering promising avenues for efficient compression (Strümpler et al., 2022), continuous signal modeling (Martin-Brualla et al., 2021; Chen et al., 2022a), and task-specific adaptation (Pumarola et al., 2021; Chen et al., 2021). Given this shared objective of data-specific modeling, we evaluate our method on a standard INR benchmark, highlighting its potential as a new architectural direction within the INR paradigm.

Neural compression. This line of work enables neural networks to learn compact representations of images and videos by incorporating advanced techniques such as entropy modeling (Ballé et al., 2018; Cheng et al., 2020) or quantization (Yang et al., 2020a;b). Early methods introduce an autoencoder-style architecture (Ballé et al., 2016; Mentzer et al., 2018), where the input is encoded into a compressed latent vector and subsequently reconstructed by the decoder. Our method adopts a similar encoder-decoder formulation while introducing state-space models (SSMs) as a new architectural component for learning compact representations, highlighting the unexplored potential of SSMs in neural compression.

A.2 SOLVING THE LINEAR DIFFERENTIAL EQUATION

We solve the linear differential equation of the state-space model (Eq. (1)), and derive its solution in the form of Eq. (2). As the derivation in this section depends on the original state-space formulation, we first restate Eq. (1) from the main paper for reference:

$$h'(x) = \mathbf{A}h(x) + \mathbf{B}\phi(x). \tag{10}$$

We start by solving a homogeneous first-order matrix ordinary differential equation (ODE), $h'_h(x) = \mathbf{A}h_h(x)$, which is a standard matrix ODE. Its solution is $h_h(x) = e^{\mathbf{A}x}C$, where $C \in \mathbb{C}^N$ is a constant vector determined by initial conditions. Allowing the constant C to vary with x, *i.e.*, $C := u(x)$, derives the particular solution of the form $h(x) = e^{\mathbf{A}x}u(x)$. Plugging this to Eq. (10) yields:

$$\frac{d}{dx}e^{\mathbf{A}x}u(x) = \mathbf{A}h(x) + \mathbf{B}\phi(x) \tag{11}$$

$$\Rightarrow \mathbf{A}e^{\mathbf{A}x}u(x) + e^{\mathbf{A}x}u'(x) = \mathbf{A}e^{\mathbf{A}x}u(x) + \mathbf{B}\phi(x), \tag{12}$$

and canceling the terms gives:

$$e^{\mathbf{A}x}u'(x) = \mathbf{B}\phi(x) \tag{13}$$

$$\Rightarrow u'(x) = e^{-\mathbf{A}x}\mathbf{B}\phi(x) \tag{14}$$

$$\Rightarrow u(x) = \int_0^x e^{-\mathbf{A}\tau}\mathbf{B}\phi(\tau)d\tau + C'. \tag{15}$$

For practical implementation, since we set $h(0) = u(0) = 0$, we can set $C' = 0$. Thus, $h(x) = e^{\mathbf{A}x}u(x)$ becomes:

$$h(x) = e^{\mathbf{A}x}u(x) \tag{16}$$

$$= e^{\mathbf{A}x}\int_0^x e^{-\mathbf{A}\tau}\mathbf{B}\phi(\tau)d\tau \tag{17}$$

$$= \int_0^x e^{(x-\tau)\mathbf{A}}\mathbf{B}\phi(\tau)d\tau, \tag{18}$$

which matches the target expression (Eq. (2)).

A.3 Proof for Theorem 3.1

Theorem 3.1. *Let $\mathbf{A}$ be diagonalizable over $\mathbb{C}$ with non-zero distinct eigenvalues $\{\lambda_i\}$. Given $\mathbf{A}$, $\mathbf{B}$, and the hidden state h computed by Eq. (2), there exists a function $f : (\mathbf{A}, \mathbf{B}) \mapsto \mathbf{F} \in \mathbb{C}^{N\times N}$ with which one can decompose the input function $\phi(t)$ as a linear combination of complex exponentials:*

$$\phi(t) = \sum_{n=1}^{N} c_n \overline{e^{\lambda_n(L-t)}}, \tag{19}$$

where c_n is the n-th element of $f(\mathbf{A}, \mathbf{B})\overline{h}$.

Proof. Since $\mathbf{A}$ is diagonalizable, we can write

$$\xi(\tau, L) = \overline{e^{(L-\tau)\mathbf{A}}\mathbf{B}} \tag{20}$$

$$= \overline{\mathbf{V}e^{(L-\tau)\mathbf{\Lambda}}\mathbf{V}^{-1}\mathbf{B}}, \tag{21}$$

for some matrix $\mathbf{V} \in \mathbb{C}^{N\times N}$ and a diagonal matrix $\mathbf{\Lambda} \in \mathbb{C}^{N\times N}$. Let $\mathbf{b} := \mathbf{V}^{-1}\mathbf{B}$, so that

$$\mathbf{V}^{-1}\mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}. \tag{22}$$

Since $e^{(L-\tau)\mathbf{\Lambda}}$ is diagonal,

$$e^{(L-\tau)\mathbf{\Lambda}}\mathbf{b} = \begin{bmatrix} e^{(L-\tau)\lambda_1}b_1 \\ e^{(L-\tau)\lambda_2}b_2 \\ \vdots \\ e^{(L-\tau)\lambda_N}b_N \end{bmatrix}, \tag{23}$$

and thus, multiplying $\mathbf{V}$ yields:

$$\mathbf{V}e^{(L-\tau)\mathbf{\Lambda}}\mathbf{b} = \mathbf{V}\begin{bmatrix} e^{(L-\tau)\lambda_1}b_1 \\ e^{(L-\tau)\lambda_2}b_2 \\ \vdots \\ e^{(L-\tau)\lambda_N}b_N \end{bmatrix} \tag{24}$$

$$= \begin{bmatrix} \sum_{n=1}^{N} v_{1n}e^{(L-\tau)\lambda_n}b_n \\ \sum_{n=1}^{N} v_{2n}e^{(L-\tau)\lambda_n}b_n \\ \vdots \\ \sum_{n=1}^{N} v_{Nn}e^{(L-\tau)\lambda_n}b_n \end{bmatrix}, \tag{25}$$

where v_{ij} is the (i,j)-th element of the matrix $\mathbf{V}$. Hence, the $\xi_k(\tau, x)$ from Eq. (20) can be expressed as:

$$\xi_k(\tau, L) = \sum_{n=1}^{N} \overline{v_{kn} e^{(L-\tau)\lambda_n} b_n}, \tag{26}$$

$$:= \sum_{n=1}^{N} d_{kn} \overline{e^{\lambda_n (L-\tau)}} \tag{27}$$

for some constant d_{kn}. Hence, it becomes natural to choose sinusoidal bases $e^{i\theta_n(t)} = \overline{e^{\lambda_n(L-t)}}$ and express the input function $\phi(t)$ as:

$$\sum_{n=1}^{N} c_n \overline{e^{\lambda_n (L-t)}}. \tag{28}$$

Note that $\{\overline{e^{\lambda_n(L-t)}}\}$ consists of complex exponentials with N distinct frequencies, which ensures their linear independence (Lang, 2012). Since this property allows them to serve as valid basis functions, we can obtain orthonormal basis functions $\{\psi_n(t)\}$ that span the same functional space as $\{\overline{e^{\lambda_n(L-t)}}\}$. Let the change-of-basis matrix from $\{\psi_n(t)\}$ to $\{\overline{e^{\lambda_n(L-t)}}\}$ defined as $\mathbf{P}$:

$$\mathbf{P} \begin{bmatrix} \psi_1(t) \\ \psi_2(t) \\ \vdots \\ \psi_N(t) \end{bmatrix} = \begin{bmatrix} \overline{e^{\lambda_1(L-t)}} \\ \overline{e^{\lambda_2(L-t)}} \\ \vdots \\ \overline{e^{\lambda_N(L-t)}} \end{bmatrix}. \tag{29}$$

