Fast Encoding and Decoding for Implicit Video Representation

Hao Chen[1] Saining Xie[2] Ser-Nam Lim[3] Abhinav Shrivastava[4]

[1]Meta AI [2]New York University [3]University of Central Florida
[4]University of Maryland, College Park

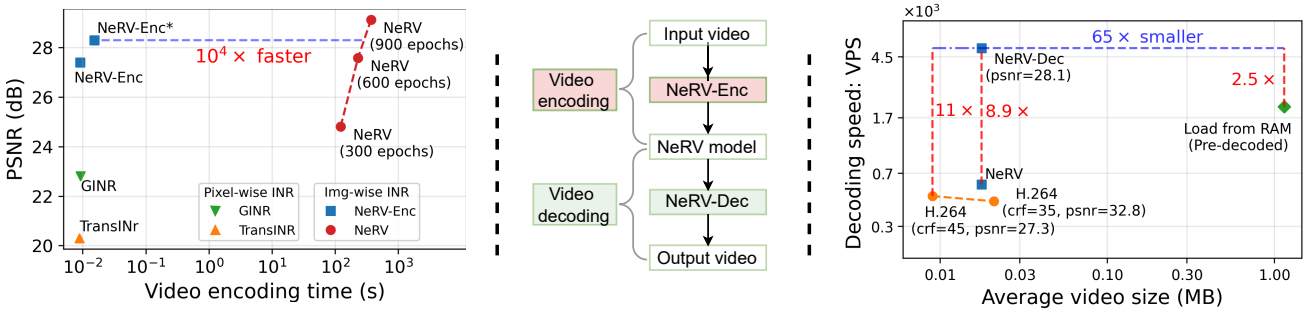

Figure 1. **Left:** video encoding for implicit video representations. NeRV-Enc is $10^4\times$ faster than NeRV [5] baseline (with gradient-based optimization). * uses a larger encoder and more training videos. **Right:** video decoding. NeRV-Dec decodes videos $8.9\times$ faster than NeRV and $11\times$ faster than H.264. It is even $2.5\times$ faster than loading pre-decoded videos from RAM while being $65\times$ smaller in video size.

Abstract

Despite the abundant availability and content richness for video data, its high-dimensionality poses challenges for video research. Recent advancements have explored the implicit representation for videos using neural networks, demonstrating strong performance in applications such as video compression and enhancement. However, the prolonged encoding time remains a persistent challenge for video Implicit Neural Representations (INRs). In this paper, we focus on improving the speed of video encoding and decoding within implicit representations. We introduce two key components: NeRV-Enc, a transformer-based hyper-network for fast encoding; and NeRV-Dec, a parallel decoder for efficient video loading. NeRV-Enc achieves an impressive speed-up of $\mathbf{10^4}\times$ *by eliminating gradient-based optimization. Meanwhile, NeRV-Dec simplifies video decoding, outperforming conventional codecs with a loading speed* $\mathbf{11}\times$ *faster, and surpassing RAM loading with pre-decoded videos (*$\mathbf{2.5}\times$ *faster while being* $\mathbf{65}\times$ *smaller in size).*

1. Introduction

Video research is a fundamental area in computer vision, owing to its rich visual content and widespread presence. However, the immense size of video data presents challenges, as video storage, loading, and processing demands are orders of magnitude larger than those for images. Recent research has explored the potential of representing high-dimensional video data as deep neural networks [5–7, 21, 27, 29]. These representations are favored in applications like video compression (achieving up to $1000\times$ size reductions while maintaining good visual quality), and video enhancement [5, 7, 13, 27]. Unlike pixel-wise Implicit Neural Representation (INR) methods that use MLP networks to model individual pixels, the NeRV series [5–7, 21] employs convolution neural networks to generate entire frames, enhancing video encoding and decoding efficiency.

Despite the efficiency gains achieved by the NeRV series, training a neural network to overfit a given video using gradient-based optimization remains time-consuming. Since video encoding involves mapping input video to NeRV model weights, we propose a straightforward alternative: NeRV-Enc. NeRV-Enc employs a hyper-network to directly generate model weights, thus avoiding the cumbersome encoding process. Our work addresses two fundamental questions: Can a hyper-network be effectively trained on a given video dataset? And can a well-trained hyper-network generalize well to unseen videos?

Given the outstanding performance of transformer networks in various visual tasks [4, 15, 22, 32, 49], we utilize transformers as the hyper-network. This hyper-network

takes video patches and initial weight tokens as input, depicted in Fig. 2. Our research demonstrates that NeRV-Enc provides positive answers to the questions raised earlier. We show that it is indeed possible to train NeRV-Enc on training videos, and it effectively generalizes to unseen videos. In comparison to training NeRV with gradient-based optimization, NeRV-Enc significantly reduces the encoding time, achieving a speed-up of $\mathbf{10^4}\times$ (Fig. 1 Left).

In addition to video encoding, efficient decoding plays a vital role. While a video is encoded just once, it can be decoded countless times. As such, the efficiency of decoding is critical for video playback, streaming, and preview. In the context of video research, especially in large-scale experiments, video loading involves a more complex decoding pipeline compared to image data loaders. Data loaders that require significant computational resources can prolong research cycles and present substantial hurdles when training video models with extensive datasets [52]. Indeed, data loading significantly hampers training efficiency, causing a slowdown of 46% in video self-supervised learning, as highlighted in [17]. In this context, video loading stands out as a critical bottleneck, hindering the progress of video research. To address this issue, we present NeRV-Dec, a parallel and efficient video loader based on NeRV.

Traditional video codecs, like H.264, face challenges with complex decoding pipelines, demanding customized design and optimization for specific applications. In contrast, our NeRV-Dec makes vidoe decoding simpler and faster by efficient parallelization. As shown in Fig. 1 Right, NeRV-Dec is $8.9\times$ faster than the NeRV baseline and achieves an $11\times$ speed improvement over H.264. Notably, NeRV-Dec outperforms RAM loading of pre-decoded videos by $2.5\times$ faster, while utilizing much less disk storage ($65\times$ smaller). Unlike existing video codecs tailored primarily for CPU use, NeRV-Dec emerges as the superior choice, especially in deep learning research. NeRV-Dec efficiently utilizes the power of advanced hardware like GPUs, TPUs, and NPUs, which are already the favored platforms for many users. It achieves this without the need for any specialized design or optimization for video loading, making it highly compatible and easy to integrate into existing workflows.

In summary, this paper introduces NeRV-Enc, a hyper-network designed to accelerate encoding by generating NeRV model weights without gradient-based optimization. Our results show the substantial potential of NeRV-Enc, achieving an impressive speed-up of $10^4\times$ compared to training NeRV with gradient-based optimization. Additionally, we present NeRV-Dec, an parallel video decoder that is $11\times$ faster than tradition codecs (H.264) and $2.5\times$ faster than loading pre-decoded videos from RAM.

