Scalable Neural Video Representations with Learnable Positional Features

Subin Kim[*,1] **Sihyun Yu**[*,1] **Jaeho Lee**[2] **Jinwoo Shin**[1]

[1]Korea Advanced Institute of Science and Technology (KAIST)

[2]Pohang University of Science and Technology (POSTECH)

{subin-kim, sihyun.yu, jinwoos}@kaist.ac.kr

jaeho.lee@postech.ac.kr

Abstract

Succinct representation of complex signals using coordinate-based neural representations (CNRs) has seen great progress, and several recent efforts focus on extending them for handling videos. Here, the main challenge is how to (a) alleviate a compute-inefficiency in training CNRs to (b) achieve high-quality video encoding while (c) maintaining the parameter-efficiency. To meet all requirements (a), (b), and (c) simultaneously, we propose *neural video representations with learnable positional features* (NVP), a novel CNR by introducing "learnable positional features" that effectively amortize a video as latent codes. Specifically, we first present a CNR architecture based on designing 2D latent keyframes to learn the common video contents across each spatio-temporal axis, which dramatically improves all of those three requirements. Then, we propose to utilize existing powerful image and video codecs as a compute-/memory-efficient compression procedure of latent codes. We demonstrate the superiority of NVP on the popular UVG benchmark; compared with prior arts, NVP not only trains 2 times faster (less than 5 minutes) but also exceeds their encoding quality as 34.07→34.57 (measured with the PSNR metric), even using >8 times fewer parameters. We also show intriguing properties of NVP, *e.g.*, video inpainting, video frame interpolation, etc.[1]

1 Introduction

Recent advances in coordinate-based neural representations (CNRs) [9, 13, 17, 40, 46] have shown great promise in the field as a new paradigm for representing complex signals, including gigapixel images [29, 34], audios [40], 3D scenes [30, 33, 35], and even large city-scale street views [47]. Instead of storing signal outputs as a coordinate grid (*e.g.*, image pixels), whose memory requirement scales unfavorably in terms of resolution and dimension, CNRs represent each signal as a compactly parameterized, continuous neural network; they interpret a signal as a coordinate-to-value function and train a neural network to approximate this mapping. CNRs enjoy numerous appealing properties, including data compression [11, 12, 59], super-resolution [6], novel view synthesis [18, 22, 33, 54], and generative modeling of complex, high-dimensional data [14, 24, 41, 42, 57, 60], while being parameter-efficient interpretation of a given signal in various scenarios.

In particular, several works have attempted to exploit CNRs to interpret *video signals* [5, 23, 42, 57] by learning a neural network $f : \mathbb{R}^3 \to \mathbb{R}^3$ with $f(x, y, t) = (r, g, b)$ and exhibited their potential as a succinct representation of videos, as well as providing numerous applications including video generative modeling [42, 57], video compression [5, 59], and video super-resolution [7]. They

[*]Equal contribution.

[1]Videos are available at the website https://subin-kim-cv.github.io/NVP.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Figure 1: Reconstruction results on Jockey in UVG-HD after training each model for "1 minute" with a single NVIDIA V100 32GB GPU. NVP can capture the detail of a video containing dynamic motions, *e.g.*, the legs of a running horse, while the prior methods generate blurry artifacts.

observe that conventional CNR architectures [40, 46] often fail to encode large-scale videos due to their complex temporal dynamics accompanied by large spatial variations, but the problem can be remarkably mitigated by designing CNR architectures specialized for videos. For instance, Chen et al. [5] proposes a CNR structure that focuses on the continuous modeling of the video signal only along the temporal dimension, allowing for more radical variations along spatial axes; it exhibits a comparable encoding quality to existing powerful video codecs (*e.g.*, H.264 [51], HEVC [43]) while enjoying lots of intriguing properties (*e.g.*, denoising and video frame interpolation).

However, CNRs suffer from a severe compute-inefficiency,[2] limiting their scalability to encode real-world, large-scale videos despite their advantages. To alleviate this issue, several works [26, 34, 37] have proposed new CNR architectures by separating a CNR f into two parts; $f = h_\phi \circ g_\theta$ for a coordinate-to-latent mapping $g_\theta(x, y, t) = \mathbf{z}$ and a latent-to-RGB mapping $h_\phi(\mathbf{z}) = (r, g, b)$. They construct g_θ as an embedding function defined with *latent grids* $\mathbf{U}_\theta$ in which the shape resembles the grid interpretation of a given signal (*e.g.*, a 2D array of C-dimensional latent codes $\mathbf{U}_\theta \in \mathbb{R}^{H \times W \times C}$ for image pixels) rather than as a neural network. These approaches have shown a promise in compute-efficiency due to the strong locality induced by a grid structure of $\mathbf{U}_\theta$; however, they result in another problem: these architectures severely sacrifice the parameter-efficiency since the parameter size of θ can be very large, growing proportionally to both the input coordinate dimension and the signal resolution. In this paper, we focus on developing video CNRs that are the best of both worlds: achieving high-quality encoding and the compute-/parameter-efficiency simultaneously.

Contribution. We introduce *neural video representations with learnable positional features* (NVP), a novel CNR for videos. NVP avoids requiring a single giant full-dimensional 3D array in g_θ by presenting *learnable positional features* that effectively amortize a given video as "2D and 3D" latent grids with succinct parameters. Specifically, we decompose the coordinate-to-latent mapping as

$$g_\theta = \underbrace{g_{\theta_{\texttt{xy}}} \times g_{\theta_{\texttt{xt}}} \times g_{\theta_{\texttt{yt}}}}_{\text{2D keyframes}} \times \underbrace{g_{\theta_{\texttt{xyt}}}}_{\text{3D sparse features}}, \qquad \text{for} \quad \theta := (\theta_{\texttt{xy}}, \theta_{\texttt{xt}}, \theta_{\texttt{yt}}, \theta_{\texttt{xyt}}),$$

where we present two types of latent grids for constructing these mappings (see Figure 2).

- *Latent keyframes*: We first design "image-like" 2D latent grids $\mathbf{U}_{\theta_{\texttt{xy}}}, \mathbf{U}_{\theta_{\texttt{xt}}}, \mathbf{U}_{\theta_{\texttt{yt}}}$ for $g_{\theta_{\texttt{xy}}}, g_{\theta_{\texttt{xt}}}, g_{\theta_{\texttt{yt}}}$ (respectively) that learn the representative video contents across *each* spatio-temporal axis and dramatically improve the parameter efficiency of NVP.[3]
- *Sparse positional features*: We then introduce a "video-like" 3D latent grid $\mathbf{U}_{\theta_{\texttt{xyt}}}$ for $g_{\theta_{\texttt{xyt}}}$, whose size is much smaller than the original video pixels, but effectively encodes video details locally.

[2] Measured with a single NVIDIA V100 32GB GPU, NeRV [5] takes at least 15 GPU hours to encode a single video of 600 frames with a 1920×1080 resolution for the desired quality.

[3] Such a spatio-temporal consideration is different from conventional approaches for specifying keyframes deterministically across only the temporal direction.

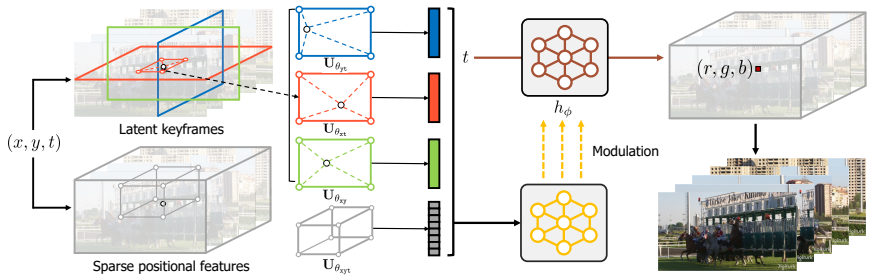


Figure 2: Overall illustration of our NVP. At a given space-time coordinate, NVP computes a latent vector from latent keyframes (see Figure 19 in Appendix E for details) and sparse positional features. The latent vector is passed through a neural network to compute the corresponding RGB output.

Moreover, we propose a compute-/memory-efficient compression procedure to further reduce the parameter θ by incorporating existing image and video codecs, *e.g.*, JPEG [49] for images and HEVC [43] for videos, respectively. In particular, we treat 2D and 3D latent grids like the image or video pixels and utilize powerful compression pipelines for them. Our compression scheme does not require any re-training of trained parameters, which significantly increases compute-efficiency to prior approaches to compressing CNR parameters but remarkably maintains the encoding quality. We also remark that such a compression approach is not applicable to the hashing-based latent grid of the prior method [34], while ours is "collision-free" and maintains the video- or image-like structures. Finally, for the choice of h_ϕ, we suggest using a modulated network (with respect to the temporal coordinate) to improve the encoding quality of videos that contain dynamic motions.

We verify the effectiveness of our method on the popular UVG benchmark [32]. In particular, NVP achieves the peak signal-to-noise ratio (PSNR; higher is better) metric of 34.57 in 5 minutes (with a single NVIDIA V100 32GB GPU): it is achieved $>$2 times faster, even with using $>$8 times fewer parameters than the state-of-the-art on compute-efficiency that reaches 34.07 in 10 minutes. Moreover, compared with prior arts on encoding quality, our method improves the learned perceptual image patch similarity (LPIPS [58]; lower is better) as 0.145$\rightarrow$0.102 ($+$29.7%) with a similar number of parameters while requiring $\sim$72.5% less training time. We also show numerous compelling properties of NVP, *e.g.*, video inpainting, video frame interpolation, super-resolution, compression, and consistent frame-wise encoding results without deviating quality.

2 Related work

Coordinate-based neural representations. Coordinate-based neural representations (CNRs), also known as implicit neural representations or neural fields, have emerged as a new paradigm for representing complex, continuous signals. They propose to encode signals through a neural network, typically a multilayer perceptron (MLP) combined with high-frequency sinusoidal activations [40] or Gaussian activations [9]. Prior works have focused on utilizing neural fields on various complicated data, *e.g.*, gigapixel images [29, 34], 2D videos [5, 42, 57], 3D static scenes, [30, 33, 35], and 3D dynamics scenes [23, 39, 52]. In particular, most approaches have focused on constructing CNR architectures for encoding 3D scenes [3, 15, 47] and exhibited how employing specialized prior knowledge for a given signal domain in the architecture can remarkably boost the encoding quality. Despite these successes, extending CNRs for videos is yet under-explored. In this paper, we aim to move toward developing a CNR for videos by exploiting their unique temporal properties.

