How to Design and Train Your Implicit Neural Representation for Video Compression

Matthew Gwilliam Roy Zhang Namitha Padmanabhan Hongyang Du Abhinav Shrivastava
University of Maryland, College Park

Abstract

Implicit neural representation (INR) methods for video compression have recently achieved visual quality and compression ratios that are competitive with traditional pipelines. However, due to the need for per-sample network training, the encoding speeds of these methods are too slow for practical adoption. We develop a library to allow us to disentangle and review the components of methods from the NeRV family, reframing their performance in terms of not only size-quality trade-offs, but also impacts on training time. We uncover principles for effective video INR design and propose a state-of-the-art configuration of these components, Rabbit NeRV (RNeRV). When all methods are given equal training time (equivalent to 300 NeRV epochs) for 7 different UVG videos at 1080p, RNeRV achieves +1.27% PSNR on average compared to the best-performing alternative for each video in our NeRV library. We then tackle the encoding speed issue head-on by investigating the viability of hyper-networks, which predict INR weights from video inputs, to disentangle training from encoding to allow for real-time encoding. We propose masking the weights of the predicted INR during training to allow for variable, higher quality compression, resulting in 1.7% improvements to both PSNR and MS-SSIM at 0.037 bpp on the UCF-101 dataset, and we increase hyper-network parameters by 0.4% for 2.5%/2.7% improvements to PSNR/MS-SSIM with equal bpp and similar speeds. Our code is available at https://github.com/mgwillia/vinrb.

1. Introduction

Implicit neural representations (INRs) are very appealing for video compression due to their fast decoding speeds. Many neural network researchers have tried to apply deep learning to the video compression problem in attempts to (1) improve video quality while (2) reducing the storage size [1, 39]. One popular family of approaches, Neural Video Compression (NVC) [26], offers promising performance in both those areas. However, these methods suffer from slow decoding speeds, which limit adoption for practical use. By contrast, INRs boast real-time decoding speeds at low storage size, and recent developments have improved the reconstruction quality substantially, to the point of being more competitive with NVC and traditional codecs [17, 21]. However, since INR-based approaches require training a neural network for each sample individually as part of the encoding, encoding speeds are incredibly slow.

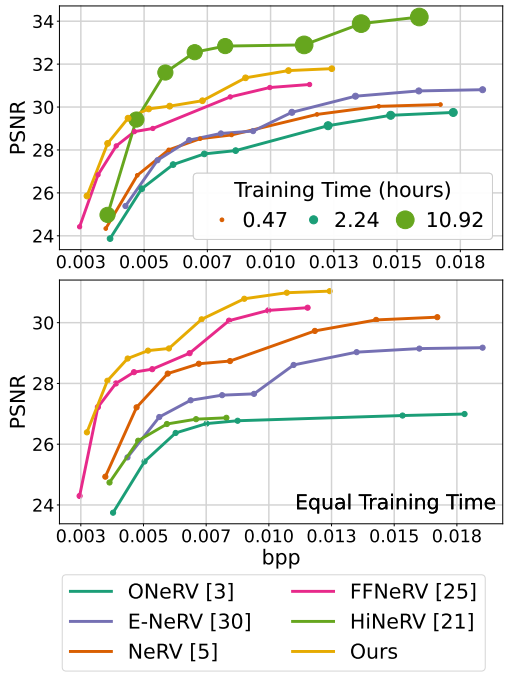

Figure 1. **Compression performance of INR-based video codecs** from the NeRV family. We examine not only size (bits per pixel) and quality (peak singal-noise ratio), but also encoding speed. Since INRs must be trained for each sample, the encoding speed is dominated by the training time. **(top)** PSNR/bpp with time as dot size. **(bottom)** PSNR/bpp for equal time (30 minutes on RTXA5000), averaged over 7 UVG videos at 1080p.

We consider size, quality, encoding time, and decoding speed as 4 equally important criteria for judging the effectiveness of video compression methods. So, we propose to tackle video INR's greatest weakness, encoding speed (training time), head-on. We find that in general the en-

coding time is both under-studied and under-valued. While existing research will typically make comparisons fair in terms of size by measuring both the number of parameters and the bits-per-pixel (bpp), for encoding speed they will typically report results with equal epochs [5, 21]. This neglects the actual real cost of training these methods, and encourages the introduction of parameters in configurations that results in large amounts of FLOPs and high wall time.

We illustrate the impact of such a paradigm in Figure 1. With equal training iterations, HiNeRV [21] dominates compared to its predecessors. However, with equal training time, such by allowing all methods the amount of time it takes NeRV to run 300 epochs as measured on a single NVIDIA RTXA5000 GPU (30 minutes), the landscape changes drastically. FFNeRV [25] emerges as the most effective method on average in this setting. By contrast, HiNeRV is more similar to the performance of the Original NeRV [3] (ONeRV), while the updated NeRV proposed in the HNeRV paper [5] (which has a simpler stem) is significantly better than both. However, as we show in Section 4, HiNeRV is still the best for longer encoding times.

With encoding speed concerns in mind, we disentangle NeRV-like methods to reveal principles for optimizing the desired combination of training time, storage size, and reconstruction quality. We use these principles to formulate our Rabbit NeRV (RNeRV), which we name for its faster training times (and rabbits are associated with speed in many cultures). We also analyze the differences between these methods through a qualitative lens by extending the XINC [38] method for the rest of the NeRV family.

To further address the encoding time issue, we turn towards hyper-networks [7]. By taking a video as input and predicting the INR weights directly, one can skip the per-sample fitting process altogether, and instead train the hyper-network before encoding time. We propose to use hyper-networks to compress entire videos, 8 frames at a time. We refer to this architecture, based on NeRV-Enc and NeRV-Dec [6] as "HyperNeRV." We devise a training strategy, Weight Token Masking, where we randomly mask parts of the hyper-network's weight predictions during training. This enables us to mask the same parts at inference time, with minimal decrease in quality. Since we do not need to store the masked parts, we are able to perform compression with a flexible number of bits that can be determined at encoding time (by choosing to mask, or not to mask). We also show how we can modify size of the "shared parameters" of the hyper-network to achieve better video quality at equal compressed size.

In summary, we propose that in light of the promising improvements for video INR methods in terms of rate-distortion, the community must focus more on encoding time. We make the following key contributions:

- We develop a library that disentangles various state-of-the-art video INRs, and use this codebase to distill and elucidate best principles for NeRV design.
- We provide a concrete configuration, RNeRV, that achieves optimal performance for different encoding (training) time budgets – 1.27% PSNR and 0.72% MS-SSIM improvements on average compared to the next-best method for 30 minutes training time on UVG.
- We propose weight token masking to enable the model to encode at 2 different storage sizes, with 1.7% improvement to PSNR and MS-SSIM on UCF-101.

2. Related Work

Video Compression. Since the rise of deep learning, many approaches have tried to supplant or enhance established video compression methods [24, 48, 51]. Some focus on improving components of standard compression pipelines using deep learning [1, 15, 31, 39, 40]. DCVC advocates for conditional coding instead of traditional predictive coding [26], and many works in the family of "Neural Video Compression" (NVC) follow this paradigm [27–29]. In our work, instead of traditional pipelines or NVC, we focus on implicit neural representation for video compression.

Implicit Neural Representation. With implicit neural representation (INR), a neural network is trained to map a set of coordinates to some signal, such as an image, video, or 3D representation [3, 36, 37, 42, 44, 49, 53]. An example INR could learn a multilayer perceptron to map (x, y) positions (often with some positional encoding) to (r, g, b) tuples, such that an image can be learned and stored as neural network weights. These networks can be designed to be much smaller than the images they represent, such that one major application of INR has been image compression [8, 23, 47]. Other works focus more directly on video compression [3, 9, 19, 22, 32, 58], and some are designed with both in mind [17]. In this work, we focus on video compression, and even more specifically we study the popular NeRV family of models [18, 21, 25, 30, 55, 57], pioneered by Chen et al. [3]. It uses convolution layers, typically upsampling from small feature maps via PixelShuffle [43], in addition to MLPs and outputs all RGB values of a frame given the positional embedding of frame index t as input. More recent methods have abandoned the fixed positional encoding; some, such as HNeRV [5], use an image encoder to learn tiny content-based embeddings [4, 12, 41, 52, 54, 59, 60]. Others, such as FFNeRV [25] and HiNeRV [21], learn grid features (and such approaches are popular even outside the NeRV family [10, 19, 32]).