2. Related Work

Implicit Neural Representations. Recent advances in deep learning have given rise to implicit neural representations, which are compact data representations [5, 6, 16, 34]. These representations fit neural networks to signals like images, 3D shapes, and videos. A prominent subset of implicit representations is coordinate-based neural representations, which take pixel coordinates as input and produce corresponding values, such as density or RGB values, using MLP networks. These representations have demonstrated promise in diverse applications, including image reconstruction [44, 48], image compression [16], continuous spatial super-resolution [2, 11, 23, 45], shape regression [12, 37], and novel view synthesis for 3D scenes [36, 41].

In contrast to coordinate-based methods, NeRV [5] introduces an image-wise implicit representation that takes the frame index as input and outputs the entire frame without iterative pixel-wise computations. NeRV leverages convolutional neural networks, offering improved efficiency and regression quality compared to coordinate-based methods. By representing videos as neural networks, NeRV transforms video compression into model compression, achieving comparable performance with common video codecs through model pruning, quantization, and entropy encoding. Building on NeRV's success, works [6, 7, 21, 29] further enhance efficiency and shows superior performance in video compression, interpolation, and enhancement. Our approach leverages image-wise representations since NeRV series provide higher capacity and faster decoding speed compared to coordinate-based methods.

Hyper-Networks. Hyper-networks [19] are commonly employed to generate model weights for another neural network based on input data or a dataset. The concept of content-adaptive weights is prevalent in deep learning, exemplified in techniques like dynamic convolution [10] and conditional convolution [53]. Various methods have been developed to modulate model weights in a latent space [35, 37, 42, 43] rather than generating all weights directly, a strategy that can alleviate learning challenges. In these approaches, the neural network takes both pixel coordinates and a content-adaptive vector for modulation, where the modulated vectors serve as the hyper-networks. TransINR [8] and GINR [26] employ hyper-networks to generate model weights for pixel-wise INRs, suitable for image and video regression.

Efficient Video Codecs. Existing video dataloaders rely on traditional codecs like MPEG [28], H.264 [50], and HEVC [47] to reduce video size. Advanced video codecs like AV1 [9] and VVC [3] offer improved compression but at the expense of longer decoding times, hampering data loading speed. Recent developments in deep learning have

Variable	Definition
x	Input video
x_t	Video frame at time t
$\hat{x}_t$	Reconstructed video frame at time t
g_ϕ	Hyper-network (w/ parameter ϕ)
f_θ	NeRV model (w/ parameter θ)
θ_0	Initial weight tokens (hyper-network input)
$\hat{\theta}'$	Hyper-network output: video-specific weights
θ_1	Video-agnostic model weights
θ'	Final NeRV model weights
$D_{\text{train}}, D_{\text{test}}$	Training and test set
$C_{\text{out}}, C_{\text{in}}$	Convolution output and input channel width
K, S	Kernel size, upscale factor for NeRV blocks
M	Number of video patches
N	Number of weight tokens
d	Token dimension for transformer encoder
d_{out}	Output dimension for weight tokens

Table 1. Variables and their definitions.

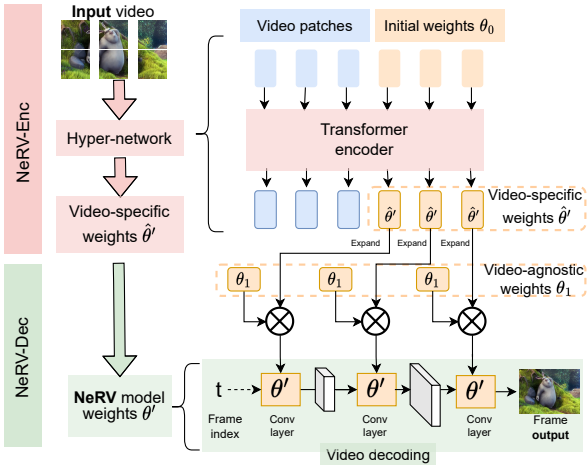

Figure 2. **Top:** video encoding. NeRV-Enc processes the input video x to get video-specific weights $\hat{\theta}'$ using the hyper-network. **Bottom:** video decoding. NeRV-Dec generates final NeRV weights θ' and reconstruct video $\hat{x}$.

introduced techniques for video compression, seeking to enhance traditional methods, including image compression, interpolation [14, 51], autoencoders [20], modeling conditional entropy between frames [31], and rethinking video compression with deep learning [1, 30, 39], or refine existing codecs [25, 40]. Although these learning-based methods improve bits-distortion performance, they introduce significant decoding latency, rendering them unsuitable for fast video loading.

In contrast, loading videos with implicit representations is straightforward, simple, and fast. NeRV [5], a recent implicit video representation, achieves comparable compression performance with traditional video codecs, by reshaping video compression into model compression. The video decoding process in NeRV is a simple feed-forward operation of the convolution network and can be seamlessly deployed across various devices without specific design or optimization. Drawing inspiration from NeRV, our NeRV-Dec significantly reduces video model size via quantization and entropy encoding. Meanwhile, it improves decoding speed further by enabling scalability and improved parallelization.

3. Method

To improve the encoding speed of implicit video representation, we present NeRV-Enc. It employs a hyper-network denoted as g_ϕ to generate weights θ' for the NeRV model f_θ, based on input video data $x \in \mathbb{R}^{t\times 3\times h\times w}$. These learned weights are then used to reconstruct video frames, yielding $\hat{x}_t = f_{\theta=\theta'}(t)$. The primary objective is to minimize the reconstruction error between $\hat{x}_t$ and the its ground truth x_t for training videos D_{train}, while ensuring the generalization to testing videos D_{test}. Additionally, we introduce NeRV-Dec, a method designed for parallel video decoding with good efficiency. Please consult Tab. 1 for a comprehensive list of symbol definitions.

3.1. Video Encoding: NeRV-Enc

Generate Video-Specific Weights. We employ a Transformer network with L encoder layers as a hyper-network to generate video-specific model weights, denoted as $\hat{\theta}'$. The input video x is partitioned into patches, which are then transformed into patch tokens using a fully connected (FC) layer. Additionally, learned position embeddings are added to these patch tokens. These patch tokens, in conjunction with the initial weight tokens θ_0, form the input tokens. Subsequently, the hyper-network processes these input tokens to produce video-specific weights $\hat{\theta}' \in \mathbb{R}^{d_{\text{out}}\times N}$, which serves as an compact representation of the video. This encoding process is depicted in Fig. 2 Top and Fig. 4 Left.

Adaptive Weight Tokens across Layers. We've observed that token importance varies across different layers. Unlike TransINR [8], which employs an identical number of weight tokens for all layers, or GINR [26], which restricts weight tokens to a specific layer (2nd layer), we introduce a flexible and adaptive approach to weight token distributions. This customization results in three distinct schemes: *uniform weight tokens* (TransINR), *layer-specific weight tokens* (GINR), and *adaptive weight tokens* (our approach), each designed to suit video-specific weight distribution across layers. These diverse weight distributions are visually depicted in Fig. 3.