Hybrid CNRs. Rather than solely designing a neural network of coordinate-to-RGB mapping, hybrid CNRs incorporate learnable latent codes that follow a grid structure, *e.g.*, image CNRs combined with 2D latent spatial grids [6, 31], and 3D scene (or shape) CNRs with latent cubic grids [4, 8, 19, 26, 29, 37]. Specifically, they compute a latent vector using the grid-structured latent code and pass it through a neural network to compute the signal output at a given input coordinate. Such approaches have shown significant efficiencies in training time and encoding quality due to the powerful locality induced by grid-shaped latent codes. However, the number of parameters required for latent grid-based representations easily grows proportionally to the input coordinate dimension or data resolution, limiting the scalability of hybrid CNRs. Remarkably, some of the recent approaches have exhibited this inefficiency can be significantly mitigated by considering multi-level (or progressive) structures for latent grids [26, 29, 44, 45]. While prior works primarily focus on encoding images or 3D scenes, we aim to design parameter-efficient hybrid CNRs for videos.

3 NVP: Neural video representations with learnable positional features

We first formulate our problem setup as follows. Given a video signal $\mathbf{v} := (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_T)$ consisting of T video frames, the goal is to find a compact neural representation $f_\mathbf{w}$ with parameters $\mathbf{w}$, from which the original video $\mathbf{v}$ can be reconstructed with high quality. Here, the quality can be defined using various distortion metrics, *e.g.*, peak signal-to-noise ratio (PSNR) [16] and LPIPS [58], for evaluating a reconstruction quality and a perceptual similarity, respectively.

To achieve this goal, we take an approach based on *coordinate-based neural representations* (CNRs)—a paradigm where each datum (*e.g.*, video) is parameterized as a neural network of coordinate mapping. In particular, we aim to represent the given video using a neural network $f_\mathbf{w} : \mathbb{R}^3 \to \mathbb{R}^3$, which maps the space-time coordinates (x, y, t) of the video to corresponding RGB values (r, g, b), where $f_\mathbf{w}$ is optimized with reconstruction objectives, *e.g.*, mean-squared error. Such an approach has significant potential, as CNRs have shown to encode other continuous, complex signals (*e.g.*, 3D scenes) [40, 46] compactly while enjoying lots of intriguing properties, *e.g.*, super-resolution [6] and denoising [5]. However, CNRs have suffered from tremendous time costs for training, and even in the case of videos, it is difficult to achieve high-quality encodings if one utilizes conventional CNR architectures that overlook the complex spatio-temporal dynamics of videos [5]. Our contribution lies in resolving these issues by designing "learnable positional features" that succinctly encode a video as latent codes with high quality and keeping their compute-/parameter-efficiency intact.

In the rest of this section, we provide a detailed description of each component in NVP. In Section 3.1, we explain the architecture of NVP. We then describe our compression procedure in Section 3.2.

3.1 Architecture

We design our video CNR $f_\mathbf{w}$ as a composition of two functions with a parameterization $\mathbf{w} := (\theta, \phi)$: a coordinate-to-latent mapping g_θ and a latent-to-RGB mapping h_ϕ. Here, we decompose the coordinate-to-latent-mapping as $g_\theta = g_{\theta_{\texttt{xy}}} \times g_{\theta_{\texttt{xt}}} \times g_{\theta_{\texttt{yt}}} \times g_{\theta_{\texttt{xyt}}}$ with $g_\theta(x, y, t) = (\mathbf{z}_{\texttt{xy}}, \mathbf{z}_{\texttt{xt}}, \mathbf{z}_{\texttt{yt}}, \mathbf{z}_{\texttt{xyt}})$ (for $\theta := (\theta_{\texttt{xy}}, \theta_{\texttt{xt}}, \theta_{\texttt{yt}}, \theta_{\texttt{xyt}})$), where each $g_{\theta_{\texttt{xy}}}$, $g_{\theta_{\texttt{xt}}}$, $g_{\theta_{\texttt{yt}}}$ is formalized with image-like 2D latent spatial grids $\mathbf{U}_{\theta_{\texttt{xy}}}$, $\mathbf{U}_{\theta_{\texttt{xt}}}$, $\mathbf{U}_{\theta_{\texttt{yt}}}$(respectively) and $g_{\theta_{\texttt{xyt}}}$ is designed with a video-like sparse 3D latent grid $\mathbf{U}_{\theta_{\texttt{xyt}}}$. We then present the latent-to-RGB mapping $h_\phi(\mathbf{z}_{\texttt{xy}}, \mathbf{z}_{\texttt{xt}}, \mathbf{z}_{\texttt{yt}}, \mathbf{z}_{\texttt{xyt}}) = (r, g, b)$ to be a multi-layer perceptrion (MLP) modulated by another neural network. To explain our architecture, we assume all the input coordinate (x, y, t) of g_θ (and $f_\mathbf{w}$) is in $[0, 1]^3 \subset \mathbb{R}^3$ without loss of generality.

Learnable latent keyframes. At a high level, learnable latent keyframes $\mathbf{U}_{\theta_{\texttt{xy}}}$, $\mathbf{U}_{\theta_{\texttt{xt}}}$, $\mathbf{U}_{\theta_{\texttt{yt}}}$are image-like 2D latent grids learned to capture the common representative contents in a given video $\mathbf{v}$ across each t-, y-, x-axis, respectively. For a given input coordinate (x, y, t), we compute latent vectors $\mathbf{z}_{\texttt{xy}}$, $\mathbf{z}_{\texttt{xt}}$, $\mathbf{z}_{\texttt{yt}}$ from $\mathbf{U}_{\theta_{\texttt{xy}}}$, $\mathbf{U}_{\theta_{\texttt{xt}}}$, $\mathbf{U}_{\theta_{\texttt{yt}}}$individually. We explain our keyframe only with $\mathbf{U}_{\theta_{\texttt{yt}}}$by letting $\mathbf{U} := \mathbf{U}_{\texttt{yt}}$ for simplicity, but note that other keyframes $\mathbf{U}_{\theta_{\texttt{xy}}}$, $\mathbf{U}_{\theta_{\texttt{xt}}}$ operate in the similar manner.

Formally, $\mathbf{U}$ is L 2D spatial grids of C-dimensional latent codes u_{ij}, whose resolution is $H_l \times W_l$:

$$\begin{aligned} \mathbf{U} &:= (U_1, \ldots, U_L), \\ U_l &:= (u^l_{ij}) \in \mathbb{R}^{H_l \times W_l \times C} \quad \text{for } l = 1, \ldots, L, \\ u_{ij} &\in \mathbb{R}^C \qquad \text{for } 1 \le i \le H_l,\ 1 \le j \le W_l. \end{aligned}$$

Here, the keyframe follows an L-level multi-resolution structure, *i.e.*, for each level l, the height H_l and the width W_l become different as $H_l = \lfloor \gamma^{l-1} H_1 \rfloor$ and $W_l = \lfloor \gamma^{l-1} W_1 \rfloor$ with fixed $\gamma > 1$ and $H_1, W_1 > 0$, where $\lfloor \cdot \rfloor$ indicates the floor function of the input. Since $\mathbf{U} = \mathbf{U}_{\theta_{yt}}$ is shared over the x-axis, we compute the latent vector $\mathbf{z}_{\texttt{yt}} := (z^1_{\texttt{yt}}, \ldots, z^L_{\texttt{yt}})$ by considering only the value of y and t at a given coordinate (x, y, t). Specifically, for $l = 1, \ldots, L$, each $z^l_{\texttt{yt}}$ is a linearly interpolated vector of four vectors in the spatial grid U_l, where the indices of these vectors are chosen as the closest ones to the relative position of the input coordinate $(y, t) \in [0, 1]^2$:

$$\begin{aligned} (m, n) &= (\lfloor y H_l \rfloor, \lfloor t W_l \rfloor) \quad \text{for } l = 1, \ldots, L, \\ z^l_{\texttt{yt}} &= \texttt{lerp}\Big((y H_l - m, t W_l - n); u^l_{mn}, u^l_{m,n+1}, u^l_{m+1,n}, u^l_{m+1,n+1} \Big), \end{aligned}$$

where `lerp` indicates a linear interpolation operation at the input coordinate between given vectors.

Note that we *learn* the keyframe as the latent codes, unlike conventional approaches that specify the keyframe in a deterministic manner from T video frames $(\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_T)$; it encourages to capture

the representative contents of the video over each spatio-temporal direction better. We also remark that our architecture involves two keyframes $\mathbf{U}_{\theta_{\texttt{xt}}}, \mathbf{U}_{\theta_{\texttt{yt}}}$ that are considered across spatial axes. Considering these keyframes may not be beneficial in the RGB space as the spatial variation of video pixels is often large. However, in our approach, these keyframes are learned under a more flexible, continuous latent space, and thus the representative frames can even be found in these spatial directions in such a space while encoding the RGB outputs of the video accurately.

Finally, recall that $\mathbf{U}_{\theta_{\texttt{xy}}}$, $\mathbf{U}_{\theta_{\texttt{xt}}}$, $\mathbf{U}_{\theta_{\texttt{yt}}}$consist of multi-resolution spatial grids, *i.e.*, the resolution of spatial grids in each latent keyframe grows from coarse to fine. Since natural scenes often include repeating patterns in various scales, *e.g.*, a scene of flowers of different sizes, this multi-resolution architecture promotes learning common patterns with reduced memory and computation costs, which is also validated in Müller et al. [34].

Sparse positional features. Given a space-time coordinate (x, y, t), we compute the latent vector $\mathbf{z}_{\texttt{xyt}}$with $\mathbf{U}_{\theta_{\texttt{xyt}}}$that represents the local details of the video at this input position. Here, $\mathbf{U}_{\theta_{\texttt{xyt}}} \coloneqq (u_{ijk}) \in \mathbb{R}^{H\times W\times S\times D}$ is a 3D *sparse* grid of D-dimensional latent codes, *i.e.*, the 3D grid size $H \times W \times S$ is much smaller than the size of the video pixels of a 3D RGB grid:

$$\mathbf{U}_{\theta_{\texttt{xyt}}} \coloneqq (u_{ijk}) \in \mathbb{R}^{H\times W\times S\times D},\ u_{ijk} \in \mathbb{R}^D \quad \text{for } 1 \le i \le H,\ 1 \le j \le W,\ 1 \le k \le S.$$

To evaluate the latent vector $\mathbf{z}_{\texttt{xyt}}$, we concatenate $h \times w \times s$ latent codes in $\mathbf{U}_{\theta_{\texttt{xyt}}}$that their indices are near the relative position of the input (x, y, t):

$$\begin{aligned}(m, n, k) &= (\lfloor xH \rfloor, \lfloor yW \rfloor, \lfloor tS \rfloor),\\ \mathbf{z}_{\texttt{xyt}} &= (u_{mnk}, \ldots, u_{m+h-1,n+w-1,k+s-1})\end{aligned}$$

where $h, w, s > 0$ are given as hyperparameters.