Hyper-Networks. Some approaches leverage meta-learning, training a single hyper-network to avoid long per-sample training times associated with INR. Given an input image or video, the hyper-network predicts INR network weights ("hypo-network") that can reconstruct the input [7, 16]. Some works predict INRs to generate novel

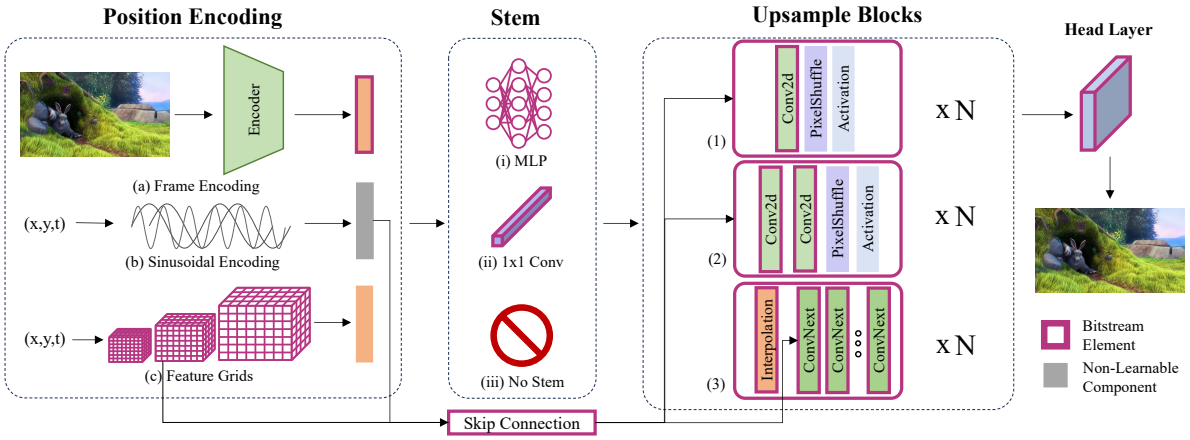

Figure 2. **Disentangling components within the NeRV family**. We isolate the various components of NeRV-like architectures and categorize critical parts of the design space – positional encoding, stem, and upsample blocks. There are other components shown in the figure (skip connections) as well as many not shown, including critical choices like parameter distribution. In this work we analyze the impact of these various components and propose configurations based on different target optimizations of the time-quality-size trade-off.

images [11, 45] or videos [56]. Latent-INR proposes per-sample hyper-networks for video compression, and even aligns these representations with CLIP embeddings for semantic-aware INR [33]. NeRV-Enc/NeRV-Dec [6] is the first work to apply non-implicit hyper-networks for video compression, sacrificing image quality and bitstream length for much faster encoding times. While NeRV-Enc shows promising results for compressing a video as a collection of 8 evenly sampled frames, we are the first work to use hyper-networks to compress entire videos. We also propose novel Weight Token Masking and slightly increasing hyper-network size for better overall performance.

3. Method

We first explain how we disentangle the components of NeRV models in Section 3.1. In Section 3.1.1 we describe the individual contributions of popular NeRV-like methods, and in Section 3.1.2 we explain how we decompose these methods for further study and improvement. As proof that such an investigation is useful, we use it to formulate our proposed RNeRV. We describe our novel Weight Token Masking for improving the size-quality performance of hyper-network compression strategies in Section 3.2.

3.1. Disentangling Video INRs

3.1.1. Existing Methods

In this work, we primarily focus on the NeRV family of INR models, pioneered by the NeRV paper itself [3]. The original NeRV trains a neural network, θ to predict a frame at some timestep, I_t, that is $I_t = \theta(t)$, using a reconstruction loss to minimize the difference between I_t and $\theta(t)$. NeRV starts by converting the frame index, t, to a sinusoidal positional encoding. An MLP expands the dimension of the encoding, which is then reshaped to a matrix of shape $fc_w \times fc_h \times fc_{\text{dim}}$. From there, NeRV gradually upsamples, with each layer first using a convolutional layer to expand the channel dimension (which is fc_{dim} for the first layer), followed by a PixelShuffle to convert the additional channel dimensions to larger spatial size, then an activation. Once the output has reached the size of the original frame, NeRV applies a final convolutional layer (called the "head layer") to convert the channel dimension to color channels.

E-NeRV [30] expands the stem of NeRV to disentangle and process an xy embedding in addition to t. It also adds learnable skip connections from t to every convolutional layer, and proposes a reworked convolutional layer for the first NeRV block (after the stem). HNeRV [5] principally proposes replacing the fixed, non-learnable positional encoding with a tiny, trainable ConvNext encoding which outputs content-adaptive embeddings. It also proposes improvements for non-hybrid NeRV, by changing the MLP stem of NeRV to a simpler, larger stem consisting of a single fully-connected layer. HNeRV highlights importance of proper parameter distribution and proposes adjusting kernel sizes and channel dimensions to ensure the later layers in the network have a good number of parameters. Instead of a hybrid encoder, FFNeRV [25] proposes a learnable grid for the positional encoding, eliminates the stem and shifts its responsibilities to the first NeRV block, and proposes a NeRV block that starts with an additional group-wise convolution. It also proposes a flow-warping strategy for modeling motion in videos. HiNeRV [21] introduces a broad array of improvements – grid-based stem, grid-based skip connection, patchwise learning and decoding, and bilinear interpolation for upsampling with ConvNext to process fea-

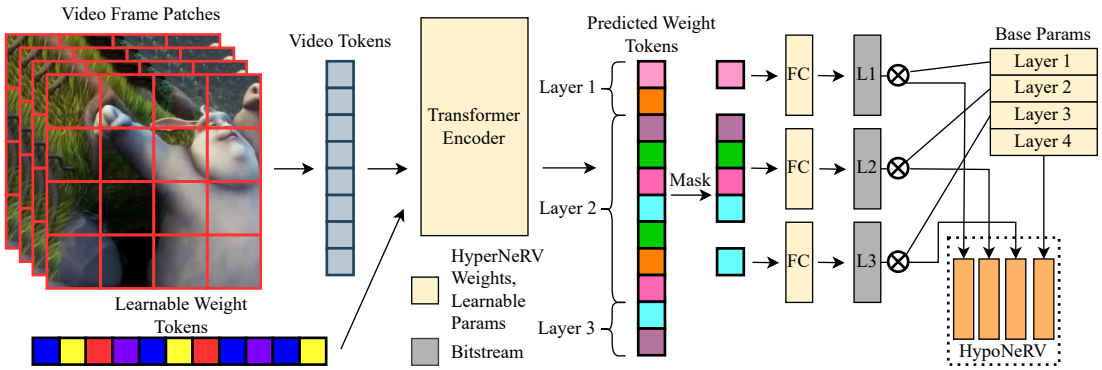

Figure 3. **Weight Token Masking.** We make hyper-networks have flexible encoding size at inference time. If we mask some of the predicted weight tokens for some portion of the samples (in our case, 50%), we can also choose to mask the same set of tokens at inference for any given input clip (group of frames), with a small decrease in reconstruction quality. That is, we can flexibly reduce the bitrate by 2x post-training, during encoding, by choosing whether to mask some tokens for each group of frames.

tures in the NeRV blocks.