Then, ξ_k from Eq. (27) can be rephrased to:

$$\xi_k(\tau, L) = [d_{k1} \quad d_{k2} \quad \cdots \quad d_{kN}] \begin{bmatrix} \overline{e^{\lambda_1(L-\tau)}} \\ \overline{e^{\lambda_2(L-\tau)}} \\ \vdots \\ \overline{e^{\lambda_N(L-\tau)}} \end{bmatrix}. \tag{30}$$

$$= [d_{k1} \quad d_{k2} \quad \cdots \quad d_{kN}] \mathbf{P} \begin{bmatrix} \psi_1(\tau) \\ \psi_2(\tau) \\ \vdots \\ \psi_N(\tau) \end{bmatrix} \tag{31}$$

$$:= \sum_{n=1}^{N} g_{kn} \psi_n(\tau), \tag{32}$$

where g_{kn} is the inner product between $[d_{k1} \quad d_{k2} \quad \cdots \quad d_{kN}]$ and n-th column of $\mathbf{P}$. Similarly, we can express $\phi(t)$ from Eq. (28) with different bases, *i.e.*, $\sum_{n=1}^{N} m_n \psi_n(t)$, where $[m_1 \quad m_2 \quad \cdots \quad m_N] = [c_1 \quad c_2 \quad \cdots \quad c_N] \mathbf{P}$. Then, plugging $\phi(t) = \sum_{n=1}^{N} m_n \psi_n(t)$ into Eq. (3) gives:

$$h_k = \langle \sum_{n=1}^{N} m_n \psi_n(\tau), \xi_k(\tau, L) \rangle_{[0,L]} \tag{33}$$

$$= \sum_{n=1}^{N} \overline{m_n} \langle \psi_n(\tau), \sum_{j=1}^{N} g_{kj} \psi_n(\tau) \rangle_{[0,L]} \tag{34}$$

$$= \sum_{n=1}^{N} \overline{m_n g_{kn}} = [\overline{g_{k1}} \ \overline{g_{k2}} \ \cdots \ \overline{g_{kN}}] \begin{bmatrix} \overline{m_1} \\ \overline{m_2} \\ \vdots \\ \overline{m_N} \end{bmatrix} \tag{35}$$

$$= [\overline{g_{k1}} \ \overline{g_{k2}} \ \cdots \ \overline{g_{kN}}] \overline{\mathbf{P}}^T \overline{\mathbf{c}}, \tag{36}$$

where $\mathbf{c} = [c_1 \quad c_2 \quad \cdots \quad c_N]^T$.

Let $\mathbf{h} := [h_1 \quad h_2 \quad \cdots \quad h_N]^T$ and $\overline{\mathbf{G}} := [\overline{g_{ij}}]$. Stacking Eq. (36) yields:

$$\mathbf{h} = \overline{\mathbf{GP}}^T \overline{\mathbf{c}} \quad \Leftrightarrow \quad \overline{\mathbf{c}} = (\overline{\mathbf{GP}}^T)^{-1}\mathbf{h} \quad \Leftrightarrow \quad \mathbf{c} = (\mathbf{GP}^T)^{-1}\overline{\mathbf{h}}, \tag{37}$$

which enables us to rewrite $\phi(t)$ by plugging $\mathbf{c}$ to Eq. (28). □

A.4 Proof for Theorem 4.1

Theorem 4.1. *Let $\mathbf{A}$ be diagonalizable over $\mathbb{C}$ with non-zero distinct eigenvalues $\{\lambda_i\}$, and Δ be the step size used for the discretization of $\mathbf{A}$. Given the final hidden state $h \in \mathbb{C}^N$ after applying S3K, there exists a function $R : (\mathbf{A}, \mathbf{B}, h) \mapsto \mathbf{H} \in \mathbb{C}^{1\times N}$ with which one can reconstruct the input sequence as:*

$$R(\mathbf{A}, \mathbf{B}, h) \left[e^{\lambda_i(L\Delta - k\Delta)}\right]_{\substack{i=1,2,\ldots,N \\ k=1,2,\ldots,L}}. \tag{38}$$

Proof. Let $T = L\Delta = x_{L-1}$. Note that

$$h = \int_0^T e^{(T-\tau)\mathbf{A}}\mathbf{B}\phi(\tau)d\tau. \tag{39}$$

The diagonalizability of A gives:

$$e^{(x-\tau)\mathbf{A}} = \mathbf{V}e^{\mathbf{\Lambda}(x-\tau)}\mathbf{V}^{-1}. \tag{40}$$

Let $\mathbf{V}^{-1}\mathbf{B} := \tilde{\mathbf{B}}$ have no zero elements, then

$$\tilde{\mathbf{h}} := \mathbf{V}^{-1}h \tag{41}$$

$$= \int_0^T e^{\mathbf{\Lambda}(T-\tau)}\tilde{\mathbf{B}}\phi(\tau)d\tau \tag{42}$$

$$\Rightarrow c_k := \frac{\tilde{\mathbf{h}}_k}{\tilde{\mathbf{B}}_k} = \int_0^T e^{\lambda_k(T-\tau)}\phi(\tau)d\tau \tag{43}$$

Note that our goal is to recover $\phi(\tau)$ from h. This can be accomplished by finding the *dual basis function* $\{f_k(\tau)\}$ of the basis function $\{e^{\lambda_k(T-\tau)}\}$ for $k \in \{1, 2, \cdots, N\}$. The dual basis function $f_k(\tau)$ satisfies

$$\int_0^T e^{\lambda_i(T-\tau)} f_j(\tau)d\tau = \delta_{ij}, \tag{44}$$

where δ_{ij} is the Kronecker delta. It is worth to note that having $f_k(\tau)$ leads to the expression of $\phi(\tau)$ as:

$$\phi(\tau) = \sum_{k=1}^{N} c_k f_k(\tau), \tag{45}$$

which can be easily shown when plugging Eq. (45) to Eq. (43). Thus, our problem is now converted to finding the dual basis $\{f_k\}$ that corresponds to $\{e^{\lambda_k(T-\tau)}\}$. We start from expressing $f_j(\tau)$ as a linear combination of $e^{\lambda_k(T-\tau)}$:

$$f_j(\tau) = \sum_{k=1}^{N} z_{jk}e^{\lambda_k(T-\tau)} \tag{46}$$

for some z_{jk}. Plugging Eq. (46) to Eq. (44) gives:

$$\int_0^T e^{\lambda_i(T-\tau)} \sum_{k=1}^{N} z_{jk}e^{\lambda_k(T-\tau)}d\tau \tag{47}$$

$$= \sum_{k=1}^{N} z_{jk} \int_0^T e^{(\lambda_i+\lambda_k)(T-\tau)}d\tau = \delta_{ij} \tag{48}$$