The locality of $\mathbf{U}_{\theta_{\texttt{xyt}}}$as 3D latent codes dramatically alleviates the compute-inefficiency in fitting videos, in contrast to conventional CNRs where the entire parameters are shared (as a neural network) for arbitrary input coordinates (x, y, t) and thus require a significant training cost. Moreover, recall that we construct $\mathbf{U}_{\theta_{\texttt{xyt}}}$as a sparse 3D grid of latent codes; remarkably, $\mathbf{U}_{\theta_{\texttt{xyt}}}$efficiently captures the video details even if the size is smaller than the number of video pixels since the common contents of a given video are effectively encoded with the latent keyframes $\mathbf{U}_{\theta_{\texttt{xy}}}$, $\mathbf{U}_{\theta_{\texttt{xt}}}$, $\mathbf{U}_{\theta_{\texttt{yt}}}$.

Note that we design $\mathbf{U}_{\theta_{\texttt{xyt}}}$as a sparse 3D grid; each single u_{ijk} should represent a wide area of videos solely without concatenation. As the single latent vector may lack expressive power to represent such a wide range and may result in a non-smooth transition as the CNR output. Hence, we mitigate this issue by concatenating multiple vectors near each other (see Figure 10 in Section 4.3).

Instead of directly selecting near latent codes from $\mathbf{U}_{\theta_{\texttt{xyt}}}$, one may consider upsampling of $\mathbf{U}_{\theta_{\texttt{xyt}}}$using linear interpolation before selecting the close latent codes at a given coordinate. Such an interpolation further helps each latent code to learn smoother representations and also generalizes to representing video frames at unseen coordinates during training. Meanwhile, this upsampling requires more computing time compared to the computational cost of other modules in NVP, and thus faces a trade-off between the smoothness and compute-efficiency in training (see Table 4 in Section 4.3).

Modulated implicit function. With the latent vector $\mathbf{z} \coloneqq [\mathbf{z}_{\texttt{xy}}, \mathbf{z}_{\texttt{xt}}, \mathbf{z}_{\texttt{yt}}, \mathbf{z}_{\texttt{xyt}}]$ evaluated from g_θ, a naïve design choice of the latent-to-RGB mapping h_ϕ is to utilize a MLP that maps $\mathbf{z}$ to the corresponding RGB output (r, g, b). However, if a video contains temporally dynamic motions, we found such a simple MLP architecture occasionally lacks expressive power and can be difficult to capture the complex dynamics of the given video, even with the large network size of h_ϕ.

To circumvent this issue, we design h_ϕ to be a K-layer MLP (coined as a synthesizer network) modulated by another modulator network [31]: where the latent vector $\mathbf{z}$ and the time coordinate t are passed through the modulator and the synthesizer, respectively. Here, the modulator network utilizes piecewise linear activations (*e.g.*, ReLU), where the synthesizer uses sinusoidal activations. Specifically, an RGB output (r, g, b) of the latent vector $\mathbf{z}$ (from (x, y, t)) is computed as follows:

$$\begin{aligned}\boldsymbol{\alpha}_0 &= t,\\ \boldsymbol{\alpha}_k &= \mathbf{z}_k \odot \sin(\boldsymbol{A}_k \boldsymbol{\alpha}_{k-1} + \boldsymbol{b}_k) \quad \text{for } k = 1, \ldots, K-1,\\ (r, g, b) &= \boldsymbol{A}_K \boldsymbol{\alpha}_{K-1} + \boldsymbol{b}_K,\end{aligned}$$

where $\boldsymbol{A}_k$, $\boldsymbol{b}_k$ are weights and biases of k-th layer of the synthesizer, $\boldsymbol{z}_k$ is k-th hidden feature of the modulator, and $\odot$ denotes an element-wise product. It helps to achieve high-quality encoding rapidly in early training iterations and often at the convergence than a naive MLP (See Table 3 for details).

Table 1: PSNR, FLIP, and LPIPS of different CNRs to encode videos in UVG-HD under each encoding time. ↑ and ↓ denote higher and lower values are better, respectively. Subscripts denote standard deviations, and bolds indicate the best results. * indicates applying the method without the corresponding compression scheme. We report the BPP values of NeRV without compressing parameters if the encoding time is $\leq$ 1 hour since the NeRV's compression requires a longer time. On the other hand, the compression procedure of NVP only takes less than 1 minute, but for a fair comparison, we do not apply it to NVP as well whenever NeRV is not compressed.

Encoding time	Method	BPP	PSNR (↑)	FLIP (↓)	LPIPS (↓)
∼5 minutes	Instant-ngp [34]	7.580	33.15±3.19	0.090±0.034	0.226±0.112
	NeRV-S* [5]	1.072	24.16±5.17	0.219±0.097	0.542±0.180
	NVP-S* (ours)	0.901	**34.57±2.62**	**0.075±0.021**	**0.190±0.100**
∼10 minutes	Instant-ngp [34]	7.580	34.07±3.01	0.082±0.030	0.204±0.105
	NeRV-S* [5]	1.072	26.53±5.92	0.176±0.088	0.460±0.184
	NVP-S* (ours)	0.901	**35.79±2.31**	**0.065±0.016**	**0.160±0.098**
∼1 hour	Instant-ngp [34]	7.580	35.69±2.72	0.071±0.025	0.162±0.090
	NeRV-S* [5]	1.072	33.26±4.31	0.094±0.038	0.240±0.112
	NVP-S* (ours)	0.901	**37.61±2.20**	**0.052±0.011**	**0.145±0.106**
∼15 hours	SIREN [40]	0.284	27.20±3.77	0.169±0.059	0.409±0.124
	FFN [46]	0.284	28.18±3.62	0.153±0.055	0.442±0.126
	Instant-ngp [34]	0.229	28.81±3.48	0.155±0.057	0.390±0.135
	NeRV-S [5]	0.201	36.14±3.97	**0.067±0.023**	0.163±0.101
∼8 hours	**NVP-S (ours)**	0.210	**36.46±2.18**	**0.067±0.017**	**0.135±0.083**
>40 hours	SIREN [40]	0.284	26.09±3.88	0.175±0.082	0.486±0.188
	FFN [46]	0.284	29.53±3.44	0.135±0.052	0.391±0.124
	Instant-ngp [34]	0.436	29.98±3.39	0.138±0.051	0.358±0.140
	NeRV-L [5]	0.485	35.00±3.31	0.079±0.020	0.145±0.100
∼11 hours	**NVP-L (ours)**	0.412	**37.47±2.08**	**0.062±0.017**	**0.102±0.061**

3.2 Compression procedure

Recall that we aim to find "compact" video CNRs; several works have focused on reducing the number of coordinate-based neural representations parameters (or bits) after training while maintaining their performance. In particular, they have relied on exploiting existing well-known techniques for neural network compression, *e.g.*, exploiting magnitude pruning [5, 21] or quantization [5, 59], and exhibited considerable results. However, these approaches mainly involve a re-training of CNR parameters, requiring severe computation costs, and thus are not suitable for practical scenarios.

Instead, we propose a compression pipeline for NVP, which does not require re-training, yet significantly reduces the number of bits while preserving the video quality. Our main idea is to incorporate existing image and video codecs that have shown their promises for the compression of given pixels. In particular, we focus on compressing keyframes $\mathbf{U}_{\theta_{\mathtt{xy}}}$, $\mathbf{U}_{\theta_{\mathtt{xt}}}$, $\mathbf{U}_{\theta_{\mathtt{yt}}}$, and sparse positional features $\mathbf{U}_{\theta_{\mathtt{xyt}}}$, as the parameter size of the modulated implicit function h_ϕ is neglectable compared with those. Specifically, we quantize $\mathbf{U}_{\theta_{\mathtt{xyt}}}$ and $\mathbf{U}_{\theta_{\mathtt{xy}}}$, $\mathbf{U}_{\theta_{\mathtt{xt}}}$, $\mathbf{U}_{\theta_{\mathtt{yt}}}$ as 3D/2D grids of 8-bit latent codes and regard them as video and image pixel grids, where the number of the channel becomes the dimension of latent codes. Based on these interpretations, we compress these latent codes by utilizing existing video and image codecs, *e.g.*, HEVC [43] for videos and JPEG [49] for images. Intriguingly, we found this procedure can significantly reduce the parameters while notably maintaining the video quality without any fine-tuning of the latent-to-RGB mapping h_ϕ (See Section 4.3).

4 Experiments

We verify the effectiveness of our framework on UVG-HD [32], a representative benchmark for evaluating video encodings. Experimental results demonstrate that our neural video representations with learnable positional features (NVP) simultaneously improves the overall performance by (a) alleviating a compute-inefficiency in training, (b) achieving high-quality video encoding, and (c) maintaining parameter-efficiency. We also show applications of our NVP, including video inpainting and spatio-temporal interpolation. Finally, we conduct ablation studies to validate each component.

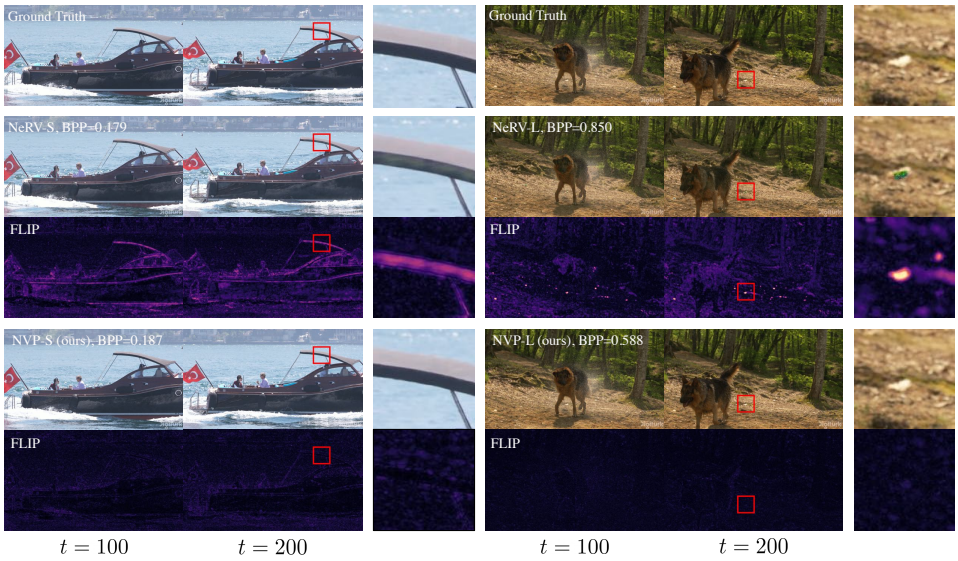


Figure 3: Illustration of reconstructions on Yachtride (left), and ShakeNDry (right) in UVG-HD. FLIP indicates the output of evaluating its metric. The red box is zoomed in as the image at the right.