3.1.2. Our Definition of the Video INR Design Space

In Figure 2, we provide an illustration of this design space. Video INRs in the NeRV family are a combination of a position encoding, some stem to process that encoding, and then upsample blocks to transform that encoding to the resolution of the video. Some leverage skip connections to improve performance. Distribution of parameters, while not represented in the figure, is crucial for maintaining a healthy balance between speed and performance. Since our work focuses on exploring this design space, we explain the possible options in the paragraphs below.

Positional Encoding and Stem. Stems are typically designed with the position encoding in mind, so in this work we typically keep them paired for our explorations. We can use sinusoidal positional encoding of t with an MLP-style stem like NeRV and E-NeRV, or a single layer stem like NeRV. We can use positional encoding of x and y with a transformer-based stem like E-NeRV. We can use a grid encoding with a single layer stem like HiNeRV, or without a stem like FFNeRV. We can use a content-adaptive encoder on frames or pixel differences like HNeRV or DiffNeRV [30], although for the sake of simplicity and scope, we do not investigate along this axis in this work.

Blocks. We can use a basic NeRV block, consisting of a convolutional layer, PixelShuffle, and activation. Alternatively, we can use an additional convolutional layer for all (FFNeRV) or the first (E-NeRV) blocks. HiNeRV uses a ConvNext-based block with bilinear upsampling.

Skips. E-NeRV and HiNeRV propose two different skips. E-NeRV passes the t positional encoding through a learned FC for each NeRV block, and within the block performs a layer normalization on the NeRV features followed by a fusion with the t skip features. HiNeRV passes local grid features (learnable per block) to an FC layer, and adds the output to the result of the block's bilinear upsampling.

Other. Parameters are equally expensive to store regardless of location, but they are not equally expensive computationally. Later parameters process larger feature maps, and are therefore much heavier in terms of FLOPs and slower in terms of wall time. We control these as in prior works, with an expansion term `exp`, a reduction term r, and a kernel size term `ks`. For most works, `ks` is set to 3. The expansion term governs the relationship between the input channels ch_{in} and output channels ch_{out} for the first layer specifically: $ch_{\text{out}} = ch_{\text{in}} * \texttt{exp}$. The reduction term governs the change in channels from one layer to the next: $ch_{n+1} = ch_n/r$. HNeRV changes `ks` from 3 to 1 in the first layer and 3 to 5 in all other layers, and changes r from its default 2 to $r = 1.2$ (which allocates more parameters for later layers). Such changes are good for quality, but hurt speed.

3.2. Flexible Size via Weight Token Masking

We adopt the hyper-network approach described in NeRV-Enc, although we re-implement the method since at the time of this writing their code is not public. In this setup, a hypo-network is designed and initialized as a set of learnable parameters, corresponding to the weights and biases of each layer of the final hypo-network prediction. These are sometimes referred to as "shared" parameters, since they are optimized across the entire dataset of videos. In order to predict a hypo-network that corresponds to an actual video, the hyper-network backbone predicts weight tokens to modulate each layer of the base hypo-network. The modulation is a simple elementwise multiplication between hyper-network predictions and the learnable shared parameters. We give a more detailed walkthrough in the Appendix.

Hyper-networks produce encodings at a fixed size. To in-

Table 1. **Video regression for 1080×1920 UVG videos** for the set of established video INR compression architectures supported in our library. We report PSNR↑/MS-SSIM↑ [50] for configurations with 1.5M and 3M parameters, all based on training and evaluation with our own library, using settings faithful to the original papers when possible. We include learnable grids and decoders in the parameter counts, since these are part of the bitstream. We do not include the HNeRV and DiffNeRV encoders.

		Video						
Method	#Params	Beauty	Bosphorus	HoneyBee	Jockey	ShakeNDry	YachtRide	ReadySetGo
ONeRV [3]	1.5M	30.80/0.8458	29.78/0.8568	33.69/0.9554	26.86/0.8016	28.76/0.8480	25.40/0.7727	21.34/0.7012
	3M	31.75/0.8632	31.69/0.9015	37.14/0.9773	28.61/0.8357	31.56/0.9091	26.49/0.8123	23.04/0.7721
E-NeRV [30]	1.5M	31.28/0.8554	30.80/0.8809	37.27/0.9790	26.44/0.7907	32.14/0.9181	25.89/0.7932	21.58/0.7097
	3M	32.65/0.8789	33.31/0.9281	38.81/0.9835	29.02/0.8372	33.68/0.9339	27.52/0.8485	23.74/0.7887
NeRV [5]	1.5M	31.39/0.8568	31.01/0.8900	34.57/0.9656	27.55/0.8120	30.13/0.8877	25.94/0.7967	21.90/0.7188
	3M	32.26/0.8730	32.54/0.9191	36.47/0.9761	29.48/0.8473	31.83/0.9162	27.06/0.8362	23.36/0.7753
FFNeRV [25]	1.5M	31.64/0.8607	30.14/0.8612	35.15/0.9681	28.93/0.8422	30.83/0.8978	26.00/0.7935	22.44/0.7420
	3M	32.81/0.8816	32.50/0.9109	37.55/0.9797	31.41/0.8859	32.63/0.9242	27.54/0.8418	24.81/0.8222
HiNeRV [21]	1.5M	**33.57/0.8947**	**35.54/0.9611**	**39.21/0.9844**	**32.25/0.8978**	**34.11/0.9386**	**29.38/0.8883**	**27.73/0.8991**
	3M	**33.71/0.8987**	**37.43/0.9739**	**39.39/0.9849**	**34.71/0.9300**	**35.40/0.9543**	**30.39/0.9083**	**30.31/0.9362**
HNeRV [5]	1.5M	31.96/0.8674	31.70/0.9000	37.87/0.9805	29.90/0.8652	32.87/0.9254	26.25/0.8062	24.00/0.8075
	3M	32.84/0.8819	33.55/0.9306	38.72/0.9832	31.34/0.8841	33.62/0.9329	27.90/0.8577	25.72/0.8524
DiffNeRV [59]	1.5M	32.89/0.8794	32.57/0.9113	37.91/0.9793	31.01/0.8845	32.82/0.9222	27.53/0.8354	24.98/0.8309
	3M	33.63/0.8903	34.33/0.9372	39.19/0.9829	32.41/0.9001	34.07/0.9342	28.68/0.8669	26.50/0.8656

troduce some flexibility, and owing to the fact that the network is already amenable to repeating and expanding the weight token predictions, we introduce weight token masking, as shown in Figure 3. During training, for a number of random samples governed by a masking ratio, we disregard half of the weight tokens for each layer. The model then learns to encode necessary information in the never-masked portion of the weight tokens, and some information for higher quality in the sometimes-masked half. At inference time, one can use all tokens for highest quality, or the never-masked half for a 2× improvement in storage size.

4. Experiments

We run most of our experiments on the 7 videos from the UVG dataset [35] listed in Table 1, always at 1080p (1080×1920 resolution). Except for ShakeNDry, which has 300 frames, all videos have 600 frames. For our hypernetwork experiments, we train on the same 10,000 video subset of Kinetics-400 [14] as in prior work [6], and we test on a subset of UCF-101 [46] with 1 video per class. For the hybrid methods HNeRV and DiffNeRV, we implement PixelShuffle with rectangular strides to allow operating with images at 16:9 aspect ratios, instead of cropping frames as in those papers. We train all INR methods with the Adam optimizer [20], and a cosine annealing learning rate scheduler. We run on single GPUs, and wherever we benchmark wall-time, we use NVIDIA RTXA5000 GPUs. Different GPUs could have different wall-times; however,

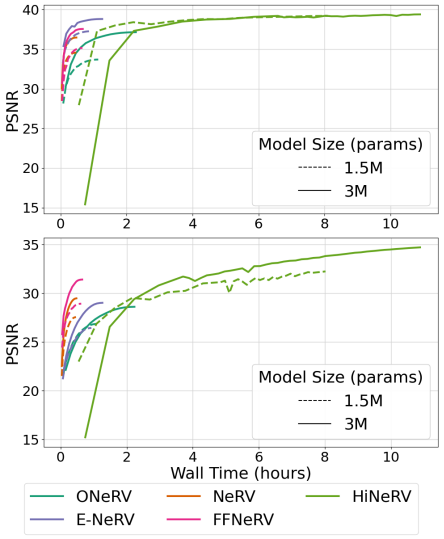

Figure 4. **Quality vs. wall time**, for UVG HoneyBee (top) and Jockey (bottom) at 1080p (Section 4.1).

we observe that the major trends in wall-time are consistent with FLOPs. For bpp results, we use both the 1.5M and 3M parameter models, quantizing each to 8, 7, 6, 5, and 4 bits before arithmetic coding [34]. For more details, see the Appendix.