If we let the matrix $\mathbf{Z} := \{z_{ik}\}$ and $\mathbf{G} = \{g_{ik}\} = \{\frac{e^{(\lambda_i+\lambda_k)T}-1}{\lambda_i+\lambda_k}\}$, directly solving the integral yields:

$$\sum_{k=1}^{N} z_{jk} \frac{e^{(\lambda_i+\lambda_k)T}-1}{\lambda_i+\lambda_k} = \delta_{ij} \iff \mathbf{ZG} = \mathbf{I}. \tag{49}$$

Hence, we obtain $\mathbf{Z} = \mathbf{G}^{-1}$. According to Eq. (46),

$$\mathbf{f}(\tau) = \mathbf{G}^{-1} \begin{bmatrix} e^{\lambda_1(T-\tau)} \\ e^{\lambda_2(T-\tau)} \\ \vdots \\ e^{\lambda_N(T-\tau)}, \end{bmatrix} \tag{50}$$

and plugging this to Eq. (45) gives:

$$\phi(\tau) = \mathbf{c}^T \mathbf{f}(\tau) = \mathbf{c}^T \mathbf{G}^{-1} \begin{bmatrix} e^{\lambda_1(T-\tau)} \\ e^{\lambda_2(T-\tau)} \\ \vdots \\ e^{\lambda_N(T-\tau)} \end{bmatrix} \tag{51}$$

where $\mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$. If we put everything together,

$$\phi(\tau) = \mathbf{c}^T \mathbf{G}^{-1} \begin{bmatrix} e^{\lambda_1(T-\tau)} \\ e^{\lambda_2(T-\tau)} \\ \vdots \\ e^{\lambda_N(T-\tau)} \end{bmatrix} \tag{52}$$

$$= \left(\frac{\mathbf{V}^{-1}\mathbf{h}}{\mathbf{V}^{-1}\mathbf{B}}\right)^T \mathbf{G}^{-1} \begin{bmatrix} e^{\lambda_1(T-\tau)} \\ e^{\lambda_2(T-\tau)} \\ \vdots \\ e^{\lambda_N(T-\tau)} \end{bmatrix}. \tag{53}$$

We abused element-wise division operation here for simplicity. Evaluating $\phi(\tau)$ at $\tau \in \{0, \Delta, 2\Delta, \cdots, (L-1)\Delta\}$ completes the proof. □

A.5 SSM FOR 1D SIGNAL RECONSTRUCTION

As a natural extension of the reconstruction experiment in Sec. 3.2, we conduct additional studies on 1D signal reconstruction to further validate the effectiveness of SSM-based architectures in capturing signals. We follow the similar experimental setup as described in Sec. 3.2, with the primary modifications being the use of 1D convolutional layers in place of 2D ones and adjustments to the input dimensionality. Specifically, we convert input images into 1D signals by flattening them in a zig-zag manner, transforming the input shape from $(3, 32, 32)$ to $(3, 1024)$. The kernel size of the convolution is maintained to 8, resulting in an encoded embedding that is one-eighth the length of the input. For clarity, we illustrate the modified baseline architecture and its variants in Fig. A1 and Fig. A2, respectively.

The results of the experiment are demonstrated in Table A1. We find the similar tendency we have observed in Sec. 3.2, which suggests that SSMs, when placed in an appropriate reconstruction setting, hold strong potential in signal reconstruction as implied in Sec. 3.1.

A.6 PARAMETER COUNT ANALYSIS

As discussed in Sec. 4.3, the direct application of S3K convolution suffers from limited expressivity, primarily due to an insufficient parameter budget that constrains the network's representational capacity. Table A2 presents how each modification contributes to the parameter count of the 2D

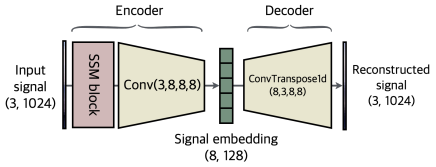
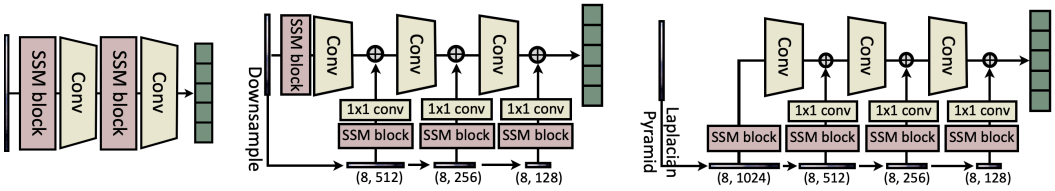


Figure A1: Baseline architecture for 1D signal reconstruction

Table A1: Performance (PSNR) of different architectures incorporating various SSMs for the 1D signal task. (a), (b), and (c) indicate the encoder variants illustrated in Fig. A2.

SSM Block	PSNR			
	Baseline	(a)	(b)	(c)
Transformer (Vaswani et al., 2017)	28.09	27.98	27.27	27.47
S4 (Gu et al., 2021a)	29.81	28.43	28.66	31.20
S4D (Gu et al., 2022a)	28.99	26.24	27.73	29.94
S4ND (Nguyen et al., 2022)	29.98	28.17	28.52	31.05
S5 (Smith et al., 2022)	28.93	26.39	28.81	28.98
Mamba (Gu & Dao, 2023)	30.62	26.78	27.30	30.73



(a) Stacked (b) Signal pyramid (c) Laplacian pyramid

Figure A2: Encoder variants incorporating SSMs for 1D signal reconstruction

Table A2: Parameter count comparison on different variants of S3K 2D convolution.