Evaluation. We follow similar setups on prior work [5] proposing coordinate-based neural representations (CNRs) for videos. All reported numbers are averaged over 7 videos in UVG-HD: Beauty, Bosphorus, HoneyBee, Jockey, ReadySetGo, ShakeNDry, and Yachtride, along with the standard deviations unless otherwise specified. We also use the Big Buck Bunny video, which is used in NeRV [5]. For quantitative evaluation, we use the following metrics: peak signal-to-noise ratio (PSNR) for reconstruction quality, LPIPS [58], FLIP [2], and SSIM [50] for perceptual similarity, where all metrics are evaluated in a frame-wise manner and averaged over the whole video. We evaluate these metrics on different bits-per-pixel (BPP; lower is better) to evaluate the parameter efficiency; see Appendix A.1 for more description of the evaluation.

Implementation details. All main experiments, including baselines, are processed with a single GPU (NVIDIA V100 32GB) and 28 instances from a virtual CPU (Intel® Xeon® Platinum 8168 CPU @ 2.70GHz), where it takes at most $\sim$11 hours to run our method and $\sim$2 days to run other baselines. Moreover, we denote NVP-S and NVP-L as the model with latent code dimensions of sparse positional features to be 2 and 4, respectively; see Appendix A.2 for more details.

Baselines. We compare our method with SIREN [40], and FFN [46], which are well-known signal-agnostic CNR architectures, Instant-ngp [34] for state-of-the-art CNRs on compute-efficiency, and NeRV [5], which is a CNR specialized for videos. For all of the baseline methods, we follow their reported experimental setups. In particular, for NeRV [5], we use two configurations provided in the official implementation: NeRV-S and NeRV-L for a small and a large model, respectively. See Appendix B for a detailed description of baseline methods.

4.1 Main results

Figure 1, Figure 3, and Table 1 summarize quantitative and qualitative results of NVP and baselines. Remarkably, as shown in Figure 1, NVP can accurately capture dynamic motions and high-frequency details of a video, *e.g.*, the legs of a running horse, while prior state-of-the-art CNRs fail to achieve and show a blurry artifact. We also emphasize such a high-quality encoding is accomplished in "less than 1 minute", which supports the superior compute-efficiency of NVP in training.

Moreover, Table 1 verifies the effectiveness of NVP with quantitative evaluations, which outperforms all other baselines at varying encoding times from $\sim$5 minutes to $>$40 hours. In particular, NVP significantly improves LPIPS compared with previous methods, demonstrating how the encoded videos are perceptually similar to ground-truth videos. Such a result is also confirmed in Figure 3; NeRV shows a distortion that some pixels significantly deviate from the ground-truth outputs, while our method does not suffer from such artifacts. We also note that the variance of NVP is relatively small compared with other baselines, which shows the robustness of videos with diverse scenes and motion. See Appendix C and D for video-wise results and discussion on decoding time, respectively.

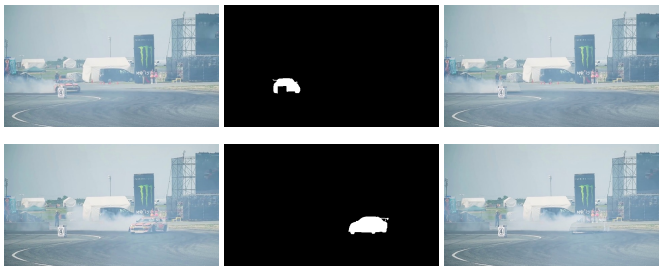


Figure 4: Video inpainting result of NVP on the drift-chicane video in DAVIS 2017 [38] to remove the masked car.

Table 2: Quantitative interpolation result of different methods on Big Buck Bunny measured with PSNR, LPIPS, and SSIM metrics. Bold indicates the best result.

Metric	NeRV [5]	NVP (ours)
PSNR (↑)	23.05	**33.76**
LPIPS (↓)	0.480	**0.311**
SSIM (↑)	0.690	**0.960**

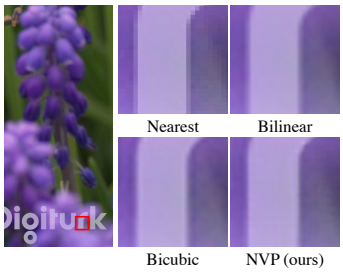


Figure 5: Super-resolution result (×8) of NVP (HoneyBee).

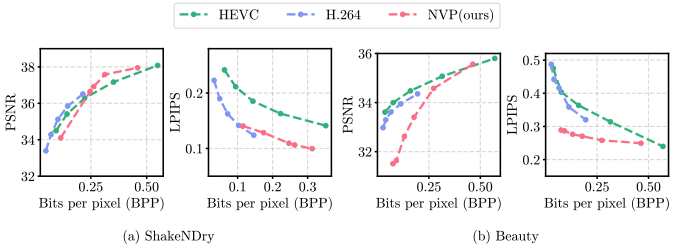


Figure 6: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on (a) the ShakeNDry video and (b) the Beauty video in UVG-HD.

4.2 Applications

In this section, we provide several applications of our method, NVP, as video CNRs. For better, playable illustrations and qualitative results, please refer to our project page.

Video inpainting. Intriguingly, our method has the capability of video inpainting, *i.e.*, the desired moving object in the video can be naturally removed by capturing shared video contents with learnable keyframes. Figure 4 visualizes the illustration of inpainting results from NVP; as shown in this figure, one can see the car is removed where such parts are filled with a natural background.

Video frame interpolation. Since video CNRs approximate videos as temporally continuous signals, they should interpolate among two different frames at an arbitrary time, even if such a frame does not exist in the training dataset. To validate the interpolation capability of NVP, we provide the quantitative results on Big Buck Bunny sequences; our method shows better interpolation results measured with various metrics, such as PSNR and LPIPS.

Video super-resolution. We remark that NVP encodes a given video as a *spatio-temporally* continuous signal. Thus, our method can interpolate the frames across spatial directions, *i.e.*, frame-wise super-resolution. Figure 5 exhibits how well NVP smoothly interpolates video frames across spatial directions while preserving the sharp edges, compared with naïve upsampling methods.

Video compression. Recall that one of the major advantages of CNRs is their succinct encoding to represent a given signal; one may consider utilizing CNRs for video compression [5, 11, 12]. To verify the potential of our method on video compression, we compare the quality of compressed videos from NVP with the ones from the current state-of-the-art video codecs. As shown in Figure 6, compressed videos from NVP show the comparable reconstruction quality (measured with PSNR metrics) while outperforming perceptual similarity (measured with LPIPS metrics).

4.3 Ablation studies

Effect of architecture components. To verify the effectiveness of each component, we train our model with all the videos in UVG-HD by removing each component while maintaining the total number of parameters, then measure PSNR metrics from these models. Table 3 summarizes the effect of three different architecture components. Without any of the components consisting of our positional features, the reconstruction quality gets dramatically worse, which validates how NVP

Table 3: PSNR values of each component of NVP: learnable keyframes, sparse feature, and modulation at 1,500 (1.5K) and 150,000 (150K) iterations. Bold indicates the scores within one standard deviation from the highest average score.

Keyframes	Sparse feat.	Module.	# Params.	1.5K	150K
✗	✓	✓	136M	29.95±2.69	31.21±2.80
✓	✗	✓	138M	29.88±4.99	32.44±4.48
✓	✓	✗	147M	32.15±3.08	**38.04**±2.27
✓	✓	✓	136M	**34.85**±2.69	**38.89**±2.11

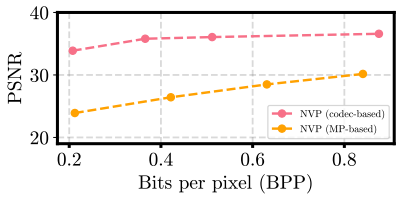


Figure 7: Rate-distortion plot of different compression strategies on ReadySetGo.

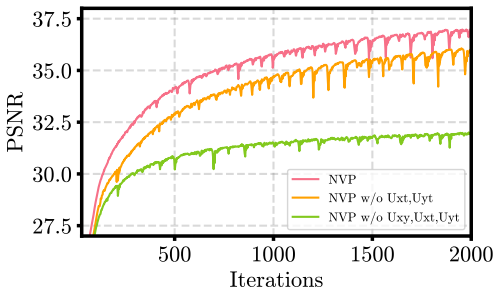


Figure 8: Convergence plot of NVP under different keyframe choices on the Jockey video.

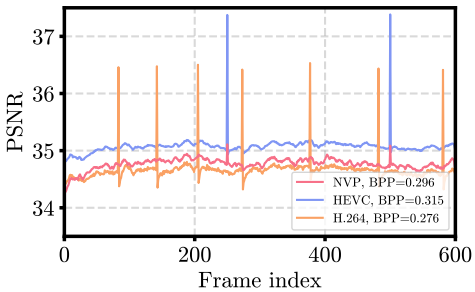


Figure 9: Frame-wise encoding quality of NVP and existing video codecs on Beauty.

succinctly encodes a given video as latent codes. We also note that the modulation not only improves the final encoding quality but also notably achieves high-quality encoding in its early training epochs.

Compression procedure. To validate the effectiveness of our compression scheme, we compare the encoding quality of (a) NVP with our compression scheme and (b) NVP compressed through magnitude-based pruning. Figure 7 shows the proposed compression pipeline outperforms conventional magnitude-based pruning under various BPP values. We also remark that our compression method does not require re-training and is thus much more time-efficient.

Analysis of non-temporal keyframes. Recall that we additionally design keyframes across spatial directions, unlike conventional approaches that designate the keyframe over only the temporal direction. To validate the effect of such keyframes, we compare NVP with (a) NVP without spatial keyframes and (b) NVP without all of the keyframes in Figure 8, where the number of parameters is equally set for a fair comparison. While utilizing latent learnable keyframes only over temporal direction is already fairly effective for high-quality encoding, one can observe the consideration of keyframes across other directions provides a further improvement.

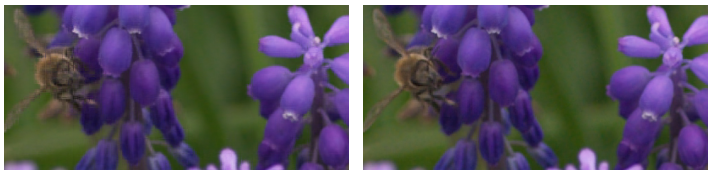
Consistent frame-wise encoding quality. Existing keyframe-based compression approaches often suffer from inconsistent encoding quality: the frame-wise quality of the compressed video highly depends on whether it is the designated keyframe or not. In contrast, NVP *learns* the keyframes and does not have this problem. Figure 9 validates the result: our method exhibits consistent encoding performance while conventional popular video codecs show several peaks that the reconstruction quality (measured with PSNR metrics) highly deviates from others.