We first verify our library's efficacy with by reproducing results for existing methods in Section 4.1. We then investigate the effectiveness of disentangled INR compo-

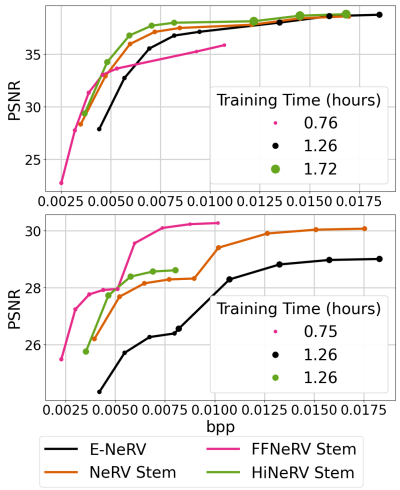

Figure 5. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **E-NeRV** for various **position-stem** combinations (Section 4.2).

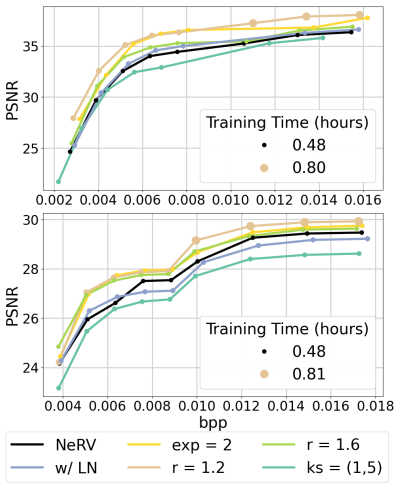

Figure 6. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **NeRV** with different **parameter distributions** (Section 4.2).

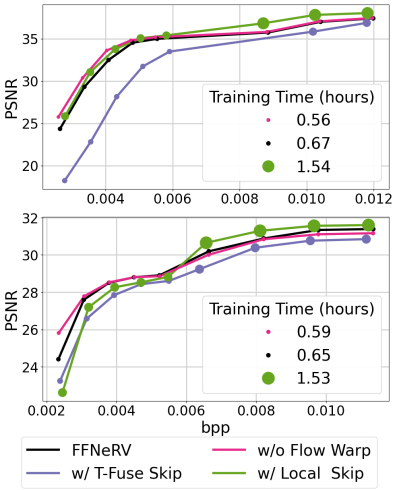

Figure 7. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **FFNeRV** with different **skip** connections and without **flow** warping (Section 4.2).

Table 2. **Weight Token Masking results.** We sacrifice some quality at full bit-width for superior quality at the more important half bit-width. We report bpp at 8-bit quantization.

Masking?		# Params		Metrics		
Train	Val	Min	Max	PSNR	MS-SSIM	bpp
N	n/a	5.15k	5.15k	23.22	0.6261	0.079
✓	✓	5.15k	10.3k	23.71	0.6451	0.079
✓	×	5.15k	10.3k	23.85	0.6492	0.157
×	n/a	2.45k	2.45k	22.55	0.6011	0.037
✓	✓	2.45k	4.90k	22.81	0.6130	0.037
✓	×	2.45k	4.90k	22.93	0.6163	0.075

Table 3. **Hypo-network size results.** We compare the 85.6k parameter hypo-network from NeRV-Enc with a larger one. We achieve better performance without increasing bitstream size.

# Params		Quality		FPS	
Total	Unique	PSNR	MS-SSIM	Enc	Dec
85.6k	24.1k	25.29	0.7319	**4455**	632.7
130k	24.1k	**25.92**	**0.7515**	4206	**717.1**

nents and suggest best principles for design in Section 4.2. We further benchmark the performance of existing models with short, medium, and long training times in Section 4.3. We formulate a state-of-the-art configuration, RNeRV, that performs especially well for short (2 minutes) and medium (30 minutes) encoding times. For even faster encoding, we explore the use of hyper-networks and give results for our novel weight token masking strategy in Section 4.4. Finally, to better analyze what neurons from different NeRV methods learn in terms of both content and motion, we provide XINC [38] analysis using our extended version of their library that we implement in Section 4.5.

4.1. Reproducing and Benchmarking Video INRs

We report reconstruction results for the set of video INR methods we reproduce in our library in Table 1. We re-use published configurations when possible, otherwise we perform extensive hyperparameter searches to find best configurations for 1.5M and 3M parameters. For example, we adjust the FFNeRV expansion from 8 to 4 to account for the smaller size compared to the 12M parameter models in their paper. When training methods at equal epochs and equal decoder parameter counts, HiNeRV has the best results for every video. In most cases, even the 1.5M parameter HiNeRV outperforms the competing methods at 3M parameters.

However, one must also consider the training time. We provide quality-time results in terms of wall time in Figure 4 (see Appendix for epochs and FLOPs). HiNeRV eventually achieves the best results, but requires significantly more time than competing methods. While E-NeRV improves on the speed of the original NeRV, FFNeRV is faster, while NeRV is the fastest. This motivates two questions: (1) which method is best with equal time, and (2) can we mix-and-match method components for better results?

4.2. Investigating NeRV Design Decisions

We show some key ablations here, with more in the Appendix. We measure the impact of the position encoding/stem choice on E-NeRV in Figure 5. We look at parameter distribution for NeRV in Figure 6 and the impact

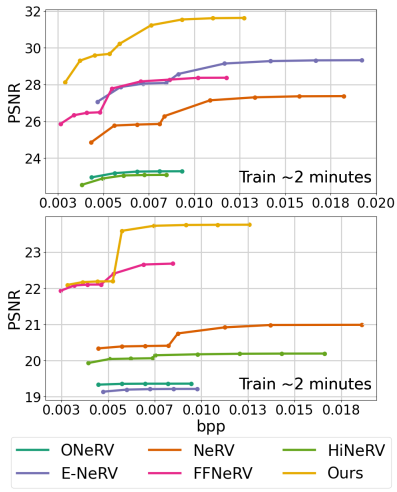

Figure 8. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p with "short" training time (Section 4.3).

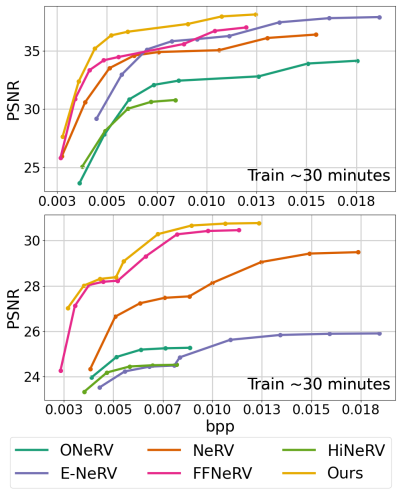

Figure 9. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p with "medium" training time (Section 4.3).

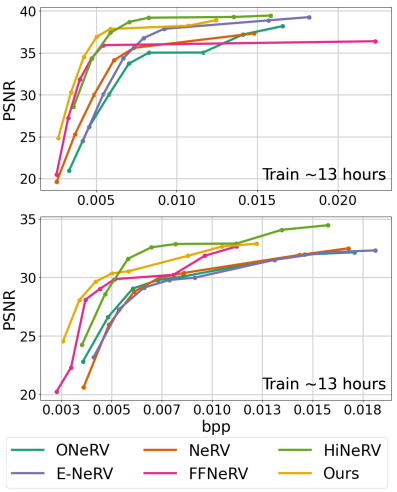

Figure 10. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p with "long" training time (Section 4.3).

of adding skips and removing flow on FFNeRV in Figure 7. We also consider blocks (see Appendix), but find that the FFNeRV, E-NeRV, and NeRV designs are roughly equivalent, while HiNeRV blocks are challenging to train without the other components of HiNeRV.