Method	subsequent 1×1 conv.	Adaptive $\mathbf{B}$	Adaptive Δ	Real $\mathbf{A}, \mathbf{B}$	# param	$C = 3, D = 16,$ $K = 5, N = 16$
Conv2d(C,D,K)	-	-	-	-	$CDK^2 + D$	3088
	✗	✗	✗	✗	$6N + 4NC$	288
	✓	✗	✗	✗	$6N + 4NC + (N+1)D$	560
S3KConv2d	✓	✓	✗	✗	$8N + 2NK + (N+1)D$	656
(C,N,D,K)	✓	✓	✓	✗	$8N + 2NK + K + (N+1)D$	664
	✓	✓	✗	✓	$8N + 2NK + (N+1)D$	624
	✓	✓	✓	✓	$6N + 2NK + K + (N+1)D$	632

convolution using S3K. Following the standard convolutional network notation, Conv2d(C, D, K) denotes a 2D convolutional layer that transforms an input with C channels into D output channels using a kernel of size K. Similarly, S3KConv2d(C, N, D, K) performs the same transformation but introduces an intermediate state of size N to model the structured state-space dynamics. The primary contributor to the high parameter count in standard convolutional networks is the CDK^2 term, which involves the multiplication of four factors and grows rapidly with channel and kernel size. In contrast, S3K layers are designed with more compact parameterization, where the largest terms involve only two multiplicative factors, resulting in significantly fewer parameters. Among all the architectural modifications, the most significant increase in parameter count arises from the subsequent 1×1 convolution, which projects the N-dimensional latent state into the desired size of output feature. On the other hand, the adaptivity of state-space parameters also introduces a relatively modest parameter increase, though the exact impact depends on the choice of state size N.

A.7 Model implementation details

This section provides additional implementation details that were omitted from the main paper due to space constraints.

A.7.1 Initialization of $\mathbf{A}$ and $\mathbf{B}$

We mainly follow the initialization scheme introduced in S5 (Smith et al., 2022) and Mamba (Gu & Dao, 2023). For complex initialization of $\mathbf{A}$ and $\mathbf{B}$ for ablation, we follow HiPPO initialization of $\mathbf{A}$ and use eigenvectors $\mathbf{V}$ from diagonalization of $\mathbf{A}$ for initialization of $\mathbf{B}$, as done in S5 (Smith et al., 2022). For real implementation, we set $\mathbf{\Lambda}$, the N diagonal elements of $\mathbf{A}$, be $\mathbf{\Lambda}_n = -(n+1)$, and employ normal initialization for $\mathbf{B}$.

A.7.2 OPERATIONAL DETAILS OF S3KCONV2D LAYER

We elaborate on operational details of S3KConv2d layer, which naturally extends to S3K convolutions for N-dimensional inputs.

S3KConv2d layer takes the same arguments of the ordinary convolutional layer: input channel dimension C_{in}, output channel dimension C_{out}, and kernel size (K_1, K_2). For simplicity, we assume trivial settings for stride, padding, and dilation. Let the input $X_0 \in \mathbb{R}^{B\times C_0\times H_0\times W_0}$, then we first project input to have C channel dimension by 1×1 convolution: $X \in \mathbb{R}^{B\times C\times H_0\times W_0}$. Now we construct a kernel for each spatial dimension. For clearer explanation, we focus on the input $X_w \in \mathbb{R}^{B\times C\times K_1\times K_2}$, which represents the local window extracted during convolution. The following operations are applied in parallel across all such windows as the kernel slides over the input. Note that in Mamba (Gu & Dao, 2023), a linear layer is applied to the length-L input sequence to obtain the input-adaptive $\mathbf{B}$ of length L. Since the kernel of i-th dimension needs to be a length-K_i 1D kernel, we apply linear layer to the other spatial dimension. Specifically, let $\mathbf{B}^{(i)}_{\text{proj}}(d_{\text{in}}, d_{\text{out}})$ be the linear projection layer that transforms the channel dimension from d_{in} to d_{out} to produce $\mathbf{B}^{(i)}$, $\mathbf{B}$ used to construct the i-th dimension S3K 1D kernel. Then, we set $(d_{\text{in}}, d_{\text{out}}) = (K_2, N)$ for $i = 1$ and (K_1, N) for $i = 2$, so that we obtain $\mathbf{B}^{(i)} \in \mathbb{R}^{B\times C\times K_i\times N}$. With $\mathbf{\Lambda}^{(i)} \in \mathbb{R}^N$ and step size $\Delta^{(i)} \in \mathbb{R}^N$, we discretize $\mathbf{\Lambda}^{(i)}$ and $\mathbf{B}^{(i)}$ using zero-order hold (ZOH) method:

$$\overline{\mathbf{\Lambda}}^{(i)} = e^{\mathbf{\Lambda}^{(i)}\Delta^{(i)}}, \qquad \overline{\mathbf{B}}^{(i)} = (\mathbf{\Lambda}^{(i)})^{-1}(\Delta^{(i)}\overline{\mathbf{\Lambda}}^{(i)} - \mathbf{I})\mathbf{B}^{(i)}, \tag{54}$$

and compute the kernel as in Eq. (7). Since we have kernel $\mathbf{K}^{(i)} \in \mathbb{R}^{B\times C\times K_i}$, we can take outer product of these kernels to construct 2D kernel $\mathbf{K} \in \mathbb{R}^{B\times C\times N\times K_1\times K_2}$, and apply this kernel to the input X_w:

$$X_w^{(\text{out})} \in \mathbb{R}^{B\times N\times 1\times 1}, \qquad \text{where } (X_w^{(\text{out})})_{bn} = \sum_c \sum_{k_2} \sum_{k_1} ((X_w)_{[b,c,k_1,k_2]})\mathbf{K}_{[b,c,n,k_1,k_2]}. \tag{55}$$

We also incorporate gating mechanisms (Gu et al., 2022a; Nguyen et al., 2022; Gu & Dao, 2023) and residual connections for complete implementation, as they have proven effective and are widely adopted as standard components in SSM block designs. Specifically, we project the initial input X_0 to have N channels: $X_{\text{res}} \in \mathbb{R}^{B\times N\times H\times W}$, and 2D average pool with the same kernel size (K_1, K_2) and the stride to produce the tensor $X_{\text{res}}^{(\text{out})}$, matching the output size of the convolution. A SiLU (Elfwing et al., 2018) activation is applied to this residual tensor, which is then used to gate the convolution output via element-wise multiplication. Finally, we add $X_{\text{res}}^{(\text{out})}$ back to the gated output to complete the residual connection.

A.8 EXPERIMENT DETAILS

This section provides additional experiment details that were omitted from the main paper due to space constraints.

A.8.1 DATASETS

Kodak, CLIC2020. The Kodak dataset (Kodak, 1993) is a set of 24 natural photographs of resolution 512×768. The CLIC2020 (Toderici et al., 2020) dataset includes 41 images of varying resolutions, allowing side lengths up to 2048 pixels. Both datasets are commonly used for image compression tasks, as they contain rich high-frequency details and complex scenes.

Bunny, UVG, DAVIS. Bunny (Roosendaal, 2008) is a 132-frame, animated short film, while UVG (Mercat et al., 2020) is a long 1080p video dataset comprising sequences of 300 or 600 frames. Both are widely used benchmarks for video compression. DAVIS (Huang et al., 2017) is a densely annotated featuring short 1080p video clips, commonly used for video segmentation. Since NeRV benchmarks often include a subset of DAVIS, we follow this convention and select the following video clips; 'bike-packing', 'blackswan', 'bmx-trees', 'breakdance', 'camel', 'car-roundabout', 'car-shadow', 'cows', 'dance-twirl', and 'dog'.