Effect of the concatenation of latent codes in sparse positional features. To validate the effectiveness of design choice on extracting latent features from sparse positional features $\mathbf{U}_{\theta_{\texttt{xyt}}}$, we compare the reconstructions from NVP with and without concatenation of $\mathbf{U}_{\theta_{\texttt{xyt}}}$. As shown in Figure 10, the concatenation of latent codes u_{ijk} in $\mathbf{U}_{\theta_{\texttt{xyt}}}$ indeed mitigates non-smooth transitions between latent codes and captures sharp details in a given video better. In particular, without the concatenation, it results in undesirable artifacts (*e.g.*, showing discontinuous borders), validating our concatenation scheme for constructing latent representations from sparse positional features.

Effect of upsampling of sparse positional features. We also examine the effect of upsampling of the sparse positional features $\mathbf{U}_{\theta_{\texttt{xyt}}}$. Figure 11 shows the result: linear interpolation of $\mathbf{U}_{\theta_{\texttt{xyt}}}$ exhibits more smooth patterns for unseen coordinates during training. Meanwhile, we note that the upsampling requires 1.61 times more training time per iteration due to the additional computation bottleneck (see Table 4); however, regardless of upsampling, we remark that NVP still achieves notable compute-efficiency compared with prior state-of-the-art methods (such as Chen et al. [5]).

Without Concatenation of $\mathbf{U}_{\theta_{\text{xyt}}}$ | With Concatenation of $\mathbf{U}_{\theta_{\text{xyt}}}$ | Ground Truth

Figure 10: Reconstruction results on ReadySetGo in UVG-HD. Concatenation of sparse positional features captures sharp details (*e.g.*, a fence) better.



Without Upsampling of $\mathbf{U}_{\theta_{\text{xyt}}}$ | With Upsampling of $\mathbf{U}_{\theta_{\text{xyt}}}$

Figure 11: Comparison of super-resolution result ($\times 8$) on HoneyBee.

Table 4: Training time per iteration with/without upsampling of sparse positional features.

Upsampling	Time
✗	0.291s
✓	0.469s

5 Discussion and conclusion

We proposed NVP, a new coordinate-based neural representation (CNR) to encode videos as succinct latent codes. Our main idea is to decompose a video into "image-like" and "video-like" structures to learn coordinate-to-latent mapping efficiently. Extensive experiments have verified the effectiveness of NVP on all the parameter-/compute-efficiency and the encoding quality. We hope our method will facilitate various future research directions in the CNR area.

Limitations and future works. Each video contains different scenes and motions so that it can be either static or dynamic, yet we utilize the same hyperparameters and architectures for encoding any video. Although such a video-agnostic design is fairly effective and outperforms prior works, we believe the video-wise consideration of the architecture and hyperparameter can remarkably boost the performance further. Moreover, we have shown the potential of utilizing powerful image and video codes for compressing latent codes in NVP; extending such codecs to be specialized for the compression of latent codes should be an interesting direction.

Negative social impacts. A side effect of CNRs is their potential unexpected behavior on encoding; they may cause undesirable artifacts in representing the given signal but are challenging to predict due to the under-explored behavior of training CNRs. Furthermore, in the case of representing videos, the encoded videos may suffer from severe distortions and conceivably cause ethical problems. In this respect, such behaviors should be extensively and carefully investigated and mitigated to exploit CNRs as the standard for encoding videos in real-world situations.

Acknowledgments and Disclosure of Funding

We would like to thank Younggyo Seo, Jihoon Tack, Jongheon Jeong, Sukmin Yun, Jongjin Park, Junsu Kim, SeongHyeon Park, Seojin Kim, Changyeon Kim, Jaehyun Nam, and anonymous reviewers for their helpful feedbacks and discussions. We also thank Max Ehrlich for providing the exact results of prior video compression methods.

This work was mainly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub; No.2019-0-00075, Artificial Intelligence Graduate School Program (KAIST); No.2019-0-01906, Artificial Intelligence Graduate School Program (POSTECH)). This research was partly supported by the Challengeable Future Defense Technology Research and Development Program (No.915027201) of Agency for Defense Development in 2022. This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2022R1F1A1075067).

References

[1] E. Agustsson, D. Minnen, N. Johnston, J. Balle, S. J. Hwang, and G. Toderici. Scale-space flow for end-to-end optimized video compression. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2020.

[2] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild. FLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(2):15:1–15:23, 2020.

[3] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *IEEE International Conference on Computer Vision*, pages 5855–5864, 2021.

[4] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe. Deep local shapes: Learning local SDF priors for detailed 3d reconstruction. In *ECCV*, pages 608–625. Springer, 2020.

[5] H. Chen, B. He, H. Wang, Y. Ren, S. N. Lim, and A. Shrivastava. NeRV: Neural representations for videos. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[6] Y. Chen, S. Liu, and X. Wang. Learning continuous image representation with local implicit image function. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[7] Z. Chen, Y. Chen, J. Liu, X. Xu, V. Goel, Z. Wang, H. Shi, and X. Wang. VideoINR: Learning video implicit neural representation for continuous space-time super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[8] J. Chibane, T. Alldieck, and G. Pons-Moll. Implicit functions in feature space for 3D shape reconstruction and completion. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6970–6981, 2020.

[9] S.-F. Chng, S. Ramasinghe, J. Sherrah, and S. Lucey. GARF: Gaussian activated radiance fields for high fidelity reconstruction and pose estimation. *arXiv e-prints*, pages arXiv–2204, 2022.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] E. Dupont, A. Golinski, M. Alizadeh, Y. W. Teh, and A. Doucet. COIN: COmpression with Implicit Neural representations. In *Neural Compression: From Information Theory to Applications – Workshop @ ICLR 2021*, 2021.

[12] E. Dupont, H. Loya, M. Alizadeh, A. Goliński, Y. W. Teh, and A. Doucet. COIN++: Data agnostic neural compression, 2022.

[13] R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021.

[14] J. Gu, L. Liu, P. Wang, and C. Theobalt. StyleNeRF: A style-based 3d aware generator for high-resolution image synthesis. In *International Conference on Learning Representations*, 2022.

[15] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec. Baking neural radiance fields for real-time view synthesis. In *IEEE International Conference on Computer Vision*, pages 5875–5884, 2021.

[16] A. Hore and D. Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

[17] Z. Huang, S. Bai, and J. Z. Kolter. Implicit^2: Implicit layers for implicit representations. In *Advances in Neural Information Processing Systems*, 2021.

[18] A. Jain, M. Tancik, and P. Abbeel. Putting NeRF on a diet: Semantically consistent few-shot view synthesis. In *IEEE International Conference on Computer Vision*, 2021.

[19] C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, T. Funkhouser, et al. Local implicit grid representations for 3D scenes. In *CVPR*, pages 6001–6010, 2020.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[21] J. Lee, J. Tack, N. Lee, and J. Shin. Meta-learning sparse implicit neural representations. In *Advances in Neural Information Processing Systems*, 2021.

[22] T. Li, M. Slavcheva, M. Zollhoefer, S. Green, C. Lassner, C. Kim, T. Schmidt, S. Lovegrove, M. Goesele, and Z. Lv. Neural 3D video synthesis from multi-view video. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[23] Z. Li, S. Niklaus, N. Snavely, and O. Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[24] C. H. Lin, H.-Y. Lee, Y.-C. Cheng, S. Tulyakov, and M.-H. Yang. InfinityGAN: Towards infinite-pixel image synthesis. In *International Conference on Learning Representations*, 2022.

[25] J. Liu, S. Wang, W.-C. Ma, M. Shah, R. Hu, P. Dhawan, and R. Urtasun. Conditional entropy coding for efficient video compression. In *European Conference on Computer Vision*, pages 453–468. Springer, 2020.

[26] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. In *Advances in Neural Information Processing Systems*, volume 33, pages 15651–15663, 2020.

[27] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[28] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao. Dvc: An end-to-end deep video compression framework. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[29] J. N. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein. ACORN: Adaptive coordinate networks for neural representation. In *ACM Trans. Graph. (SIGGRAPH)*, 2021.

[30] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth. NeRF in the wild: Neural radiance fields for unconstrained photo collections. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[31] I. Mehta, M. Gharbi, C. Barnes, E. Shechtman, R. Ramamoorthi, and M. Chandraker. Modulated periodic activations for generalizable local functional representations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 14214–14223, 2021.

[32] A. Mercat, M. Viitanen, and J. Vanne. UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, page 297–302, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368452. doi: 10.1145/3339825.3394937.

[33] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, 2020.

[34] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127.

[35] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. Deformable neural radiance fields. In *IEEE International Conference on Computer Vision*, 2021.

[36] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[37] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger. Convolutional occupancy networks. In *ECCV*, pages 523–540. Springer, 2020.

[38] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung, and L. Van Gool. The 2017 DAVIS challenge on video object segmentation. *arXiv:1704.00675*, 2017.

[39] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[40] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems*, 2020.

[41] I. Skorokhodov, S. Ignatyev, and M. Elhoseiny. Adversarial generation of continuous images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[42] I. Skorokhodov, S. Tulyakov, and M. Elhoseiny. StyleGAN-V: A continuous video generator with the price, image quality and perks of StyleGAN2. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[43] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012. doi: 10.1109/TCSVT.2012.2221191.

[44] C. Sun, M. Sun, and H.-T. Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[45] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[46] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, 2020.

[47] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. *arXiv preprint arXiv:2202.05263*, 2022.

[48] S. Tomar. Converting video formats with ffmpeg. *Linux journal*, 2006(146):10, 2006.

[49] G. K. Wallace. The JPEG still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[50] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[51] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003. doi: 10.1109/TCSVT.2003.815165.

[52] W. Xian, J.-B. Huang, J. Kopf, and C. Kim. Space-time neural irradiance fields for free-viewpoint video. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[53] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[54] D. Xu, Y. Jiang, P. Wang, Z. Fan, H. Shi, and Z. Wang. SinNeRF: Training neural radiance fields on complex scenes from a single image. *arXiv preprint arXiv:2204.00928*, 2022.

[55] R. Yang, F. Mentzer, L. V. Gool, and R. Timofte. Learning for video compression with hierarchical quality and recurrent enhancement. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 6628–6637, 2020.

[56] R. Yang, Y. Yang, J. Marino, and S. Mandt. Hierarchical autoregressive modeling for neural video compression. *arXiv preprint arXiv:2010.10258*, 2020.

[57] S. Yu, J. Tack, S. Mo, H. Kim, J. Kim, J.-W. Ha, and J. Shin. Generating videos with dynamics-aware implicit generative adversarial networks. In *International Conference on Learning Representations*, 2022.

[58] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[59] Y. Zhang, T. van Rozendaal, J. Brehmer, M. Nagel, and T. Cohen. Implicit neural video compression. In *ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022.