Based on these ablations, we first observe the promise of the FFNeRV stem. From Figure 5, it achieves very strong results on the Jockey video. In general, all other stems and positional encodings considered are better than the original E-NeRV stem and fixed positional encoding. We also notice, from Figure 6, that distributing parameters to later layers is helpful for performance. However, a key exception is doing this by changing the parameters, as in HNeRV, which seems to generally be unhelpful. Additionally, redistributing the parameters can dramatically increase training times (with a nearly 2x gap between the original $r = 2$ and $r = 1.2$), and must therefore be done with care. We observe that while the HiNeRV-style local skip helps FFNeRV in Figure 7, it is quite expensive. We also find that omitting flow warping saves time, with minimal quality penalty.

4.3. Optimizing NeRV Designs for Time

Based on our findings in Section 4.2, we propose a time-efficient alternative to existing models. The first is a combination of the other methods – we use the modified NeRV blocks from FFNeRV, the feature grids and stemless approach of FFNeRV, local skips and layer norms from E-NeRV, we change r from 2 to 1.2 for the 1.5M model and 1.4 for the 3M parameter model (necessary to preserve the equal size) while reducing the first-layer expansion to 4 as well. Notably we opt for the E-NeRV skip in spite of its struggles in Figure 7; we hypothesize it requires the layer normalization [2] to work well.

We show results for existing NeRV configurations compared to our proposed configuration, in terms of PSNR/bpp, for encoding time equivalent to 20 NeRV epochs (short) in Figure 8, 300 NeRV epochs (medium) in Figure 9, and 300 HiNeRV epochs (long) in Figure 10. As expected, HiNeRV struggles at short and medium training times, although it is notably sometimes better than E-NeRV and ONeRV. We note that our configuration achieves the best results for both Jockey and HoneyBee at short and medium training times. Additionally, our configuration achieves the best results at low bpp even for long training time. Due to the strength and orthogonality of their individual contributions, when existing video INR methods are combined, they are worth even more than the sum of their parts.

4.4. Hyper-network Improvements

While we call 2 minutes "short" encoding time for the 600 frame, 1080p UVG videos, this is nowhere near the real-time encoding speeds that are desirable for many applications. By contrast, hyper-network methods can perform encoding with a single forward pass. However, hyper-networks like NeRV-Enc do not come close to the quality-size performance of other methods. To help solve this, we introduce Weight Token Masking to enable hyper-networks to encode videos with more adaptive bitstream lengths. We show results in Table 2 for our improved HyperNeRV with adaptive weight token masking, where masked tokens are not stored, as described in Section 3. We achieve better PSNR and MS-SSIM than the baseline NeRV-Dec networks at equal bpp, and when we do not perform the masking on those same networks (doubling the size), we get slightly better quality. While such small improvements would not justify the increase in size, this still signals a promising di-

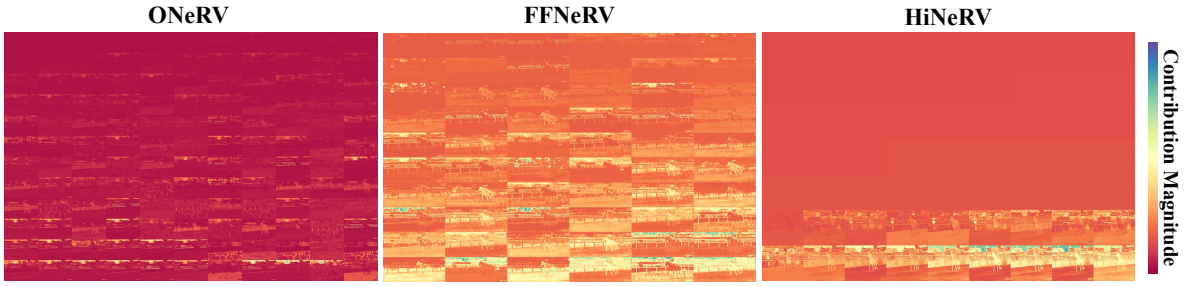

Figure 11. **XINC contribution maps** on 1080p Jockey. XINC dissects an INR to understand what parts of the visual signal are represented by each neuron (2D convolution kernel). We show contribution maps for the last (head) layer, sorted by magnitude for ease of comparison. Darker red corresponds to lower contribution from that kernel for that spatial location, while blue/purple corresponds to high contribution. ONeRV tends to show predominantly near-zero contributions, while most HiNeRV neurons lack discernible spatial correlations.

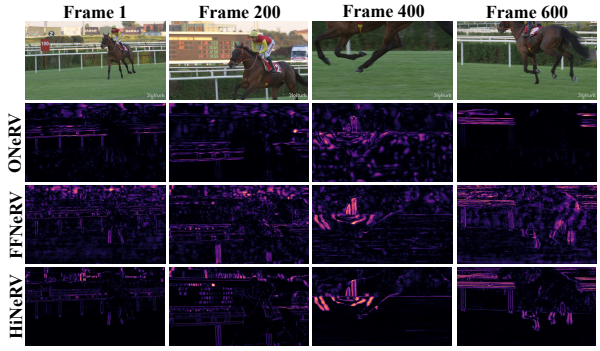

Figure 12. **XINC motion analysis** for the last (head) layer, for 1080p Jockey. We show fluctuation in total kernel contributions in response to motion between adjacent frames at various points in the video. While HiNeRV's changes are driven by fine differences between frames, ONeRV exhibits much less structure.

rection for exploration that we leave for future work.

We also examine the role of "shared parameters" (base hypo-network size) in Table 3. We can increase the size of the hypo-network for significant improvements to MS-SSIM and PSNR at equal bpp. Future work on hyper-networks ought to focus even more on the hypo-network design, optimizing the design of the base hypo-network to be as large as is useful (since it can be "installed" with the algorithm) while allowing the unique parameters to be as small as possible (since these are actually stored per-clip).

4.5. Analysis

We finally perform some qualitative analysis to better understand the different NeRV methods. We extend XINC [38] to dissect and analyze representations not just from NeRV, but also for other NeRV variants in this paper, including hypo-NeRV (see the Appendix). We show contribution maps for their head (last) layers in Figure 11. We observe that ONeRV stands out compared to other methods with many kernels in the head layer giving low, near-zero contributions. Notably HiNeRV has many maps without as much contrast as those from other methods.

In Figure 12 we show how the different networks handle motion, by taking the differences in total contributions (all maps summed) for adjacent frames. Notably the best method, HiNeRV, seems to have very structured changes, focusing specifically on adapting contributions for the fine details that change between frames. By contrast, FFNeRV, perhaps due to its flow-warping constraint, has contribution changes more spread out spatially. In general the weaker method, ONeRV, noticeably has less structured changes due to the motion between frames. For further analysis, including for hypo-networks, see the Appendix.

5. Conclusion

We have provided a new library which integrates NeRV-like methods to allow for the disentangling of their components, and the assembly of superior, time-optimal NeRVs. We have distilled the principles necessary for the creation of such networks and proposed a highly effective novel configuration. We proposed Hyper-network innovations for smaller hypo-networks with better quality, namely weight token masking and allocating additional base parameters for the hypo-network. We extended XINC to analyze how these networks represent content and motion in videos.

Acknowledgments. This work was partially supported by NSF CAREER Award (#2238769) and Dolby-UMD Joint Seed Grant. The authors would like to thank Fangjun Pu, Peng Yin, Birendra Kathariya, Tong Shao, and Guan-Ming Su for their feedback. The authors acknowledge UMD's supercomputing resources made available for conducting this research. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, Dolby, or the U.S. Government.