Table A3: Implementation details of the experiments from Sec. 5

Dataset	Encoder	Enc. strides	Decoder	Dec. strides	Feature dims.	Learning rate
Kodak	ConvNeXt	[16, 4, 2, 2]	ConvTranspose2D	[4, 4, 4, 2, 2]	[64, 64, 64, 16]	1e-2
	LPNet-Conv	[16, 4, 2, 2]	ConvTranspose2D	[4, 4, 4, 2, 2]	[64, 64, 64, 16]	3e-2
	LPNet-Mamba	[32, 2, 2, 2]	ConvTranspose2D	[4, 4, 4, 2, 2]	[64, 64, 64, 16]	1e-2
	LPNet-S3K	[32, 2, 2, 2]	ConvTranspose2D	[4, 4, 4, 2, 2]	[64, 64, 64, 16]	2e-2
CLIC2020	ConvNeXt	[4, 4, 2]	ConvTranspose2D	[4, 2, 2, 2]	[64, 64, 64, 16]	1e-2
	LPNet-Conv	[4, 4, 2]	ConvTranspose2D	[4, 2, 2, 2]	[64, 64, 64, 16]	5e-3
	LPNet-Mamba	[8, 2, 2]	ConvTranspose2D	[4, 2, 2, 2]	[64, 64, 64, 16]	1e-2
	LPNet-S3K	[8, 2, 2]	ConvTranspose2D	[4, 2, 2, 2]	[64, 64, 64, 16]	1.6e-2
Objaverse	ConvNeXt	[2, 2, 2, 2]	ConvTranspose3D	[2, 2, 2, 2]	[64, 64, 64, 16]	3e-3
	LPNet-Conv	[2, 2, 2, 2]	ConvTranspose3D	[2, 2, 2, 2]	[64, 64, 64, 16]	3e-3
	LPNet-Mamba	[4, 2, 2]	ConvTranspose3D	[2, 2, 2, 2]	[64, 64, 64, 16]	4e-3
	LPNet-S3K	[4, 2, 2]	ConvTranspose3D	[2, 2, 2, 2]	[64, 64, 64, 16]	3e-3
Bunny	LPNet-S3K	[5, 4, 4, 2, 2]	SNeRV (Kim et al., 2024b)	[5, 4, 4, 2, 2]	[64, 64, 64, 16]	3e-4
UVG	LPNet-S3K	[10, 8, 3, 2]	HNeRV (Chen et al., 2023)	[5, 4, 4, 3, 2]	[64, 64, 64, 16]	2e-4
	LPNet-S3K	[10, 8, 3, 2]	SNeRV (Kim et al., 2024b)	[5, 4, 4, 3, 2]	[64, 64, 64, 16]	2e-4
	LPNet-S3K	[10, 8, 3, 2]	PNeRV-L (Zhao et al., 2024)	-	-	2e-4
DAVIS	LPNet-S3K	[10, 8, 3, 2]	SNeRV Kim et al. (2024b)	[5, 4, 4, 3, 2]	[64, 64, 64, 16]	2e-4
	LPNet-S3K	[10, 8, 3, 2]	PNeRV-L Zhao et al. (2024)	-	-	2e-4

Objaverse. Objaverse is a large-scale dataset containing over 800K web-crawled 3D objects, which covers a wide range of functional categories and geometric variations. As one of the largest publicly available collections of 3D assets, it is often used for 3D understanding, neural rendering, shape reconstruction, and vision-language grounding. In our work, we take the 1K 3D objects from '*Furnitures*' subset, which consists of everyday household items such as sofas, chairs, and tables. We voxelize each furniture object to have a binary grid of size 32^3 (approximately 33K voxels), where the voxel occupancy encodes the object's structure.

A.8.2 Implementation details

We provide detailed implementation configurations in Table A3. Since the decoder architecture remains consistent across all experiments, we list the different experimental variants under the 'Encoder' column. The 'Enc. strides' column specifies the strides applied at each encoder stage, indicating the spatial downsampling factor between successive layers. Analogously, 'Dec. strides' indicates the upsampling factor at each decoder stage. PNeRV (Zhao et al., 2024) does not employ conventional upsampling methods such as deconvolution (Noh et al., 2015) or pixelshuffle (Shi et al., 2016), thus we do not specify its decoder strides. We observe that SSM-based encoders perform better when the first encoder layer uses a larger kernel size, while standard convolution-based encoders tend to perform well with comparably more uniform stride settings across layers. Accordingly, we use a larger stride in the first encoder layer for SSM-based models to match this behavior. The 'Feature dims.' column specifies the channel width at each encoder stage, which is kept consistent across all experiments. Since each model has different characteristics, we observe that they require distinct learning rates to achieve optimal performance. Therefore, we carefully search across a range of learning rates for each dataset and report the best results.

A.8.3 Ablation studies

We present an ablation study in Table A4 to evaluate the impact of the architectural modifications introduced in Sec. 4.3 and Sec. 4.4. In addition, we evaluate an alternative design inspired by Mamba (Gu & Dao, 2023), where the step size Δ is made input-adaptive. The results show that each proposed component contributes positively to performance, except for the adaptive Δ: performance drops significantly from 36.92 to 36.26 when Δ is made input-dependent. We hypothesize that this is due to a mismatch between the objective of input-selectivity, a primary reason for adopting adaptive Δ, and input reconstruction. Since input reconstruction demands uniform attention across all regions of the input for accurate reconstruction, the input selectivity introduced by adaptive Δ may be less effective than in other tasks. We also find that the complex initialization of $\mathbf{A}$ and $\mathbf{B}$ (Smith et al., 2022) is equally helpful, aligning with observations in Mamba (Gu & Dao, 2023) that such

Table A4: Ablation study of LPNet-S3K components on the Bunny dataset. We employ HNeRV (Chen et al., 2023) decoder for the experiment.

subsequent 1×1 conv.	Adaptive $\mathbf{B}$	Adaptive Δ	Inverted Bottleneck	Real $\mathbf{A}, \mathbf{B}$	SiLU act.	RMS norm	PSNR
✗	✗	✗	✗	✗	✗	✗	36.37
✓	✗	✗	✗	✗	✗	✗	36.67
✓	✓	✗	✗	✗	✗	✗	36.92
✓	✓	✓	✗	✗	✗	✗	36.26
✓	✓	✗	✓	✗	✗	✗	36.94
✓	✓	✗	✓	✗	✓	✓	37.03
✓	✓	✗	✓	✓	✗	✗	36.92
✓	✓	✗	✓	✓	✓	✗	36.99
✓	✓	✗	✓	✓	✓	✓	37.04

Table A5: Ablation of state size on the Bunny dataset.

state size	PSNR
16	36.74
32	36.88
64	37.03
128	36.92

Table A6: Ablation of state size on the Kodak dataset.

state size	PSNR
4	27.65
8	27.98
16	28.09
32	28.04

Table A7: Ablation of state size on the Objaverse dataset.

state size	PSNR
8	17.56
16	17.98
32	18.34
64	18.29

initialization of $\mathbf{A}$ is aids in processing continuous inputs. However, for performance optimization in terms of speed, we stick to the real parametrization.