NeRV-Enc for Video Restoration. Implicit video representations have proven to be good at video restoration tasks such as denoising and inpainting. Besides reconstruc-

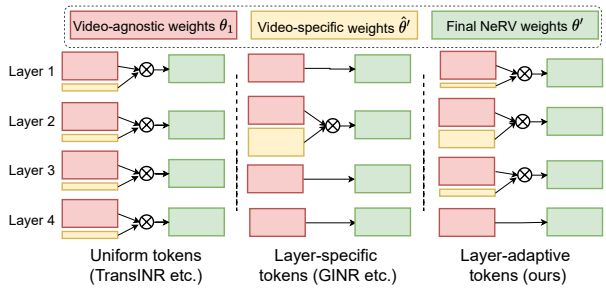

Figure 3. Weight token distributions across layers. **Left:** Uniform (TransINR [8]). **Middle:** Layer-specific (GINR [27]). **Right:** Layer-adaptive (ours).

tion, we also extend NeRV-Enc to various video restoration tasks. NeRV-Enc takes degraded videos as input and use ground truth videos as supervision. Evaluations on unseen videos indicate that NeRV-Enc effectively addresses these degradation tasks. Note that while previous implicit methods like NeRV and HNeRV demonstrate competence in restoration tasks, they often require additional supervision, such as masks for inpainting. Furthermore, their training might require reference data for the test video, like high-resolution or de-blurred frames. In contrast, NeRV-Enc leverages large-scale training to learn restoration capabilities and shows robust generalization to unseen videos. This makes NeRV-Enc a more practical and versatile tool for video restoration.

3.2. Video Decoding: NeRV-Dec

Efficient Video Storage. Prior to video decoding, it is necessary to store the video-specific weights efficiently. To achieve storage efficiency, we employ weight quantization and entropy encoding for these weights, similar to compression techniques used in NeRV approaches [5, 7].

Generate NeRV Weights. After obtaining the video-specific weights $\hat{\theta}'$, we move on to calculate the final weights θ' for a NeRV model, which we represent as $f_{\theta=\theta'}$. As shown in Fig. 4 (b), the video-specific weights $\hat{\theta}' \in R^{d_{\text{out}} \times N}$ for a convolution layer (with parameters in $R^{C_{\text{out}} \times C_{\text{in}} \times K \times K}$) might be smaller. To compensate, we introduce *learnable* parameters $\theta_1 \in R^{C_{\text{out}} \times C_{\text{in}} \times K \times K}$ for each layer, which are shared across all videos and known as *video-agnostic* parameters. The final weights of the NeRV model, detailed in Fig. 4, are determined by an element-wise multiplication of the video-agnostic weights θ_1 and the video-specific weights $\hat{\theta}'$.

Video Decoding Prelimaries for NeRV. With the final NeRV model weights, we access the video frame using a feedforward operation with the frame index t. Following a common practice in implicit neural representations [5, 36], we initially normalize t to the interval [-1, 1], apply positional encoding, and obtain a time embedding vector. This

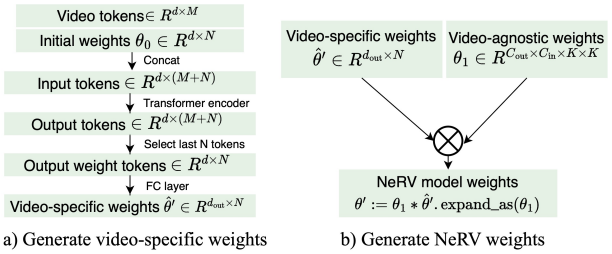

Figure 4. **Left** Generate video-specific weights $\hat{\theta}'$ via the hyper-network. **Right** Generate NeRV weights θ' by element-wise multiplication of $\hat{\theta}'$ and video-agnostic weights θ_1.

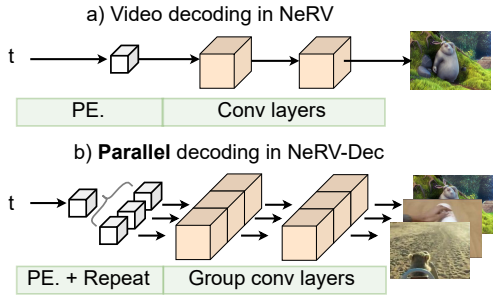

Figure 5. **Parallel video decoding** of NeRV-Dec is achieved using group convolution. NeRV decoding is a special case of NeRV-Dec when the group size is 1. 'PE.' denotes position encoding. 'Repeat' indicates embedding repetition for input expansion.

time embedding is used as input to the NeRV model f_θ, which generates the video frame $\hat{x}_t$. The output of the final layer is adjusted by an output bias of 0.5, considering image normalization within the [0, 1] range. In essence, video decoding can be expressed as $\hat{x}_t = f_{\theta=\theta'}(t)$, as shown in Fig. 5 (a). Efficient video decoding can be accomplished by running NeRV with a batch size encompassing all frames.

Parallel Decoding for NeRV-Dec. NeRV-Dec enables parallel decoding by running several NeRV models at once, allowing multiple videos to be decoded simultaneously. This process is efficiently executed using group convolution, where each group corresponds to a separate NeRV model, as illustrated in Fig. 5 (b). Note that NeRV decoding is a special case of NeRV-Dec when video number is 1. Since video decoding is an essential operation in implicit neural representations, this method of parallelization significantly boosts the training speed of NeRV-Enc. Furthermore, it improves the scalability and efficiency of NeRV-Dec when used as a video data loader.

3.3. Model Optimization

To optimize NeRV-Enc, our objective is to minimize the reconstruction loss between the ground truth frame x_t and the reconstructed frame $\hat{x}_t$, which is expressed as:

$$\phi^* = \arg\min_{\phi,\theta_0,\theta_1} \sum_{x \in D_{\text{train}}} \sum_t \| f_{\theta=g_\phi(x)}(\mathbf{t}) - \mathbf{x}_t \|_2^2 \quad (1)$$

. Once we have the optimized NeRV-Enc ϕ^*, we evaluate it on the test set D_{test}, addressing the two key questions raised in Sec. 1: can NeRV-Enc effectively fit training videos, and can it successfully generalize to test videos.

NeRV-Enc consists of three sets of learnable parameters: those for the hyper-network ϕ, initial weight tokens θ_0, and model-agnostic weights θ_1. For optimization, we utilize a training objective based on the mean square error (MSELoss) between the output frame and the ground truth frames.

4. Experiment

4.1. Datasets and Implementation Details

In our experiments, we utilize three widely-adopted video datasets: Kinetics-400 (K400, our training dataset)[24], Something-Something V2 (SthV2)[18], and UCF101 [46]. Kinetics-400 comprises about 240k training videos and 20k test videos, each lasting 10 seconds, across 400 classes. Due to the extensive size of the full dataset, we may use a subset of K400 for training, consisting of 25 videos per class. For evaluation, we employ the test sets of K400, SthV2 (about 20k motion-centric videos), and UCF101 (about 3.5k human-centric videos). The quality of the reconstructed videos is evaluated using PSNR and SSIM.