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. 1, 2

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 7

[3] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser Nam Lim, and Abhinav Shrivastava. Nerv: Neural representations for videos. *Advances in Neural Information Processing Systems*, 34:21557–21568, 2021. 2, 3, 5

[4] Hao Chen, Matt Gwilliam, Bo He, Ser-Nam Lim, and Abhinav Shrivastava. Cnerv: Content-adaptive neural representation for visual data, 2022. 2

[5] Hao Chen, Matthew Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. Hnerv: A hybrid neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10270–10279, 2023. 2, 3, 5

[6] Hao Chen, Saining Xie, Ser-Nam Lim, and Abhinav Shrivastava. Fast encoding and decoding for implicit video representation, 2024. 2, 3, 5, 1

[7] Yinbo Chen and Xiaolong Wang. Transformers as meta-learners for implicit neural representations, 2022. 2

[8] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 2

[9] Ge Gao, Ho Man Kwan, Fan Zhang, and David Bull. Pnvc: Towards practical inr-based video compression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3068–3076, 2025. 2

[10] Sharath Girish, Abhinav Shrivastava, and Kamal Gupta. Shacira: Scalable hash-grid compression for implicit neural representations, 2023. 2

[11] Kilichbek Haydarov, Aashiq Muhamed, Xiaoqian Shen, Jovana Lazarevic, Ivan Skorokhodov, Chamuditha Jayanga Galappaththige, and Mohamed Elhoseiny. Adversarial text to continuous image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6316–6326, 2024. 3

[12] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6132–6142, 2023. 2

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. 1

[14] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017. 5

[15] Mehrdad Khani, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Efficient video compression via content-adaptive super-resolution. *ICCV*, 2021. 2

[16] Chiheon Kim, Doyup Lee, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Generalizable implicit neural representations via instance pattern composers. *arXiv preprint arXiv:2211.13223*, 2022. 2

[17] Hyunjik Kim, Matthias Bauer, Lucas Theis, Jonathan Richard Schwarz, and Emilien Dupont. C3: High-performance and low-complexity neural compression from a single image or video, 2023. 1, 2

[18] Jina Kim, Jihoo Lee, and Je-Won Kang. *SNeRV: Spectra-Preserving Neural Representation for Video*, page 332–348. Springer Nature Switzerland, 2024. 2

[19] Subin Kim, Sihyun Yu, Jaeho Lee, and Jinwoo Shin. Scalable neural video representations with learnable positional features. In *Advances in Neural Information Processing Systems*, 2022. 2

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 5

[21] Ho Man Kwan, Ge Gao, Fan Zhang, Andrew Gower, and David Bull. Hinerv: Video compression with hierarchical encoding-based neural representation. In *Advances in Neural Information Processing Systems*, pages 72692–72704. Curran Associates, Inc., 2023. 1, 2, 3, 5

[22] Ho Man Kwan, Ge Gao, Fan Zhang, Andrew Gower, and David Bull. Nvrc: Neural video representation compression, 2024. 2

[23] Théo Ladune, Pierrick Philippe, Félix Henry, Gordon Clare, and Thomas Leguay. Cool-chic: Coordinate-based low complexity hierarchical image codec, 2023. 2

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

[25] Joo Chan Lee, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Ffnerv: Flow-guided frame-wise neural representations for videos. In *Proceedings of the 31st ACM International Conference on Multimedia*, page 7859–7870. ACM, 2023. 2, 3, 5

[26] Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression, 2021. 1, 2

[27] Jiahao Li, Bin Li, and Yan Lu. Hybrid spatial-temporal entropy modelling for neural video compression. In *Proceedings of the 30th ACM International Conference on Multimedia*. ACM, 2022. 2

[28] Jiahao Li, Bin Li, and Yan Lu. Neural video compression with diverse contexts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22616–22626, 2023.

[29] Jiahao Li, Bin Li, and Yan Lu. Neural video compression with feature modulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26099–26108, 2024. 2

[30] Zizhang Li, Mengmeng Wang, Huaijin Pi, Kechun Xu, Jianbiao Mei, and Yong Liu. E-nerv: Expedite neural video representation with disentangled spatial-temporal context, 2022. 2, 3, 4, 5

[31] Haojie Liu, Tong Chen, Ming Lu, Qiu Shen, and Zhan Ma. Neural video compression using spatio-temporal priors. *arXiv preprint arXiv:1902.07383*, 2019. 2

[32] Shishira R Maiya, Sharath Girish, Max Ehrlich, Hanyu Wang, Kwot Sin Lee, Patrick Poirson, Pengxiang Wu, Chen Wang, and Abhinav Shrivastava. Nirvana: Neural implicit representations of videos with adaptive networks and autoregressive patch-wise modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14378–14387, 2023. 2

[33] Shishira R Maiya, Anubhav Gupta, Matthew Gwilliam, Max Ehrlich, and Abhinav Shrivastava. Latent-inr: A flexible framework for implicit representations of videos with discriminative semantics. In *European Conference on Computer Vision*, pages 285–302. Springer, 2024. 3

[34] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 5, 1

[35] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM multimedia systems conference*, pages 297–302, 2020. 5

[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, 2020. 2

[37] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41 (4):1–15, 2022. 2

[38] Namitha Padmanabhan, Matthew Gwilliam, Pulkit Kumar, Shishira R Maiya, Max Ehrlich, and Abhinav Shrivastava. Explaining the implicit neural canvas: Connecting pixels to neurons by tracing their contributions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10957–10967, 2024. 2, 6, 8, 7

[39] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G. Anderson, and Lubomir Bourdev. Learned video compression. In *ICCV*, 2019. 1, 2

[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. 2

[41] Jens Eirik Saethre, Roberto Azevedo, and Christopher Schroers. Combining frame and gop embeddings for neural video representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9253–9263, 2024. 2

[42] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G. Baraniuk. Wire: Wavelet implicit neural representations, 2023. 2

[43] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016. 2

[44] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 2

[45] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images, 2021. 3

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

[47] Yannick Strümpler, Janis Postels, Ren Yang, Luc van Gool, and Federico Tombari. Implicit neural representations for image compression, 2022. 2

[48] 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

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

[50] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, pages 1398–1402. Ieee, 2003. 5

[51] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003. 2

[52] Chang Wu, Guancheng Quan, Gang He, Xin-Quan Lai, Yunsong Li, Wenxin Yu, Xianmeng Lin, and Cheng Yang. Qs-nerv: Real-time quality-scalable decoding with neural representation for videos. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 2584–2592, 2024. 2

[53] Dejia Xu, Peihao Wang, Yifan Jiang, Zhiwen Fan, and Zhangyang Wang. Signal processing for implicit neural representations, 2022. 2

[54] Yunjie Xu, Xiang Feng, Feiwei Qin, Ruiquan Ge, Yong Peng, and Changmiao Wang. Vq-nerv: A vector quantized neural representation for videos, 2024. 2

[55] Hao Yan, Zhihui Ke, Xiaobo Zhou, Tie Qiu, Xidong Shi, and Dadong Jiang. Ds-nerv: Implicit neural video representation with decomposed static and dynamic codes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 23019–23029, 2024. 2

[56] Sihyun Yu, Jihoon Tack, Sangwoo Mo, Hyunsu Kim, Junho Kim, Jung-Woo Ha, and Jinwoo Shin. Generating videos with dynamics-aware implicit generative adversarial networks, 2022. 3

[57] Xinjie Zhang, Ren Yang, Dailan He, Xingtong Ge, Tongda Xu, Yan Wang, Hongwei Qin, and Jun Zhang. Boosting neural representations for videos with a conditional decoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2556–2566, 2024. 2

[58] Yunfan Zhang, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. Implicit neural video compression, 2021. 2

[59] Qi Zhao, M. Salman Asif, and Zhan Ma. Dnerv: Modeling inherent dynamics via difference neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2031–2040, 2023. 2, 5

[60] Qi Zhao, M. Salman Asif, and Zhan Ma. Pnerv: Enhancing spatial consistency via pyramidal neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19103–19112, 2024. 2

How to Design and Train Your Implicit Neural Representation for Video Compression

Supplementary Material

6. Additional Results

We report PSNR and MS-SSIM vs. bpp for an average across the 7 videos we use in this paper, for "short" training time in Figure 13, and for "medium" training time in Figure 14. When we compute an average, we average in both size and quality simultaneously. Specifically, for each model, for every video, we take the bpp and PSNR (or MS-SSIM) for the 3M parameter model, with 8 bit quantization plus arithmetic coding, and average them. We do this for 7 bits, 6 bits, and so on. We repeat the process for the 1.5M parameter models.