We also provide thorough ablations on state size of S3K layers in Table A5-A7. We use a state size of 8 (images), 32 (3D data), 64 (videos), and apply the same state size across all S3K layers for each setting. Empirically, increasing the state size in the first S3K layer meaningfully enhances the performance, likely because this layer is the primary opportunity to capture high-frequency spatial details and a larger state size helps preserve this. In contrast, increasing the state size in later layers yields only marginal gains. For simplicity and consistency, we adopt the first layer's optimal state size for all layers. In our experiments, performance improves as the state size increases but begins to saturate once the state size is sufficiently large relative to the signal complexity. In practice, this occurs around 8 for images, around 32 for 3D data, and around 64 for videos–values we therefore use throughout the model.

A.8.4 Comparison to MLP-based methods

LPNet-S3K is yet inapplicable to coordinate-to-rgb mapping methods (*i.e.*, MLP-based methods) such as Liu et al. (2024); Sitzmann et al. (2020) (images) or Kwan et al. (2024); Kim et al. (2022) (videos), categorizing itself as a *convolution-based* neural representation, in contrast to other mainstream neural representation *MLP-based* neural representation (coordinate-based, INR). Hence, we have compared our methods only with convolution-based methods in Sec. 5.1, as it has been a standard practice to compare models within the same family (Kim et al., 2024b; Yan et al., 2024; Kim et al., 2022; Shin et al., 2024), due to their fundamentally different characteristics which are illustrated in Table A8. Fig. A3 also exhibits their differences clearly: convolution-based models offer very fast decoding speed while sacrificing their reconstruction quality. On the other hand, MLP-based methods in general show higher fidelity, but have slower decoding speed, which limits their real-time application. Unlike the prevailing NeRV trend of trading decoding speed for higher reconstruction quality, LPNet-S3K serves as a fidelity enhancer for convolution-based models without adding inference cost.

We also provide a comparison with MLP-based image neural representation, FINER (Liu et al., 2024) in Table A9. FINER shows better fidelity compared to LPNet-S3K, but requires much more time and memory for training. On the other hand, LPNet-S3K shows greater efficiency, while underperforming in terms of reconstruction quality.

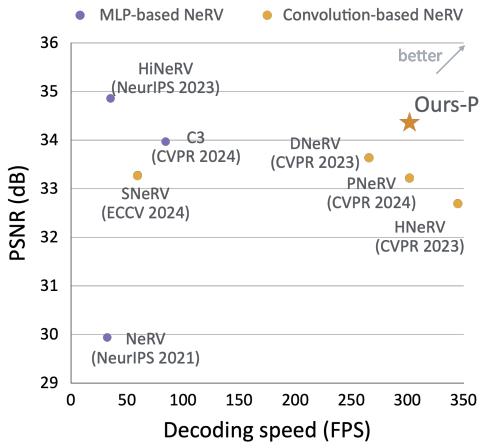


Figure A3: Reconstruction quality and decoding speed trade-off in modern NeRV models. We collect models of 0.02-0.03bpp on a 600-frame UVG video, which occupy 300 epochs to train. FPS is measured with NVIDIA A6000ada GPU.

Table A8: Comparison between MLP-based methods and convolution-based methods

Type	recon. quality	decoding speed	Examples
MLP-based (INR)	high	low	SIREN (Sitzmann et al., 2020) FINER (Liu et al., 2024) HiNeRV (Kwan et al., 2024) NeRV (Chen et al., 2021) NVP (Kim et al., 2022) C3 (Kim et al., 2024a)
Convolution -based	low	high	HNeRV (Chen et al., 2023) DNeRV (Zhao et al., 2023) PNeRV (Zhao et al., 2024) DS-NeRV (Yan et al., 2024) SNeRV (Kim et al., 2024b) Ours

Table A9: Comparison between LPNet-S3K to FINER.

Method	Epochs	Num params	Training time (s/epoch)	Memory (MB)	Kodak	CLIC
LPNet-S3K	300	150K	0.05	35210	28.09	26.33
FINER (Liu et al., 2024)	300	199K	0.4	4106	28.68	27.12

A.8.5 Additional qualitative results

We provide additional qualitative results in Fig. A4 (Kodak), Fig. A5 (CLIC2020), Fig. A6 (Objaverse), and Fig. A7-A12 (DAVIS).