The default video size is 256×256 with 8 frames. To preprocess the data, we first resize the input video so that its shorter side is 256. We then perform a center crop to obtain a 256×256 clip. Subsequently, we uniformly sample 8 frames from the clip and input them to the model. For data tokenization, we divide the videos into 64×64 patches. The video model consists of four NeRV blocks, each with upscale factors of 4, 4, 4, and 4, respectively [5]. The convolution layers in these blocks maintain a consistent channel width of 16, except for the first block, where the input channel represents the time embedding dimension, and the last block, where the output channel corresponds to the video channels. The kernel size is 1 for the first convolution layer and 3 for the subsequent ones.

The default transformer hyper-network is composed of 6 encoder layers with a hidden dimension of 720 and a forward dimension of 2800. We employ the AdamW [33] optimizer with a batch size of 32, an initial learning rate of 1e-4. Our learning rate undergoes a step-wise decay, decreasing by a factor of 0.1 at 90% of the total training steps. Our implementation is built on PyTorch [38]. Model training is conducted using 8 A100 GPUs for all experiments, unless otherwise specified. For video decoding, we test on a machine with 1 A100 GPU and 8 CPUs [1]. Additional implementation details and visualization results are available in the supplementary material.

[1] Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz

4.2. Video Encoding

NeRV-Enc *vs*. Pixel-wise INRs. We begin by comparing our method to pixel-wise methods TransINR [8] and GINR [26], which also employ hyper-networks for generating implicit representations. However, these methods rely on pixel-wise Implicit Neural Representations (INRs), which exhibit inherent inefficiencies, particularly in large-scale training, as demonstrated in Tab. 2.

Firstly, pixel-wise INRs are notably slower than image-wise INRs, a fact also demonstrated in NeRV [5]. For instance, with the same 150-epoch training, NeRV-Enc outperforms pixel-wise methods, achieving $\mathbf{7}\times$ faster training for 4 frames, $\mathbf{10.8}\times$ for 8 frames, and $\mathbf{15.6}\times$ for 16 frames. *Secondly* despite having a smaller INR model θ and less model-specific weights $\hat{\theta}'$ (*i.e*. smaller video size), NeRV-Enc achieves superior video reconstruction quality for both training and testing videos, as measured by PSNR and SSIM metrics. The difference in quality becomes more noticeable when comparing pixel-wise methods to NeRV-Enc with *similar training times*. In this comparison, NeRV-Enc outperforms them by **+4.6, +5.0,** and **+4.1** PSNR for videos with 8 frames in the K400, SthV2, and UCF101 test sets, respectively. Qualitative results are provided in Fig. 6, comparing NeRV-Enc with pixel-wise methods. These visual comparisons illustrate that NeRV-Enc excels in capturing videos with superior fidelity and fine details.

Scale NeRV-Enc. We also investigate scaling techniques to further improve the performance of NeRV-Enc: longer training epochs, more training videos, and a larger hyper-network. The results are presented in Tab. 3. It is evident that expanding the training epochs and incorporating more training videos leads to substantial improvements in reconstruction quality, with a notable increase of $+2$ in PSNR and $+0.06$ in SSIM for both training and testing videos. Additionally, increasing the size of the hyper-network, together with a larger dropout ratio for transformer layers as well, enhances reconstruction performance. The integration of these three techniques results in our final NeRV-Enc model, leading to **+2.6, +3.1**, and **+2.9** PSNR improvements for three test sets. We find that NeRV-Enc's reconstruction performance improves with increased computational resources and provide more ablation results in the appendix.

NeRV-Enc *vs*. NeRV. Using the finalized NeRV-Enc, we compare it with the NeRV baseline [5] which uses gradient-based optimization for model fitting and video encoding. The results, depicted in Fig. 7, highlight the effective generalization of NeRV-Enc across various video datasets, including K400, SthV2 and UCF101. Significantly, NeRV-Enc demonstrates a remarkable encoding acceleration, being $\mathbf{10^4}\times$ times faster than the NeRV baseline, yet maintaining comparable output quality, as measured by PSNR and SSIM metrics. It's noteworthy that NeRV-Enc enables

Methods	F	Encoder size	INR size ↓	$\#\hat{\theta}' \downarrow$	Epoch	GPU hrs ↓	PSNR ↑ Train	K400	SthV2	UCF101	SSIM ↑ Train	K400	SthV2	UCF101
TransINR [8]	4	48.0M	99k	25k	150	63	23.7	22.1	24.6	22.1	0.659	0.631	0.728	0.622
GINR [27]	4	47.6M	139.4k	25.6k	150	65	24.5	23.2	25.9	23.1	0.685	0.66	0.744	0.66
NeRV-Enc	4	47.6M	85.6k	24.1k	150	9	**26.6**	**26.6**	**29.4**	**26**	**0.756**	**0.754**	**0.816**	**0.752**
NeRV-Enc	4	47.6M	85.6k	24.1k	1000	62	27.9	27.5	30.5	27.1	0.794	0.783	0.838	0.783
TransINR [8]	8	48.0M	99k	25k	150	119	22.3	20.3	22.8	20.7	0.626	0.595	0.703	0.591
GINR [27]	8	47.6M	139.4k	25.6k	150	123	23.9	22.8	25.3	22.7	0.671	0.65	0.737	0.651
NeRV-Enc	8	47.6M	85.6k	24.1k	150	11	**25.8**	**25.8**	**28.5**	**25.2**	**0.732**	**0.727**	**0.795**	**0.723**
NeRV-Enc	8	47.6M	85.6k	24.1k	1500	110	27.8	27.4	30.3	26.8	0.791	0.78	0.835	0.778
TransINR [8]	16	48.0M	99k	25k	150	234	21.5	18.4	21.1	19.2	0.615	0.555	0.678	0.561
GINR [27]	16	47.6M	139.4k	25.6k	150	242	22.9	21.7	24.2	21.7	0.647	0.624	0.72	0.625
NeRV-Enc	16	47.6M	85.6k	24.1k	150	15	**23.6**	**23.2**	**25.9**	**22.9**	**0.657**	**0.642**	**0.731**	**0.642**
NeRV-Enc	16	47.6M	85.6k	24.1k	2000	200	25.4	24.9	27.7	24.5	0.711	0.693	0.772	0.692

Table 2. **NeRV-Enc *vs*. Pixel-wise INR methods**. NeRV-Enc is much faster (up to 15×) than pixel-wise methods for training. It also shows better quality in reconstructing videos across datasets, as measured by PSNR and SSIM. 'F' refers to frame number, $\#\hat{\theta}'$ is the size of video-specific weights. Training time is measured in 'GPU hrs'.