[60] P. Zhou, L. Xie, B. Ni, and Q. Tian. CIPS-3D: A 3D-Aware Generator of GANs Based on Conditionally-Independent Pixel Synthesis. arXiv preprint 2110.09788, 2021.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section 4
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 4 and supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.
   (b) Did you mention the license of the assets? [Yes] See Section 4.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See the supplementary material.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix: Scalable Neural Video Representations with Learnable Positional Features

Website: https://subin-kim-cv.github.io/NVP

A More description of experimental setups

A.1 Metrics

- **PSNR**. To measure the reconstruction quality of models, we use peak signal-to-noise ratio (PSNR), evaluated as $-\log_{10}(\texttt{MSE})$, where MSE denotes the mean-squared error between the ground-truth video and the reconstructed video (as 0-1 scale).
- **LPIPS**. We use AlexNet [20] pretrained on ImageNet [10], which best performs as a forward metric.[4] It measures a weighted distance between normalized internal features of the real video and its reconstruction.
- **FLIP**. FLIP automates the difference evaluation between alternating images by building on principles of human perception, where we use the official implementation.[5]

A.2 Further implementation details

Training details. We train the network by adopting mean-squared error as our loss function and using the AdamW optimizer [27] with a learning rate of 0.01. We utilize HEVC [43] for the compression of our sparse positional features and JPEG [49] for reducing the parameters of keyframes.

Architecture. For the learnable keyframes, which follow a multi-level structure, we set the number of levels as 16, per level scale γ as 1.35, and the coarsest resolution as 16×16. To build models with different sizes, we set the number of features per level as 2 and 4 for NVP-S and NVP-L, respectively. For the *sparse* positional features, considering the number of video frames, we use a 3D grid size of $300 \times 300 \times 300$ for ShakeNDry and $300 \times 300 \times 600$ for other videos in UVG-HD, which is $\sim$23$\times$ smaller than the video pixels. We then concatenate $3 \times 3 \times 1$ latent codes in sparse positional features. Same as the learnable keyframes, we set the latent dimensions of sparse positional features to be 2 and 4 for NVP-S and NVP-L, respectively. In addition, we design modulated implicit function as a 3-layer multi-layer perceptron (MLP) modulated by another modulator network; both have a hidden size of 128. For the synthesizer network, we use SIREN [40], which uses $\sin(\sigma_t \mathbf{z})$ with $\sigma_t \in \mathbb{R}$ as the activation functions, and set the temporal frequency of the first layer as $\sigma_t = 30$ and the other layers as $\sigma_t = 1$. This network is modulated by a modulator network with LeakyReLU [53] activation.

We train the network with a batch size of 1,245,184, *i.e.*, at each iteration, we randomly sample 1,245,184 pixels from entire video pixels. We initialize the parameters of learnable keyframes (and sparse positional features) from the uniform distribution $U(-10^{-4}, 10^{-4})$ and use Kaiming normal initialization for the modulator networks. We train the network using AdamW optimizer [27] with the initial learning rate $\eta = 0.01$ and weight decay $\lambda = 0.001$. We use a cosine annealing learning rate scheduler, where the current learning rate η_t at the t-th iteration is defined as follows:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta - \eta_{\min})\Big(1 + \cos\Big(\frac{t}{T}\pi\Big)\Big),$$

where $\eta_{\min}$ is set to 0.00001, and the total iteration T is set to 100,000 for both NVP-S and NVP-L.

Compression procedure. Following the prior work[5], we used *ffmpeg* [48] to extract RGB frames from compressed UVG-HD video files and to compress learnable positional features of NVP.

First, we use the following command to extract RGB videos from the original YUV videos of UVG-HD:

```
$ffmpeg -f rawvideo -vcodec rawvideo -s 3840x2160 -r 120 -pix_fmt yuv420p \
-i INPUT.yuv OUTPUT/f%05d.png
```

where INPUT is the input file name, and OUTPUT is a directory to save decompressed RGB frames.

[4] https://github.com/richzhang/PerceptualSimilarity
[5] https://github.com/NVlabs/flip

Then we use the following commands to compress learnable keyframes:

```
$ffmpeg -hide_banner -i input.png -qscale:v SCALE output.jpg
```

where SCALE is an output option that controls image quality.

We also use the following commands to compress sparse positional features:

```
$ffmpeg  -framerate FR -i INPUT/f%05d.png -c:v hevc -preset slow -x265-params \
bframes=0 -crf CRF OUTPUT.mp4
```

where FR and CRF indicate the frame rate and constant rate factor value that controls video quality and compression ratio, respectively. We summarize the hyperparameters used in the below table:

Table 5: Hyperparameter values used in the compression procedure.

Model	SCALE $\mathbf{U}_{\theta_{\mathtt{xy}}}$	SCALE $\mathbf{U}_{\theta_{\mathtt{xt}}}$	SCALE $\mathbf{U}_{\theta_{\mathtt{yt}}}$	FR	CRF
NVP-S	2	3	3	25	21
NVP-L	2	2	2	40	21

B Description of baseline methods

In this section, we briefly describe the specific parameter we used as baselines of video coordinate-based neural representations (CNRs) for evaluating our framework at a high level. To compare the parameter efficiency, we consider two situations: the average bits-per-pixel (BPP) of each baseline is either near 0.200 or 0.400. However, in the case of Instant-ngp [34], to compare the compute efficiency, we set the total number of parameters as similar as NVP-L when encoding time is ≤ 1 hour

- **SIREN** [40], which uses high frequency *sine* activations (*i.e.*, $\sin(\omega_0 \boldsymbol{z})$ with $\omega_0 \gg 1$) and takes spatio-temporal coordinate (x, y, t) as input then outputs a corresponding RGB value for each pixel. We use a 5-layer multi-layer perceptron (MLP) with a hidden size of 2,048, where the frequency ω_0 is set to 30 for all sinusoidal activations.
- **FFN** [46] leverages random fourier feature (RFF) (*i.e.*, $[\sin(2\pi\mathbf{W}\mathbf{z}), \cos(2\pi\mathbf{W}\mathbf{z})]$) as a positional embedding layer to encode spatio-temporal coordinates (x, y, t) and uses ReLU activation for further layers to output a corresponding RGB value for each pixel. We use RFF with $\mathbf{W} \in \mathbb{R}^{3\times 1024}$ with $W_{ij} \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 10$, and a 4-layer MLP with a hidden size of 2,048.
- **NeRV** [5], a CNR specialized for videos, takes a time index as input and outputs a corresponding RGB image. We use two configurations provided in the official implementation:[6] NeRV-S and NeRV-L for a small and a large model, respectively. Specifically, we first apply a 2-layer MLP on the output of the positional encoding layer, and then we stack 5 NeRV blocks with upscale factors 5, 3, 2, 2, 2, respectively. We set the output channel for the first fully-connected layer as 128 for both models and change the expansion at the beginning of convolutional block size for 4 and 8 for NeRV-S and NeRV-L, respectively. After training one model for a single video separately, we prune the trained parameters by removing 15% of the weights, quantize model weights to 8-bit, and apply entropy coding following the compression procedure in the paper.
- **Instant-ngp** [34] uses multiresolution hash tables of trainable feature vectors for a given input coordinate (x, y, t). To be specific, on the UVG-HD benchmark, we set the number of levels as 15, the number of features per level as 2, the maximum entries per level as 2^{24}, and the coarsest resolution as 16. We set per level scale γ as 1.3 for ShakeNDry, and 1.5 for other videos in the UVG-HD. We use a small neural network with 64 neurons and two hidden layers that use ReLU as output activation for a small latent-to-RGB mapping.

[6] https://github.com/haochen-rye/NeRV

C Video-wise main results

In this section, we provide video-wise quantitative results and qualitative illustrations.

Table 6: Quantitative results of NVP-S on 7 videos in UVG-HD: Beauty, Bosphorus, HoneyBee, Jockey, ReadySetGo, ShakeNDry, and Yachtride.

Encoding time	Video name	BPP	PSNR (↑)	FLIP (↓)	LPIPS (↓)
∼5 minutes	Beauty	0.875	33.80	0.061	0.399
	Bosphorus	0.875	35.45	0.077	0.117
	HoneyBee	0.875	37.89	0.048	0.127
	Jockey	0.875	35.51	0.082	0.235
	ReadySetGo	0.875	30.74	0.109	0.161
	ShakeNDry	1.056	36.84	0.056	0.147
	Yachtride	0.875	31.73	0.090	0.146
∼10 minutes	Beauty	0.875	34.52	0.055	0.362
	Bosphorus	0.875	37.52	0.060	0.085
	HoneyBee	0.875	38.51	0.044	0.124
	Jockey	0.875	37.14	0.065	0.216
	ReadySetGo	0.875	32.37	0.094	0.112
	ShakeNDry	1.056	36.92	0.067	0.128
	Yachtride	0.875	33.53	0.074	0.096
∼1 hour	Beauty	0.875	35.42	0.048	0.361
	Bosphorus	0.875	40.06	0.048	0.065
	HoneyBee	0.875	39.51	0.039	0.121
	Jockey	0.875	38.91	0.050	0.201
	ReadySetGo	0.875	34.68	0.074	0.080
	ShakeNDry	1.056	38.81	0.045	0.118
	Yachtride	0.875	35.88	0.060	0.068
∼8 hours	Beauty	0.277	34.82	0.054	0.286
	Bosphorus	0.172	39.16	0.058	0.069
	HoneyBee	0.192	38.38	0.049	0.095
	Jockey	0.172	37.03	0.073	0.220
	ReadySetGo	0.181	33.49	0.094	0.090
	ShakeNDry	0.297	37.85	0.055	0.104
	Yachtride	0.180	34.49	0.083	0.084
∼11 hours	Beauty	0.523	35.43	0.051	0.170
	Bosphorus	0.333	40.21	0.056	0.049
	HoneyBee	0.409	39.13	0.044	0.064
	Jockey	0.323	37.86	0.067	0.205
	ReadySetGo	0.351	34.49	0.091	0.073
	ShakeNDry	0.585	38.70	0.052	0.088
	Yachtride	0.359	36.44	0.075	0.066

Figure 12: Reconstruction results on Beauty in UVG-HD after training NVP-S for “5 minutes”, “10 minutes”, and “1 hour” with a single NVIDIA V100 32GB GPU.

Figure 13: Reconstruction results on Bosphorus in UVG-HD after training NVP-S for “5 minutes”, “10 minutes”, and “1 hour” with a single NVIDIA V100 32GB GPU.