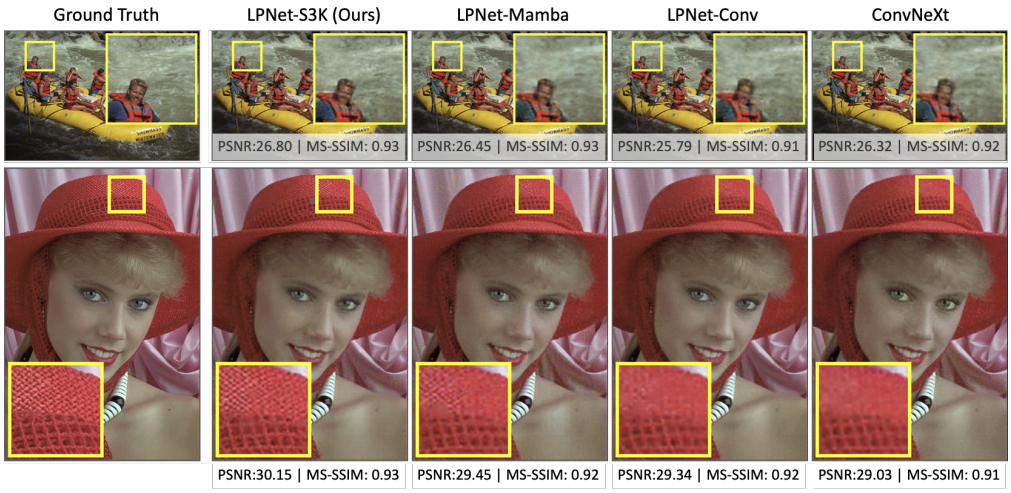


Figure A4: Reconstruction results on the Kodak dataset

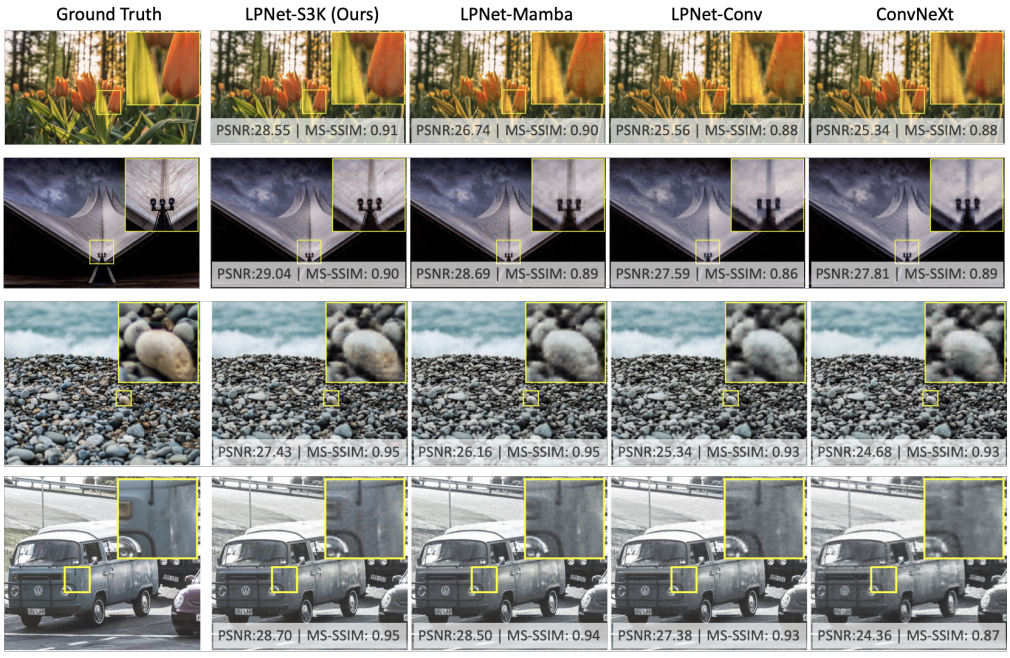


Figure A5: Reconstruction results on the CLIC2020 dataset

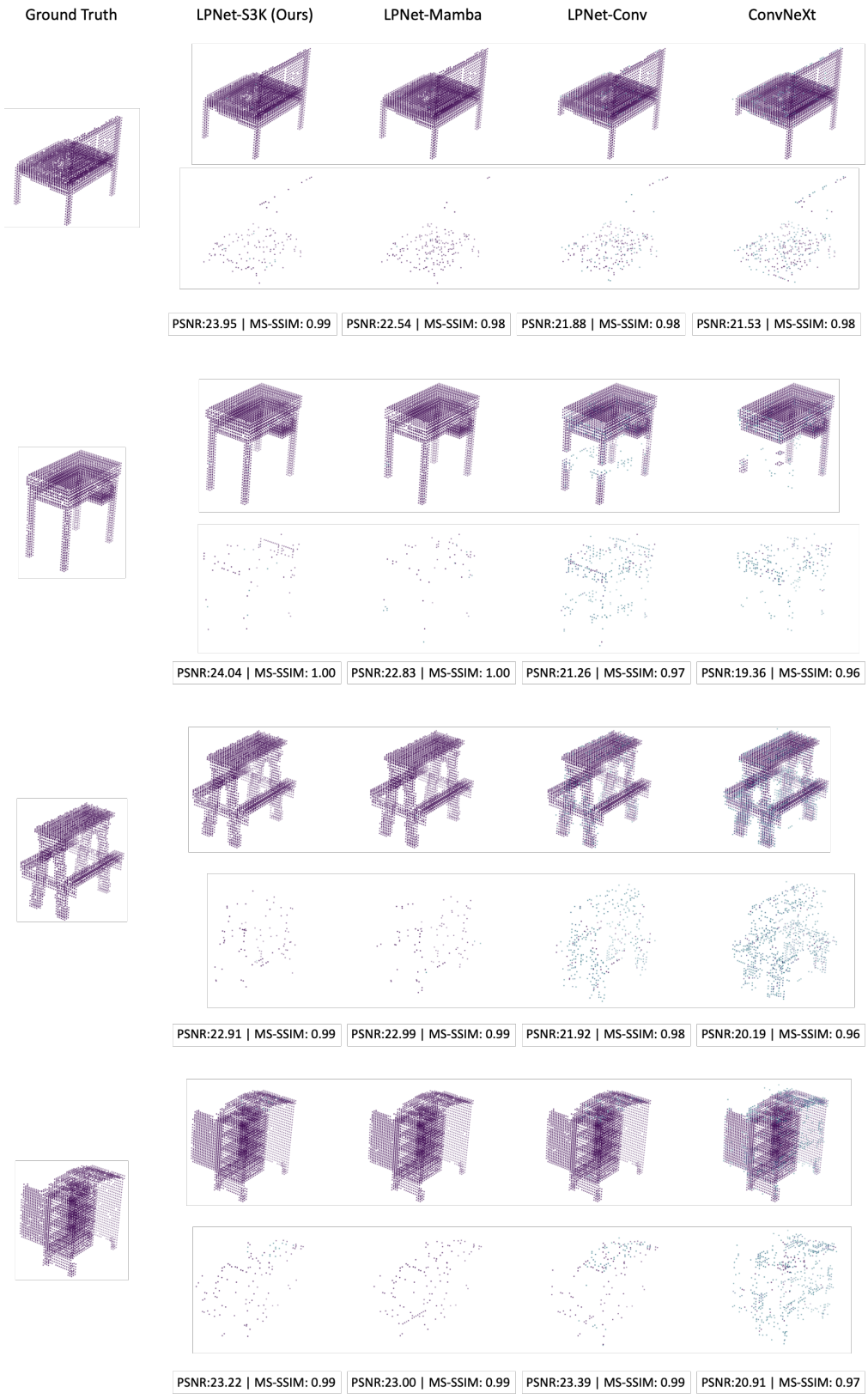


Figure A6: Reconstruction results on the Objaverse dataset. For clearer comparison, we present the difference visualization under each reconstruction result.