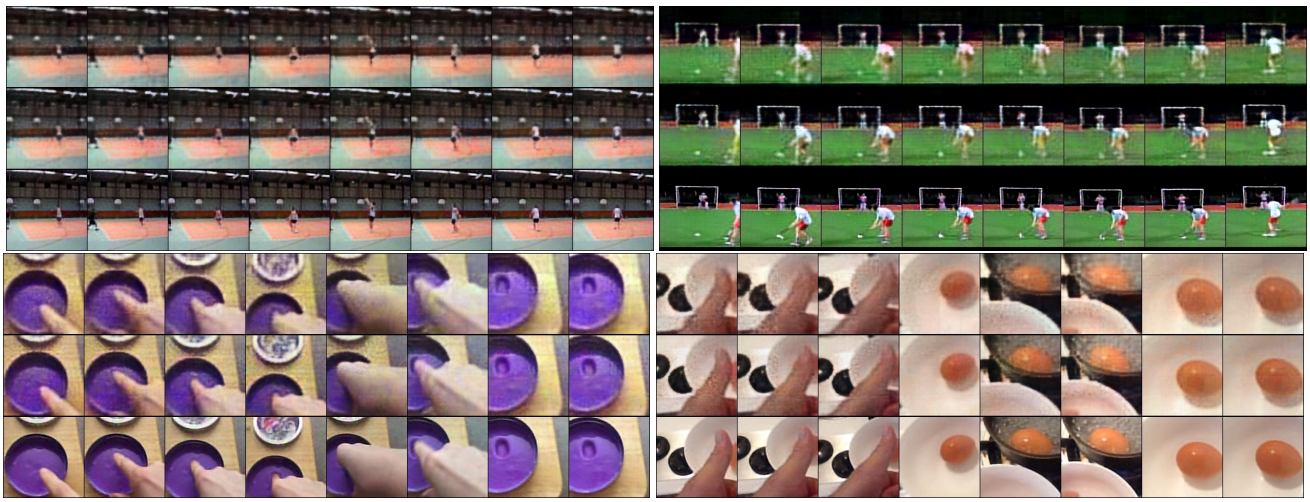

Figure 6. Visualizations for INR encoding methods: TransINR [8] (**Top**), GINR [26] (**Middle**), and NeRV-Enc (**Bottom**, ours). Our method excels in reconstructing videos with superior fidelity and fine details. Best viewed digitally and zoomed in.

$\mathcal{E}$	$\mathcal{S}$	$\mathcal{N}$	PSNR ↑ Train	K400	SthV2	UCF	SSIM ↑ Train	K400	SthV2	UCF
			25.8	25.8	28.5	25.2	0.732	0.727	0.795	0.723
✓			27.8	27.4	30.4	26.9	0.791	0.782	0.837	0.781
	✓		26.5	26.7	29.5	26.2	0.756	0.762	0.822	0.759
		✓	27.8	27.9	30.9	27.4	0.792	0.799	0.852	0.802
✓	✓	✓	28.1	28.4	31.6	28.1	0.803	0.808	0.862	0.817

Table 3. **Scale NeRV-Enc** by increasing training epochs $\mathcal{E}$, hypernerwork size $\mathcal{S}$, training video number $\mathcal{N}$.

real-time video encoding, positioning implicit video representation as a viable and efficient option for video codec.

NeRV-Enc for Video Restoration. Like prior methods that use implicit representations for videos, we found that NeRV-Enc is also effective for video restoration tasks. Our findings, detailed in Fig. 8, cover three types of video degradation: downsampling, blurring, and masking of input videos. We observed that these *degradations in pixel space are effectively restored in implicit space*. While our primary focus in this paper is on fast encoding and decoding, we have included quantitative results in the appendix for further reference.

Ablation for Layer-adaptive Modulation. The ablation study on layer-wise modulation is presented in Tab. 4, demonstrating that even with reduced parameters in Layer 2 (see row 3), we can achieve results on par with the layer-specific approach (see row 2). Moreover, we incrementally increase parameters in other layers (rows 4 to 7) until no additional improvements were observed. In examining the

Methods	Layer 1/2/3/4 params	TotalParam	Train	K400	SthV2	UCF101
Layer-uniform [8]	4.1k_9.2k_9.2k_6.9k	29.4k	22.1	21.5	24.1	21.6
Layer-specific [27]	0_36.9k_0_0	36.9k	25.5	25.4	28.1	24.8
Layer-adaptive (ours)	0_18.4k_0_0	18.4k	25.1	25.0	27.7	24.4
	1.1k_18.4k_4.6k_0	**24.1k**	**25.8**	**25.8**	**28.5**	25.2
	2.4k_18.4k_4.6k_0	25.4k	25.5	25.4	28.2	25.0
	1.1k_18.4k_9.2k_0	28.7k	25.8	25.8	28.5	**25.3**
	1.1k_18.4k_4.6k_0.1k	24.2k	25.7	25.6	28.3	25.1

Table 4. **Ablation Study** on layer-daptive modulation. We compare uniform tokens (TransINR [8]), layer-specific tokens (GINR [27]), and our layer-adaptive tokens. Our approach surpasses the layer-specific method in reconstruction performance (PSNR ↑) and achieves a 50% reduction in video size (total parameters ↓).

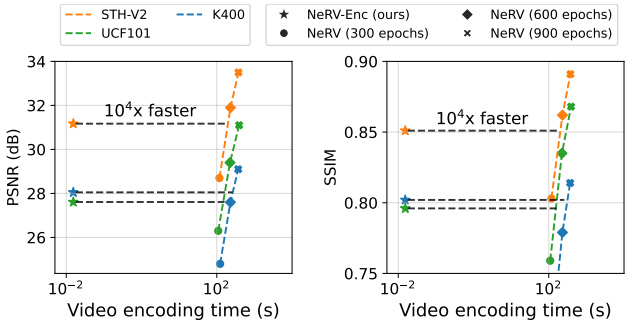

Figure 7. **Encoding speed.** NeRV-Enc (ours) is $10^4\times$ faster than NeRV [5] baseline (using gradient-based optimization) across multiple datasets.

distribution of weight tokens, we evaluate uniform tokens (TransINR [8]), layer-specific tokens (GINR [27]), and our proposed layer-adaptive weight tokens, which use 29.4K, 36.9K, and 24.1K video-specific weights, respectively. The results indicate that our approach not only achieves better reconstruction quality but also does so with fewer video-specific parameters, contributing to reduced video size and enhanced compression efficiency. Note that our layer-adaptive modulation (highlighted in the gray row) not only surpasses the performance of the layer-specific method but also achieves a *50% reduction* in the total number of parameters (video size).

4.3. Video Decoding

Parallelization. NeRV-Dec improves upon NeRV by using multiple stacked models to decode several videos at once. Its efficiency comes from a shared time embedding layer and a group convolution process that can be parallelized. We evaluate this parallelization by varying the number of videos for decoding. NeRV decoding is a specific case of NeRV-Dec when decoding video number is 1. As shown in Fig. 9 (Left), there is a near-linear speedup in decoding as video batch size increases from 1 to 16. Beyond a batch size of 16, decoding speed plateaus around 5200 videos per second, maximizing GPU usage. Video decoding (data loading) is crucial as videos are decoded repeatedly, essential for playback, streaming, and research. Preview of multiple videos is also common for video platforms. Typically, videos are divided into clips or picture groups for storage. Our parallel decoding significantly boosts video loading efficiency.