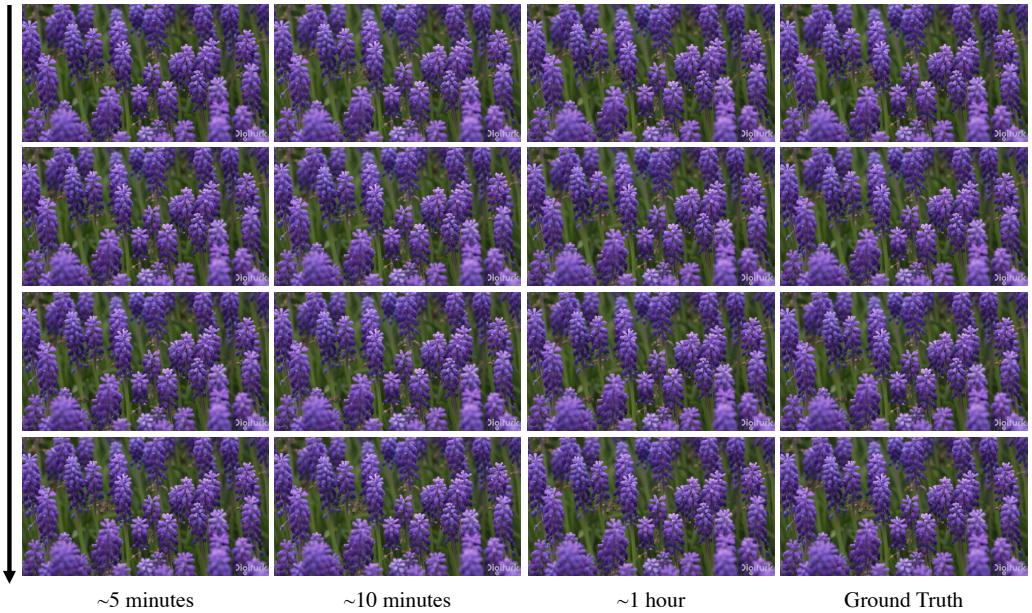


Figure 14: Reconstruction results on Honeybee in UVG-HD after training NVP-S for “5 minutes”, “10 minutes”, and “1 hour” with a single NVIDIA V100 32GB GPU.

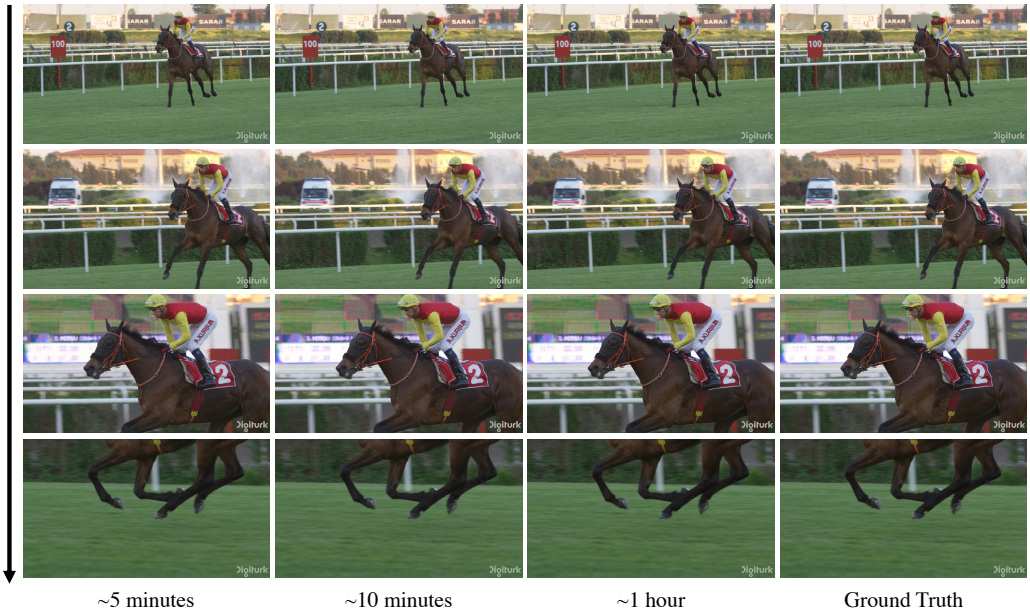


Figure 15: Reconstruction results on Jockey in UVG-HD after training NVP-S for “5 minutes”, “10 minutes”, and “1 hour” with a single NVIDIA V100 32GB GPU.

~5 minutes ~10 minutes ~1 hour Ground Truth

Figure 16: Reconstruction results on ReadySetGo in UVG-HD after training NVP-S for “5 minutes”, “10 minutes”, and “1 hour” with a single NVIDIA V100 32GB GPU.

~5 minutes ~10 minutes ~1 hour Ground Truth

Figure 17: Reconstruction results on ShakeNDry in UVG-HD after training NVP-S for “5 minutes”, “10 minutes”, and “1 hour” with a single NVIDIA V100 32GB GPU.

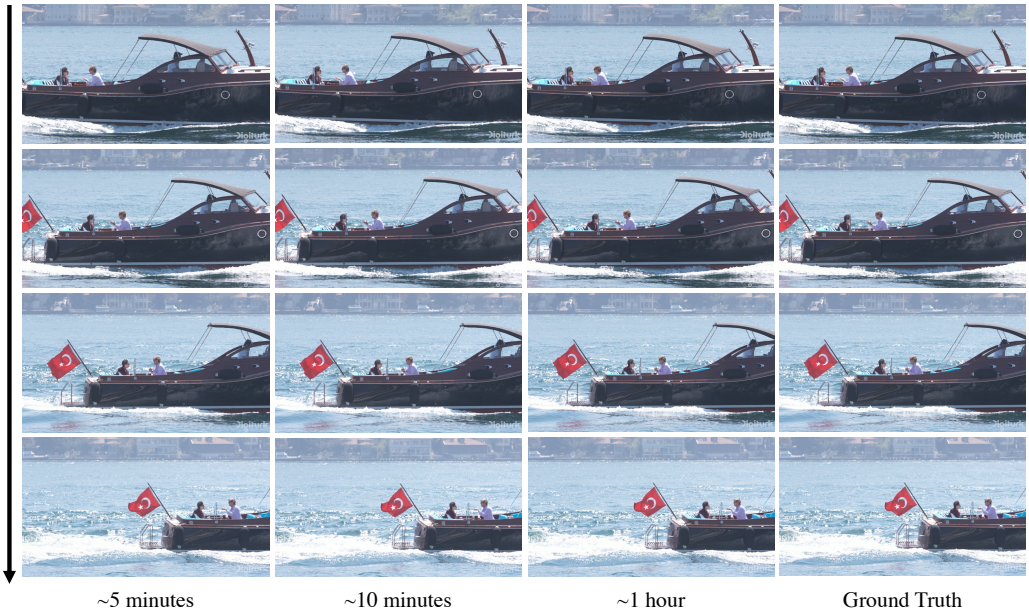


Figure 18: Reconstruction results on Yachtride in UVG-HD after training NVP-S for "5 minutes", "10 minutes", and "1 hour" with a single NVIDIA V100 32GB GPU.

D Comparison of video decoding time

We compare the decoding time of NVP with other baselines on a single GPU (NVIDIA V100 32GB).

Table 7: Decoding time of coordinate-based representations measured with FPS (higher is better). It was measured on Jockey (600 frames with 1920 $\times$ 1080 resolution) in UVG-HD with a single NVIDIA V100 32GB GPU.

Model	BPP	FPS
Instant-ngp [34]	7.352	29.81
NeRV-S [5]	0.177	45.39
NeRV-L [5]	0.426	15.28
NVP-S	0.172	6.51
NVP-L	0.359	4.87

NVP requires more decoding time than prior works, mainly due to (a) *three* learnable keyframes along each spatio-temporal axis (three times more access than the single grids of Instant-ngp [34]) and (b) the modulation architecture while evaluating the corresponding RGB values. However, note that the computation through each keyframe does not depend on each other; in this respect, the decoding time of NVP can be sped up significantly with a parallel design and implementation, *e.g.*, following the implementation details from Instant-ngp.[7] Moreover, while (b) exhibits considerable improvements in encoding complicated videos, we also demonstrate that our method still shows reasonable video encoding without modulations (both are shown in Section 4.3 of the main text); hence, one can control the trade-off between the decoding time and encoding quality by designating the multilayer perceptron (MLP) size and the whether the modulation is applied (or not).

[7]Unlike our method that utilizes the well-known Pytorch [36] framework, Instant-ngp utilizes C++/CUDA to implement *all* of the components for enabling strong parallelism in training and inference. As the official implementation of Instant-ngp[8] states that the decoding time highly deviates if one uses non-C++/CUDA frameworks, *e.g.*, Pytorch, we argue the gap of decoding time in Table 7 mainly stems from such different implementation details, and it can be remarkably mitigated by following their implementation.

[8]https://github.com/NVlabs/instant-ngp

E More description and visualization of latent keyframes

Our keyframes aim to learn meaningful contents that are shared across a given video along an axis. For instance, the temporal axis ($\mathbf{U}_{\theta_{\mathtt{xy}}}$) learns common contents in every timeframe, such as background and the watermark in the video, which is invariant to timesteps.

The meanings of the other two keyframes ($\mathbf{U}_{\theta_{\mathtt{xt}}}$ and $\mathbf{U}_{\theta_{\mathtt{xt}}}$) may not be straightforward from their visualizations, as the shared contents across the other two directions in raw RGB space is often ambiguous. However, we note that we are learning the common contents in "latent space"; although they do not seem straightforward, they indeed play a crucial role in promoting the succinct parametrization of a given video (see Figure 8).

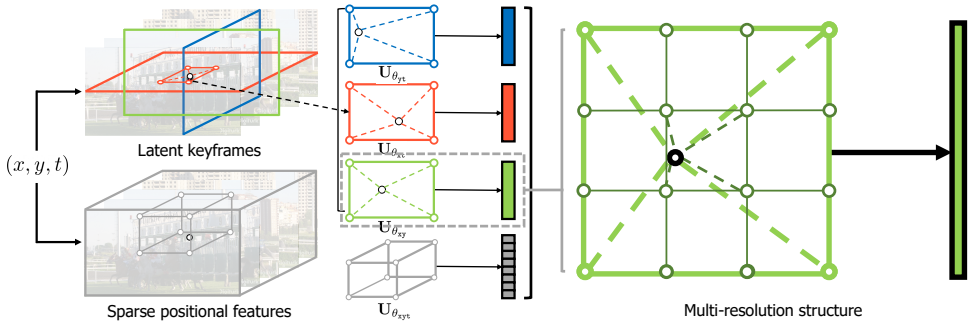
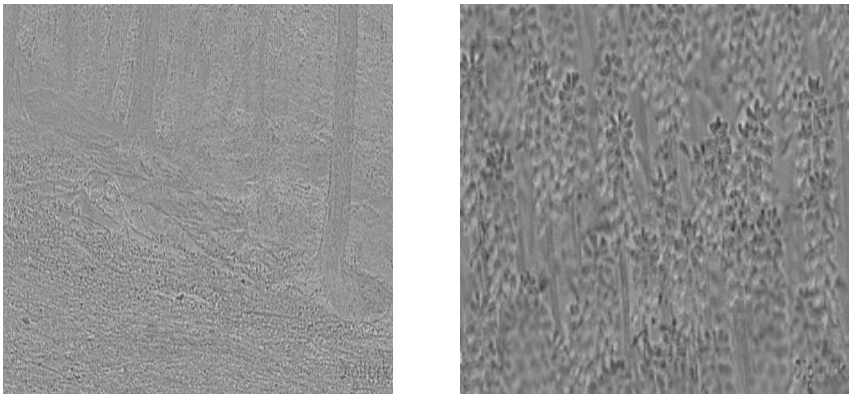


Figure 19: Illustration of our latent keyframe structure.