We report MS-SSIM vs. bpp corresponding to the figures in Section 4.2 in Figure 15, Figure 16, and Figure 17. We report more ablations for FFNeRV in terms of PSNR in Figure 18, Figure 19, and Figure 20. We report more ablations for E-NeRV in terms of PSNR in Figure 21, Figure 22, and Figure 23. We report more ablations for NeRV in terms of PSNR in Figure 24, Figure 25, and Figure 26.

We report MS-SSIM vs. bpp corresponding to the figures in Section 4.3 in Figure 27, Figure 28, and Figure 29. We also try a version of HiNeRV with $r = 1.6$ and compare it to HiNeRV and RNeRV in the "long" time setting, for PSNR in Figure 30 and MS-SSIM in Figure 31.

We perform evaluations for the k400-trained HyperNeRV on the UVG videos, downsampled to 256 height and center-cropped. We report the PSNR and MS-SSIM in Table 4. Keep in mind that these results are both higher bpp, lower resolution, and lower quality than anything we observe with the NeRV models themselves. However, the trends are quite different – HyperNeRV performs better on the videos with motion, whereas it struggles substantially on the HoneyBee video which is easy for most INR-based methods (perhaps it is out of the training distribution).

We also consider the performancne of the NeRV models with different hybrid encoders in Table 5. We reserve these for supplementary due to the results in the "# Encode" column. With their native settings, HNeRV adds some bpp without any quality gains. To get some gains, the architecture would have to be reworked (possibly by shifting parameters, but this would add encoding time). DiffNeRV adds a very large amount of extra information to store which renders any comparison with other methods unfair. We leave the issue of reconciling the size discrepancies between these and the other methods for future work.

7. Implementation Details

7.1. NeRV Compression.

We follow the quantization procedure as explained in HiNeRV [21]. Any time we report bpp, we measure the actual space requirement after compressing with torchac [34]. We then measure the quality after loading the compressed model. While our library supports quantization-aware training, we observe limited benefit and skip it.

To compute a PSNR/bpp or MS-SSIM/bpp curve, we train each method at a 1.5 million and 3 million parameter setting. We then quantize each setting at 8, 7, 6, 5, and 4 bits, then compress it with torchac. We then report the quality/size pairs in strictly ascending order; if some higher size achieves worse quality than a previous size, we do not report it. This sometimes happens in the case where we transition from an 8-bit 1.5M parameter model, to a 4-bit 3M parameter model, for example.

There are rare cases where the quality/size curve performs well for the points corresponding to the 1.5M parameter model, and poorly for the 3M parameter model (such as with the FFNeRV stem in Figure 24. This is not necessarily a reflection of poor stem design, but rather a limitation of our study. That is, every single component of a NeRV model affects the success of every other component. It is likely that with that our specific configuration for that experiment is suboptimal; perhaps with a different fc_{dim} or r or `exp` we would observe better performance. In general we test multiple configurations to avoid such cases, but, as our study in part helps prove, it is quite difficult to be sure that one has found a true global optimum for all the different hyperparameters and components of these models.

7.2. HyperNeRV.

Training details for the larger Hyper-Network Hypo-Network settings:

- Video: The frames from the input video are organized into clips of 8 consecutive frames each. In contrast to [6], we train on all frames of the video (separated into clips). Each clip is treated as an independent input when predicting the weights of the hyponet during training and inference.
- Batch size: 32
- Tokenizer patch size: 64
- HypoNeRV position embedding dimension: 16
- HypoNeRV activation layer: GELU [13]
- HypoNeRV fc_{dim}: 20

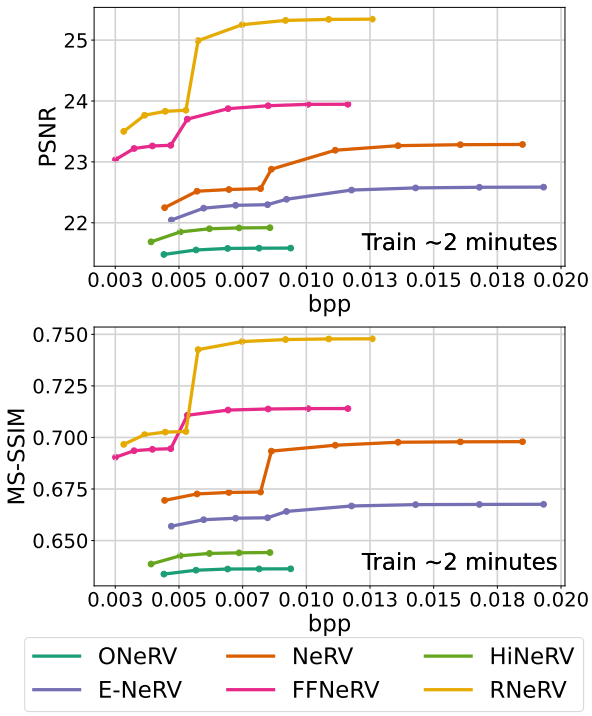

Figure 13. **Quality vs. size**, averaged across the 7 UVG videos we use in this paper, for **"short"** training time.

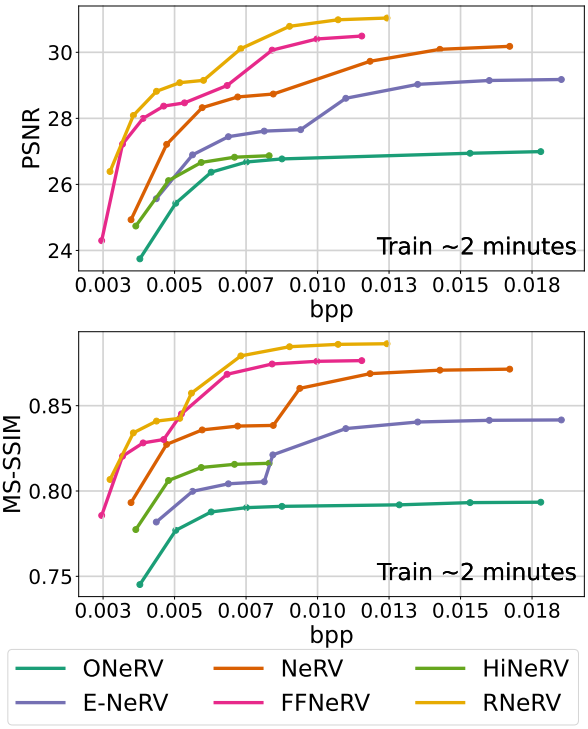

Figure 14. **Quality vs. size**, averaged across the 7 UVG videos we use in this paper, for **"medium"** training time.

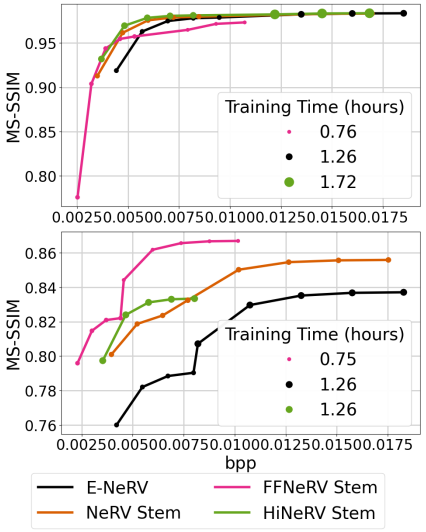

Figure 15. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **E-NeRV** for various **position-stem** combinations.