Comparison with H.264. NeRV-Dec outperforms a traditional data loader using the H.264 codec (485 videos per second for 8 CPUs), achieving a $11\times$ increase in loading speed. To further understand the decoding performance of H.264, we explore various ways for its speed improvement. Surprisingly, transitioning from CPU to GPU as the decoding device does not yield speed gains, except for larger videos. Consequently, we increase the number of CPUs and data loader workers, presenting the results in Fig. 9 (Right). The results show that while augmenting the CPU count enhances video loading speed, the improvements diminish, especially beyond 16 CPUs. It's worth noting that even with **96** CPUs, H.264 is still $\mathbf{2}\times$ slower than our NeRV-Dec. Given that powerful chips like GPUs are already preferred for deep learning practitioners, the video loading advantages of NeRV-Dec can be leveraged without the need for additional hardware or specialized design and optimization.

Detailed Comparisons. Besides the video decoding speed, we conduct a comprehensive assessment of NeRV-Dec, examining video reconstruction quality and compression performance. Our method undergoes comparisons against various video representations, including traditional codecs like H.264 and AV1, as well as RAM reading (load pre-decoded videos from RAM) for efficient data loading. Detailed results are presented in Tab. 5.

Video Compression. To compress the video size in NeRV-Dec, we implement quantization to the video-specific weights ($\hat{\theta}'$) and employ Huffman encoding. These techniques substantially decreases disk storage requirements while maintaining video quality. It retains 99% of the PSNR and SSIM values of the original model using just *6 bits* per parameter. With a reduced video size

Figure 8. **Top:** input videos with various degradations from test set. **Bottom:** output videos of NeRV-Enc. *Left*: downsampled; *Middle*: blurred; *Right*: mask.

	RAM	AV1	H.264			NeRV-Dec (ours)				
		CRF 60	CRF 35	CRF 40	CRF 45	8 bits	7 bits	6 bits	5 bits	4 bits
Size ↓	1.15MB	21.9KB	20.4KB	13.1KB	**8.7KB**	23.7KB	20.7KB	17.7KB	14.7KB	11.6KB
PSNR ↑	-	32.4	**32.8**	30.0	27.3	28.4	28.3	28.1	27.5	25.6
SSIM ↑	-	0.910	**0.912**	0.860	0.788	0.808	0.807	0.802	0.784	0.712
VPS ↑	2031	313	447	460	485			**5175**		

Table 5. **Detailed comparison.** NeRV-Dec reduces video size by 65× via weight quantization, while being 2.5× faster than loading pre-decoded videos from RAM. Although AV1 and H.264 provide better compression (smaller video size) and video quality (higher PSNR and SSIM), NeRV-Dec decodes videos much faster (higher VPS).

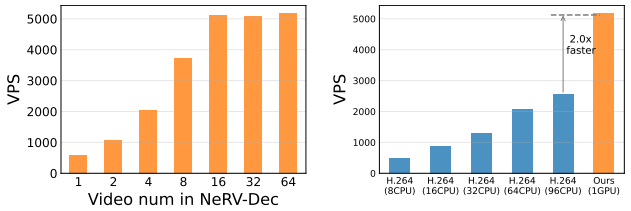

Figure 9. **Left:** NeRV-Dec's decoding speed scales efficiently with increasing video numbers. Note NeRV decoding is a special case of NeRV-Dec when video number is 1 (the leftmost bar). **Right:** NeRV-Dec is **2**× faster than H.264 with **96** CPUs.

(65× smaller), NeRV-Dec even surpasses loading videos from RAM, achieving a 2.5× faster speed with GPU assistance. As deep learning platforms commonly utilize powerful hardware like GPUs for computation, the fast and simple video loading provided by NeRV-Dec can significantly enhance video research and alleviate data loading bottlenecks.

Comparison with Traditional Codecs. While the compression ratio of our method is slightly lower than that of conventional codecs such as H.264 and AV1, our method is 11× faster in video decoding and can be easily implemented on most devices without any specific design or optimization. As outlined in Tab. 3, enhancing the scale of NeRV-Enc's training by employing additional resources (like more videos for training, extended training epoches, or a larger hyper-network) could yield further improvements and lead to better compression performance. Beyond just numerical comparisons, we provide visual comparisons between H.264 and NeRV-Dec at an equivalent PSNR of 28.1, as shown in Fig. 10. These visual examples clearly demonstrate the blocking artifacts present in H.264, emphasizing the superior visual quality offered by NeRV-Dec. This This advantage highlights the potential preference for NeRV-Dec in scenarios where visual quality is a priority.

4.4. Limitations and Future Works

Hybrid INRs enhance the NeRV framework by utilizing inputs that are either content-adaptive embeddings or learnable 2D/3D grids. These enhancements allow for superior performance in tasks like video compression or interpolation compared to the original NeRV. We extend NeRV-Enc to also produce these video-specific embeddings or learnable grid features, which then serve as inputs for NeRV. We show results in Tab. 6 and NeRV-Enc is adaptable to diverse INR methods. While NeRV-Enc shows promise in facilitating hybrid INRs, further investigation is required to achieve better results. The hybrid INR approach (HNeRV), which involves learning both input embedding and decoder weights, necessitates a more complex design compared to NeRV where only decoder weights are learned.

Our current research is confined to video encoding with predetermined frame numbers and spatial resolutions. A potential direction for future work is to adapt our methodology to efficiently manage videos with varying lengths and resolutions, without the need to simply segment them into shorter clips. NeRV-Enc does not yet match the performance of established video codecs such as H.264 in video

Figure 10. **Visual comparison:** H.264 (**Top**) *vs*. NeRV-Dec (**Bottom**) at similar PSNR. H.264 exhibits noticeable blocking artifacts, whereas NeRV-Dec provides a more visually appealing result. Best viewed digitally and zoomed in.

	PSNR ↑				SSIM ↑			
INR	Train	K400	SthV2	UCF101	Train	K400	SthV2	UCF101
NeRV	25.8	25.8	28.5	25.2	0.732	0.727	0.795	0.723
HNeRV	23.1	22.9	25.1	22.6	0.647	0.644	0.727	0.642

Table 6. NeRV-Enc results for NeRV and hybrid INR (HNeRV).

compression. Closing this performance gap is another focus for our future work. Meanwhile, further exploration of reconstruction loss and restoration tasks holds promise to produce visually-appealing results.