(a) ShakeNDry (b) Honeybee

Figure 20: Illustration of our learnable latent keyframes $\mathbf{U}_{\theta_{\mathtt{xy}}}$ after encoding (a) ShakeNDry, and (b) Honeybee in UVG-HD, respectively.

F Additional experiments on different datasets

F.1 Experiment on Big Buck Bunny

Following the experimental setup in NeRV [5], we provide additional experimental results in the Big Buck Bunny video for a more intuitive comparison to prior work.

F.2 Experiment with new videos

We also provide additional experimental results on more complex videos. In particular, we consider the following three new videos, where all of these videos are collected under the CC0 license:

Table 8: PSNR values and encoding time of different CNRs to encode the Big Buck Bunny video. ↑ and ↓ denote higher and lower values are better, respectively.

Method	BPP	PSNR (↑)	Encoding time (hr, ↓)
NeRV-S [5]	0.128	32.36	1.734
NVP-S (ours)	0.136	32.56	0.925
NeRV-M [5]	0.249	36.50	1.762
NVP-M (ours)	0.248	36.49	0.925
NeRV-L [5]	0.496	39.26	1.774
NVP-L (ours)	0.456	39.88	0.925

- (Street) A timelapse video of a London street. People, cars, and buses are moving at different speeds as the traffic signal changes.[9]
- (City) A timelapse video of the city at night. A lot of cars are moving speedily.[10]
- (Surfing) A video of a man surfing in the ocean. Huge ocean waves are changing dramatically and fast.[11]

Table 9: PSNR values to encode more temporally complex videos.

	UVG-HD (avg.)	Street	City	Surfing
BPP	0.412	0.214	0.173	0.311
PSNR	37.71	38.90	38.45	43.71

[9] https://pixabay.com/videos/id-28693/
[10] https://pixabay.com/videos/id-19627/
[11] https://pixabay.com/videos/id-110734/

G Video-wise compression result

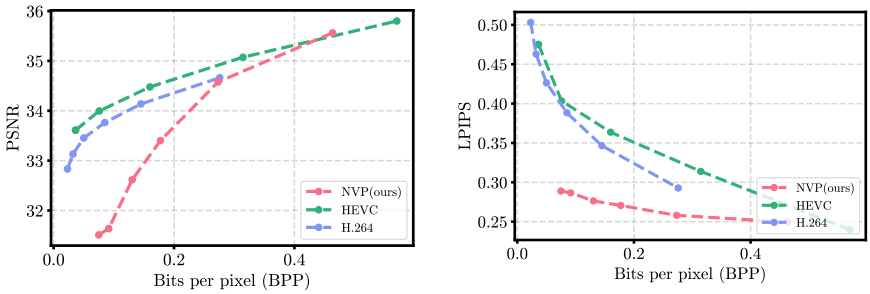


Figure 21: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the Beauty video in UVG-HD.

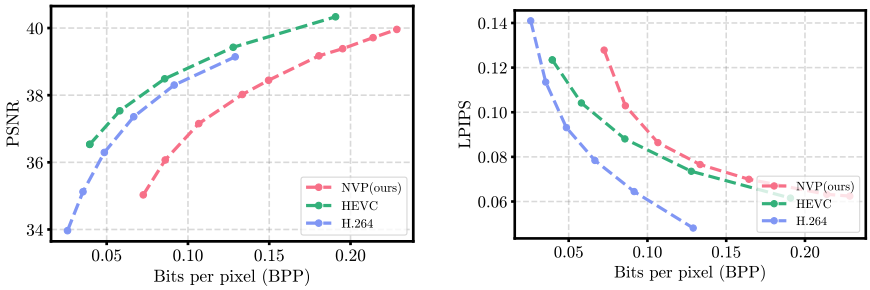


Figure 22: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the Bosphorus video in UVG-HD.

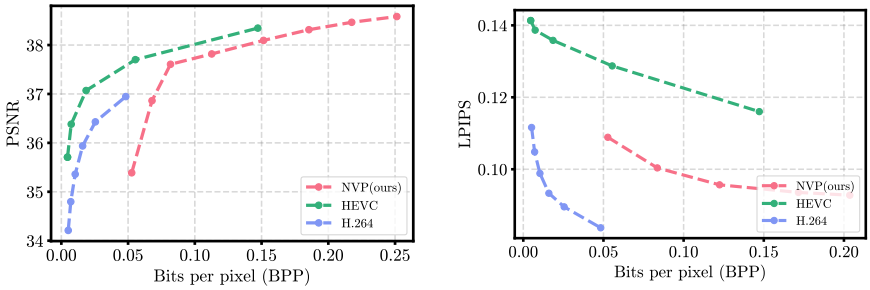


Figure 23: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the Honeybee video in UVG-HD.

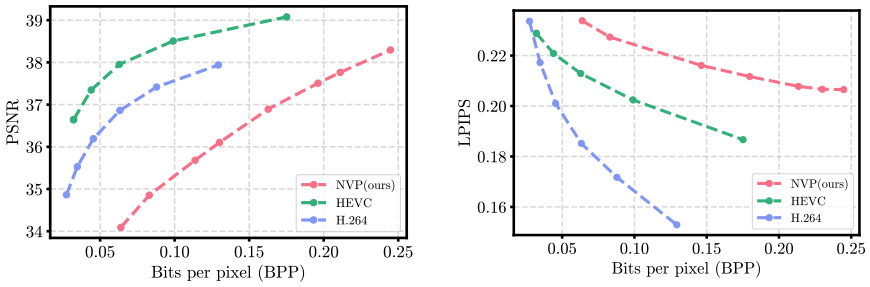


Figure 24: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the Jockey video in UVG-HD.

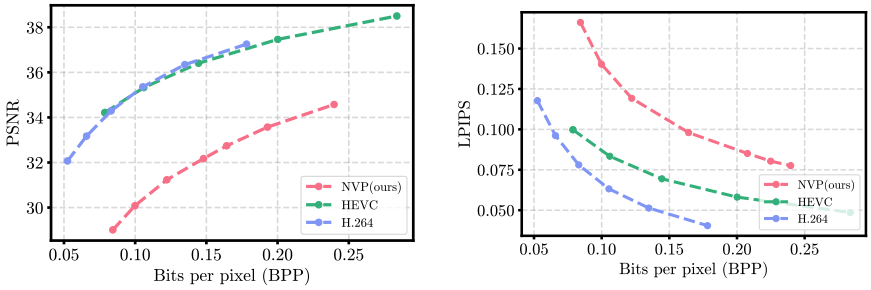


Figure 25: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the ReadySetGo video in UVG-HD.

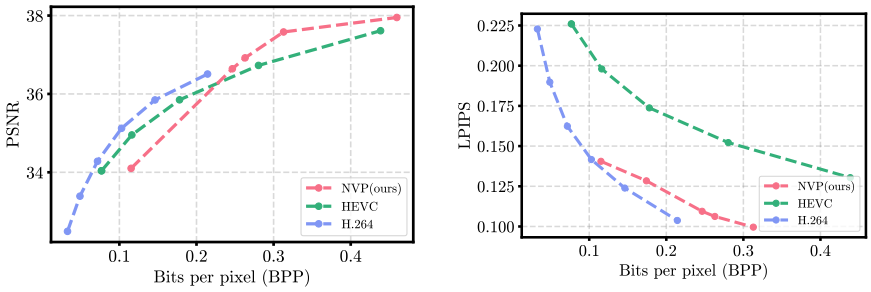


Figure 26: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the ShakeNDry video in UVG-HD.

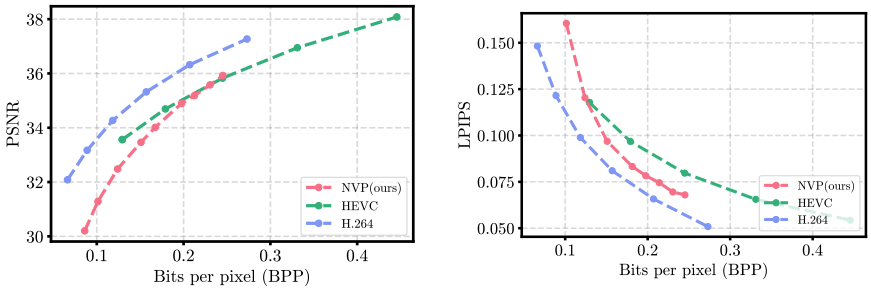


Figure 27: PSNR and LPIPS values of NVP and well-known video codecs over different BPP values computed on the Yachtride video in UVG-HD.

H Comparison to learning-based video compression

In Figure 28, we compare compressed videos from NVP with well-known video codecs (H.264 [51], HEVC [43]) and state-of-the-art learning-based video compression methods (DVC [28], STAT-SSF-SP [56], HLVC [55], Scale-space [1], Liu et al. [25], and NeRV [5]) on UVG-HD. We train NVP for every single video separately, where we use the same architecture and hyperparameters for encoding and compressing all videos in UVG-HD.

Since our primary focus is not on video compression, there exists a gap between the existing compression methods and NVP. However, we strongly believe that there are numerous future directions to engage NVP for video compression, such as investigating per-video hyperparameter search strategy or designing better CNR architecture more specialized for compression.

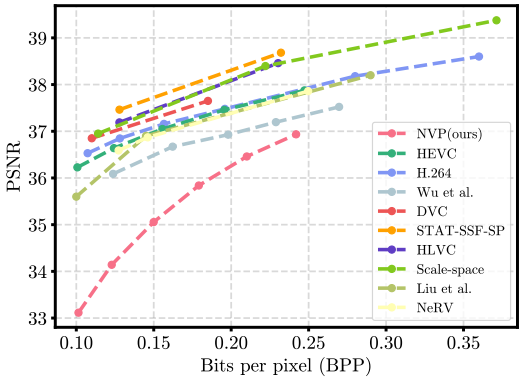


Figure 28: PSNR values of NVP, well-known video codecs, and learning-based video compression methods over different BPP values computed on UVG-HD.