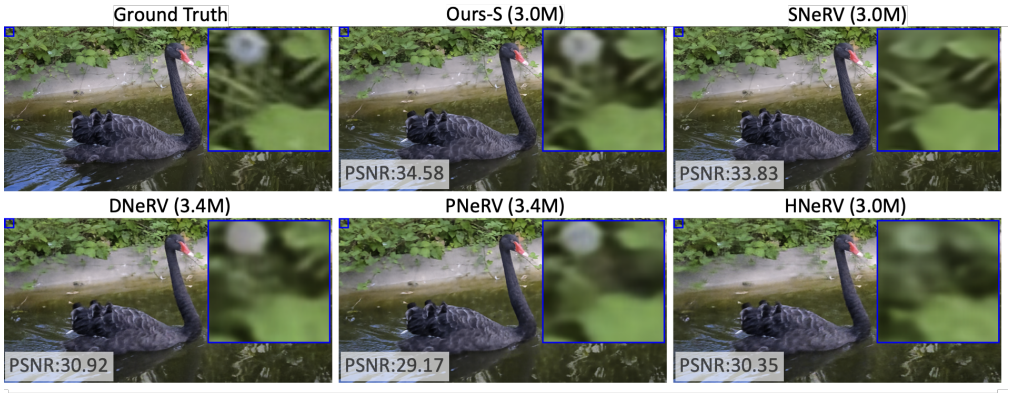


Figure A7: Reconstruction results on 'blackswan' from the DAVIS dataset

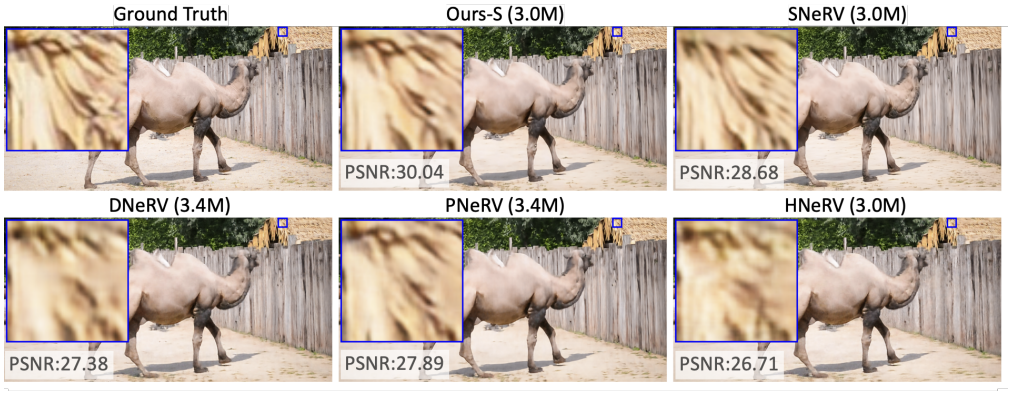


Figure A8: Reconstruction results on 'camel' from the DAVIS dataset

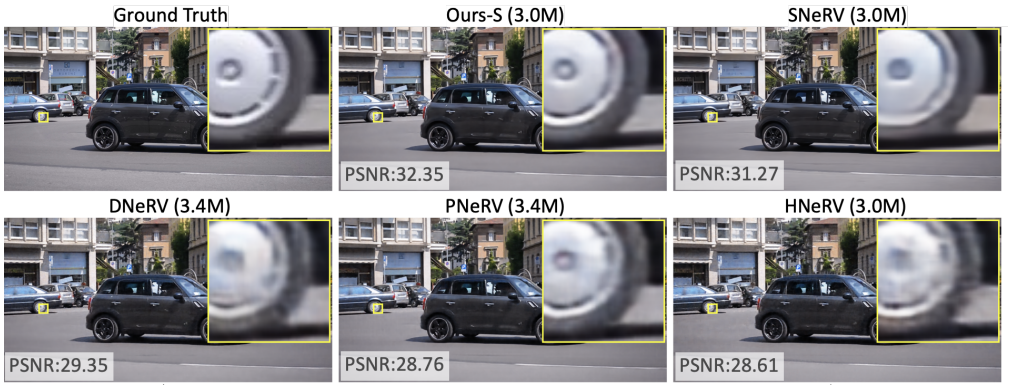


Figure A9: Reconstruction results on 'car-roundabout' from the DAVIS dataset

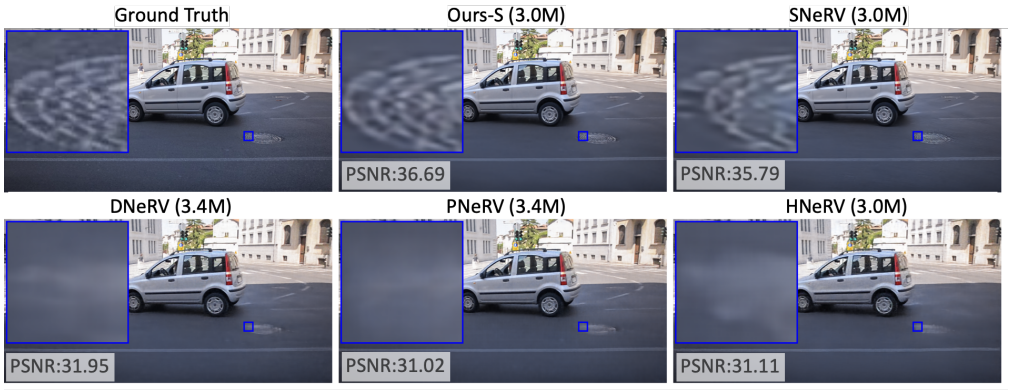


Figure A10: Reconstruction results on 'car-shadow' from the DAVIS dataset

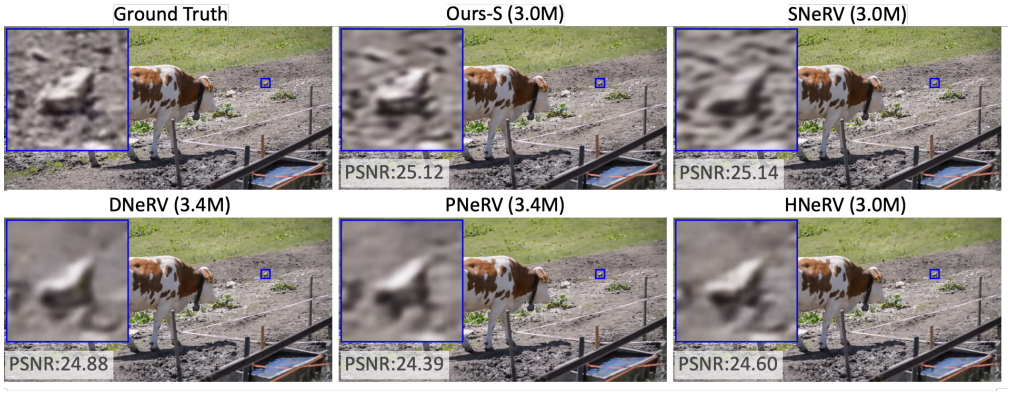


Figure A11: Reconstruction results on 'cow' from the DAVIS dataset

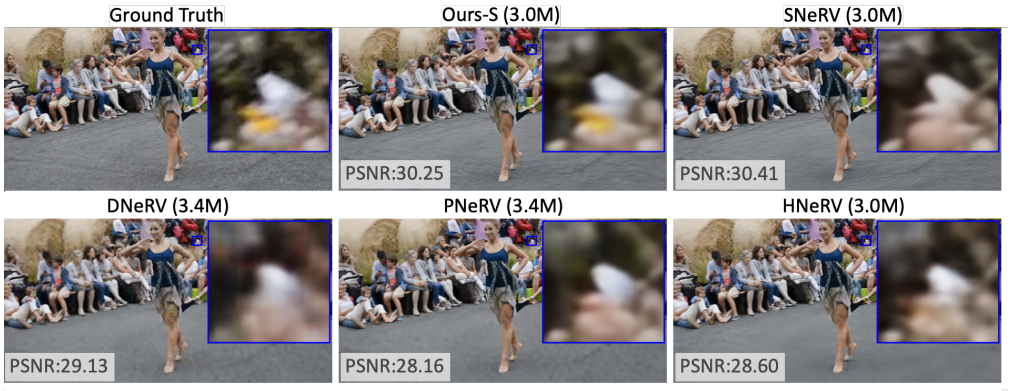


Figure A12: Reconstruction results on 'dance-twirl' from the DAVIS dataset