5. Conclusion

In this paper, we introduce NeRV-Enc, a hyper-network that improve encoding speed by generating weights for the NeRV model. Our findings reveal that NeRV-Enc significantly accelerates the encoding process, achieving a remarkable speed-up of 10^4 times compared to the traditional training of NeRV using gradient-based optimization. Additionally, we present NeRV-Dec, a parallel video decoder that surpasses traditional codecs in speed by $11\times$, and outperforms the speed of loading pre-decoded videos from RAM by $2.5\times$. This advancement offers a significant improvement in video decoding and loading efficiency.

References

[1] Eirikur Agustsson, David Minnen, Nick Johnston, Johannes Balle, Sung Jin Hwang, and George Toderici. Scale-space flow for end-to-end optimized video compression. In *CVPR*, 2020. 3

[2] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. In *CVPR*, pages 14278–14287, 2021. 2

[3] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm. Overview of the versatile video coding (vvc) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021. 2

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*. Springer, 2020. 1

[5] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. NeRV: Neural representations for videos. In *NeurIPS*, 2021. 1, 2, 3, 4, 5, 7

[6] Hao Chen, A Gwilliam Matthew, Bo He, Ser-Nam Lim, and Abhinav Shrivastava. CNeRV: Content-adaptive neural representation for visual data. In *BMVC*, 2022. 2

[7] Hao Chen, Matthew Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. HNeRV: Neural representations for videos. In *CVPR*, 2023. 1, 2, 4

[8] Yinbo Chen and Xiaolong Wang. Transformers as meta-learners for implicit neural representations. In *European Conference on Computer Vision*, 2022. 2, 3, 4, 5, 6, 7, 11

[9] Yue Chen, Debargha Murherjee, Jingning Han, Adrian Grange, Yaowu Xu, Zoe Liu, Sarah Parker, Cheng Chen, Hui Su, Urvang Joshi, et al. An overview of core coding tools in the av1 video codec. In *2018 Picture Coding Symposium (PCS)*, pages 41–45. IEEE, 2018. 2

[10] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *CVPR*, pages 11030–11039, 2020. 2

[11] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *CVPR*, pages 8628–8638, 2021. 2

[12] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019. 2

[13] 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. *CVPR*, 2022. 1

[14] Abdelaziz Djelouah, Joaquim Campos, Simone Schaub-Meyer, and Christopher Schroers. Neural inter-frame compression for video coding. In *ICCV*, 2019. 3

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 1

[16] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. In *ICLR workshop*, 2021. 2

[17] Christoph Feichtenhofer, Yanghao Li, Kaiming He, et al. Masked autoencoders as spatiotemporal learners. *Advances in neural information processing systems*, 35:35946–35958, 2022. 2

[18] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fründ, Peter Yianilos, Moritz Mueller-Freitag, Florian Hoppe, Christian Thurau, Ingo Bax, and

Roland Memisevic. The "something something" video database for learning and evaluating visual common sense. In *ICCV*, 2017. 5

[19] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *ICLR*, 2017. 2

[20] Amirhossein Habibian, Ties van Rozendaal, Jakub M. Tomczak, and Taco S. Cohen. Video compression with rate-distortion autoencoders. In *ICCV*, 2019. 3

[21] Bo He, Xitong Yang, Hanyu Wang, Zuxuan Wu, Hao Chen, Shuaiyi Huang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. Towards scalable neural representation for diverse videos. In *CVPR*, 2023. 1, 2

[22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *CVPR*, 2022. 1

[23] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *arXiv preprint arXiv:2106.12423*, 2021. 2

[24] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 5

[25] Mehrdad Khani, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Efficient video compression via content-adaptive super-resolution. *arXiv preprint arXiv:2104.02322*, 2021. 3

[26] Chiheon Kim, Doyup Lee, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Generalizable implicit neural representations via instance pattern composers. *CVPR*, 2023. 2, 3, 5, 6

[27] Subin Kim, Sihyun Yu, Jaeho Lee, and Jinwoo Shin. Scalable neural video representations with learnable positional features. *NeurIPS*, 2022. 1, 4, 6, 7, 11

[28] Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Commun. ACM*, 1991. 2

[29] Zizhang Li, Mengmeng Wang, Huaijin Pi, Kechun Xu, Jianbiao Mei, and Yong Liu. E-nerv: Expedite neural video representation with disentangled spatial-temporal context. *ECCV*, 2022. 1, 2

[30] Haojie Liu, Tong Chen, Ming Lu, Qiu Shen, and Zhan Ma. Neural video compression using spatio-temporal priors. *arXiv preprint arXiv:1902.07383*, 2019. 3

[31] Jerry Liu, Shenlong Wang, Wei-Chiu Ma, Meet Shah, Rui Hu, Pranaab Dhawan, and Raquel Urtasun. Conditional entropy coding for efficient video compression. In *ECCV*, 2020. 3

[32] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *ICCV*, 2021. 1

[33] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 5

[34] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. In *ICCV*, pages 14214–14223, 2021. 2

[35] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019. 2

[36] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, pages 405–421. Springer, 2020. 2, 4

[37] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 2

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 5

[39] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G. Anderson, and Lubomir Bourdev. Learned video compression. In *ICCV*, 2019. 3

[40] Oren Rippel, Alexander G. Anderson, Kedar Tatwawadi, Sanjay Nair, Craig Lytle, and Lubomir Bourdev. Elf-vc: Efficient learned flexible-rate video coding. In *ICCV*, 2021. 3

[41] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *NeurIPS*, 2020. 2

[42] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *NeurIPS*, 2019. 2

[43] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020. 2

[44] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020. 2

[45] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. In *CVPR*, pages 10753–10764, 2021. 2

[46] Khurram Soomro, Amir Roshan Za4mir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012. 5

[47] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 2012. 2

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

[49] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training

data-efficient image transformers & distillation through attention. In *ICML*, pages 10347–10357. PMLR, 2021. 1

[50] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 2003. 2

[51] Chao-Yuan Wu, Nayan Singhal, and Philipp Krahenbuhl. Video compression through image interpolation. In *ECCV*, 2018. 3

[52] Chao-Yuan Wu, Ross Girshick, Kaiming He, Christoph Feichtenhofer, and Philipp Krahenbuhl. A multigrid method for efficiently training video models. In *CVPR*, pages 153–162, 2020. 2

[53] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *NeurIPS*, 32, 2019. 2

A. Appendix

A.1. Pseudocode of NeRV-Enc and NeRV-Dec

We firstly provide pseudocode for NeRV-Enc and NeRV-Dec in Algorithm 1.

Algorithm 1 Pseudocode in a PyTorch style.

```
############ 1) Video encoding: NeRV-Enc ############
# Input: video x, initial weights θ_0
# Output: video-specific weights θ̂'

# Video tokenization
x = FC_1(x.tokennize())                              # d \times M

# Concat video patches and initial weights as input
x = x.concat(θ_0)                                    # d \times (M+N)

# Hypernetwork g_φ output video-specific weights θ̂'
θ̂' = g_φ.forward(x)[-N:].                            # d \times N
θ̂' = FC_2(θ̂')                                       # C_out \times N

############ 2) Video decoding: NeRV-Dec ############
θ̂' = θ̂'.expand_as(θ_1)          # broadcast into needed shape
θ' = θ_1 * θ̂'                   # C_out \times C_in \times K \times K

# Initial NeRV model f_θ with generated weights
f_θ.reset_parameter(θ')

# Input frame index t, and output video frame x̂_t
x̂_t = f_θ.forward(t)

############### 3) Model optimization ###############
# Compute loss and backward gradients
loss = MSELoss(x̂_t, x_t)
loss.backward()

# update all learnable parameters
update([φ, θ_0, θ_1])
```

FC: fully connected layer;
MSELoss: mean square error loss.

A.2. Scaling the training of NeRV-Enc

We explore factors such as the number of training videos, training epochs, encoder size, and weight token distributions, as outlined in Tab. 7. Generally, NeRV-Enc's reconstruction performance improves with increased computational resources, although the gains tend to plateau as training duration becomes adequate. Higher dropout ratios are essential for achieving improved generalization in longer training epochs or with larger hyper-networks. These ablations underscore that the reconstruction quality of NeRV-Enc can be significantly enhanced by scaling the training with additional resources, including more training videos, longer training epochs, and larger hyper-networks.

	PSNR ↑				SSIM ↑			
	Train	K400	SthV2	UCF101	Train	K400	SthV2	UCF101
Epoch ablation								
150	25.8	25.8	28.5	25.2	0.732	0.727	0.795	0.723
300	26.8	26.7	29.5	26.2	0.763	0.757	0.819	0.757
600	27.2	27.1	30	26.6	0.774	0.768	0.827	0.766
1200	27.6	27.3	30.2	26.7	0.787	0.776	0.832	0.774
1800	27.8	27.4	30.4	26.9	0.791	0.782	0.837	0.781
Encoder size ablation								
47.6M	25.8	25.8	28.5	25.2	0.732	0.727	0.795	0.723
125M	26.4	26.2	28.9	25.6	0.751	0.743	0.806	0.737
251M	26.5	26.7	29.5	26.2	0.756	0.762	0.822	0.759
404M	26.5	26.5	29.3	25.9	0.753	0.755	0.816	0.751
Video number ablation								
10k	25.8	25.8	28.5	25.2	0.732	0.727	0.795	0.723
20k	26.7	26.6	29.4	26	0.757	0.752	0.814	0.751
40k	27	26.9	29.7	26.4	0.764	0.761	0.821	0.762
80k	27.6	27.7	30.6	27.2	0.791	0.791	0.845	0.794
240k	27.8	27.9	30.9	27.4	0.792	0.799	0.852	0.802

Table 7. **NeRV-Enc ablations**. For ablation of weight tokens, we compare uniform tokens (TransINR [8]), layer-specific tokens (GINR [27] at the 2nd layer), and our proposed layer-adaptive weight tokens. Please refer to Fig. 3 for their distinction.

Figure 11. **Top:** Input videos with various degradations (*Left*: downsampled; *Middle*: blurred; *Right*: mask). **Bottom:** Output videos by NeRV-Enc.

A.3. NeRV-Enc for video restoration tasks.

Our NeRV-Enc framework is versatile across various downstream tasks, and shows robust restoration quality for various degradations. Results in Fig. 11 and Tab. 8 demonstrate that the reconstruction quality for downsampled, blurred, and masked input videos is comparable to that achieved through conventional video regression. *This underscores the framework's effectiveness in restoring common pixel degradations within the implicit space.*

A.4. Weight quantization for efficient storage.

In this quantization procedure, each element of a vector μ is mapped to the nearest integer using the linear transforma-

Input degradation	Input PSNR				Output PSNR			
	Train	k400	sth-v2	ucf101	Train	k400	sth-v2	ucf101
Downsample	20.1	20.3	22.9	19.5	24.5	24.3	26.8	23.9
Gaussian blur	23.1	23.3	26.0	22.3	24.8	24.7	27.3	24.0
Inpainting	19.0	18.6	18.1	18.4	25.5	25.2	27.9	24.7
No	-	-	-	-	25.8	25.8	28.5	25.2

Table 8. Results for downstream tasks with NeRV-Enc.

tion defined by the formula:

$$\mu_i = \text{Round}\left(\frac{\mu_i - \mu_{\min}}{\text{scale}}\right) * \text{scale} + \mu_{\min}, \text{where}$$
$$\text{scale} = \frac{\mu_{\max} - \mu_{\min}}{2^b - 1}, \tag{2}$$

Here, μ_i represents a vector element, *Round* is a rounding function, b is the quantization bit length, $\mu_{\max}$ and $\mu_{\min}$ are the maximum and minimum values of vector μ, and 'scale' is the scaling factor. Additionally, we use Huffman encoding to further reduce the disk storage.

Results for Model Quantization We extend our analysis on model quantization in Tab. 9, assessing performance on three datasets: K400, Something-V2, and UCF-101.

Bits	PSNR ↑			SSIM ↑		
	K400	STH-V2	UCF101	K400	STH-V2	UCF101
32	28.4	31.6	28.1	0.808	0.862	0.817
8	28.4	31.5	28.1	0.808	0.861	0.816
7	28.3	31.5	28	0.807	0.86	0.815
6	28.1	31.2	27.9	0.802	0.855	0.811
5	27.5	30.2	27.3	0.784	0.836	0.794
4	25.6	27.7	25.6	0.712	0.759	0.725

Table 9. Ablation study on **model quantization**.

A.5. Implementation details

Video Encoding. We firstly provide training details of NeRV-Enc below.

- NeRV-Enc:
  - Video: 8 frames, frame stride evenly sample from the whole video, 256×256 resolution.
  - Batch size: 32
  - Patch size: 64
  - Position embedding dimension for NeRV: 16
  - Activation layer in NeRV: GeLU
  - Kernel size for convolution layers in NeRV: 1, 3, 3, 3
  - Upscale factor for NeRV blocks: 4, 4, 4, 4
  - Token number for NeRV layers: 4, 128, 64, 0
  - Token dimensions for NeRV layers: 256, 144, 288, 0
  - Model dimension and feed-forward dimension for transformer encoder layers: 720 and 2800 for NeRV-Enc of 47.6M, 1600 and 6400 for NeRV-Enc of larger NeRV-Enc (251M)
  - Dropout ratio in transformer encoder layers: 0 for default training, 0.15 for larger NeRV-Enc and long training
  - Optimizer: AdamW
  - Learning rate: 0.0001

Video Decoding. To assess the decoding speed of NeRV-Dec, H.264, RAM, and AV1, we employ a PyTorch dataloader to facilitate parallel decoding. We initially stack the NeRV model weights before inputting them into NeRV-Dec. Regarding video compression, we utilize the 'torchvision.io.write_video' function to store videos, applying various CRF (Constant Rate Factor) settings. For video loading, we experiment with two backends: 'decord' and 'torchvision.io.read_video', selecting the one that offers superior performance for H.264 and AV1.