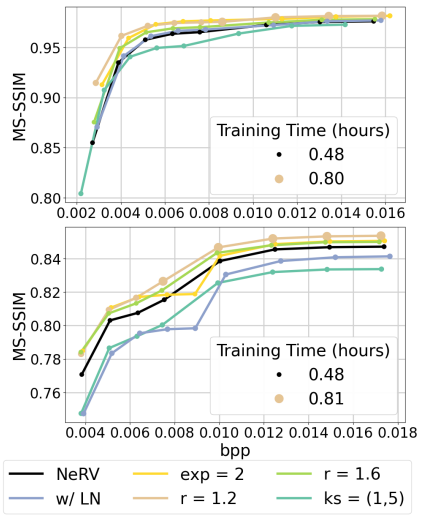

Figure 16. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **E-NeRV** with different **parameter distributions**.

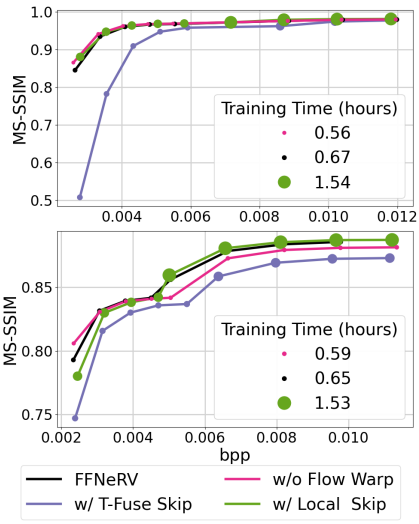

Figure 17. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **FFNeRV** with different **skip** connections and without **flow** warping.

- HypoNeRV kernel sizes (first layer to last): 1, 3, 3, 3
- HypoNeRV upscale factors (PixelShuffle strides): 4, 4, 4, 4
- HypoNeRV token numbers (first layer to last): 4, 80, 16, 0
- HypoNeRV token dimensions (first layer to last): 256, 240, 240, 0
- Transformer token dimension and feed-forward dimen-

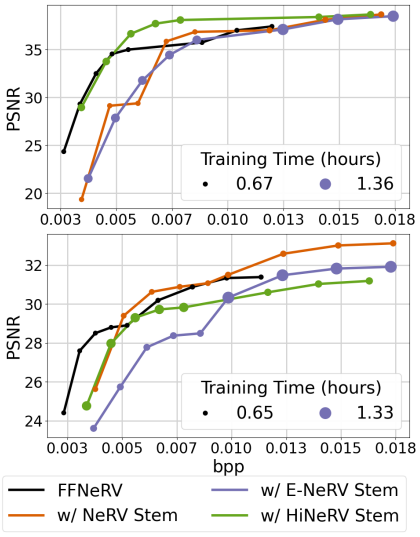

Figure 18. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **FFNeRV** for various **position-stem** combinations.

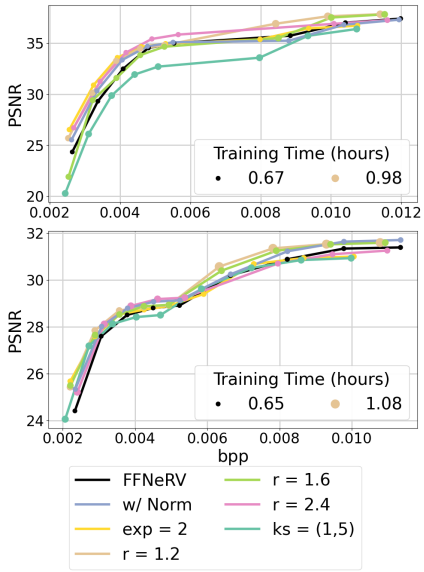

Figure 19. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **FFNeRV** with different **parameter distributions**.

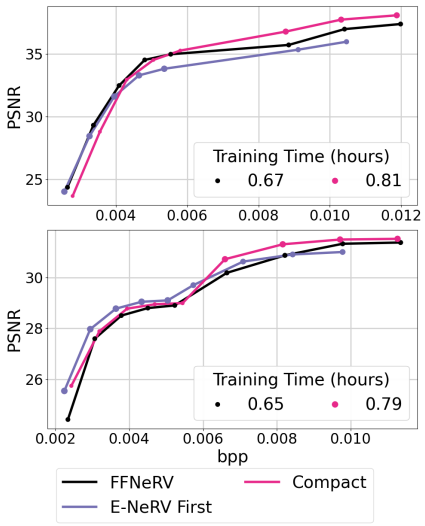

Figure 20. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **FFNeRV** with different **block designs**.

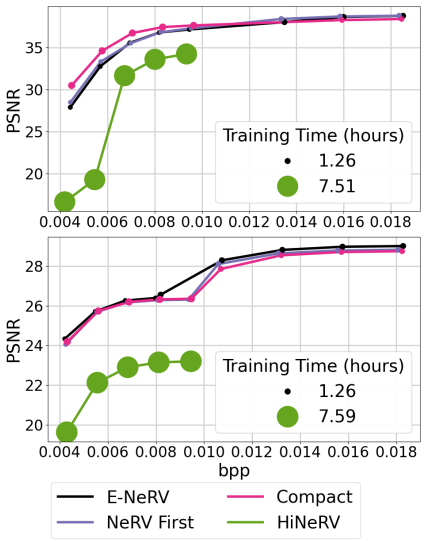

Figure 21. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **E-NeRV** for various **block designs**.

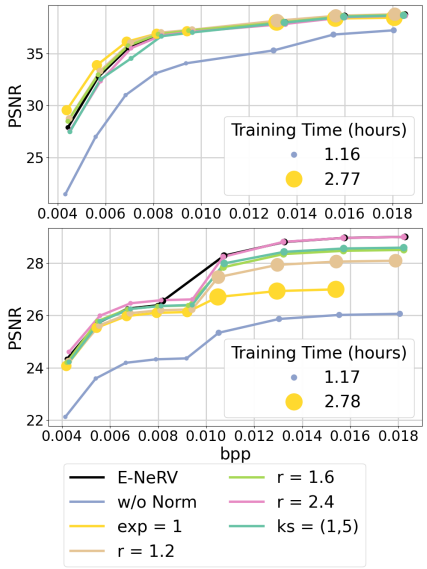

Figure 22. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **E-NeRV** with different **parameter distributions**.

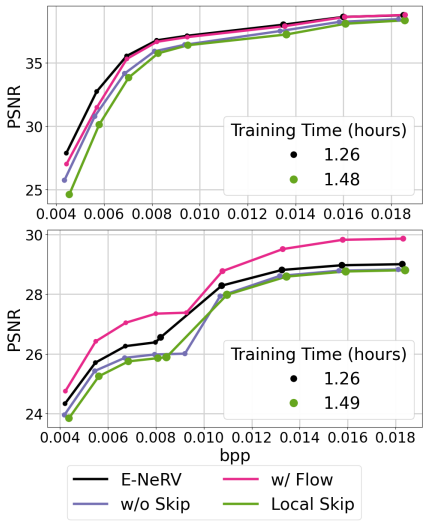

Figure 23. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **E-NeRV** with different **skip** connections and without **flow** warping.

sion for transformer encoder layers: 720 and 2880; 12 heads, 6 blocks, for HyperNeRV of size 47.9M.

- Optimizer: Adam
- Learning rate: 0.0001

HypoNeRV $fc_{\text{dim}} = 16$. For HypoNeRV with $fc_{\text{dim}} = 16$, the following settings are altered from the base HyperNetwork training.

- HypoNeRV fc_{dim}: 16
- HypoNeRV layers token numbers: 4, 64, 16, 0
- HypoNeRV layers token dimensions: 256, 288, 288, 0

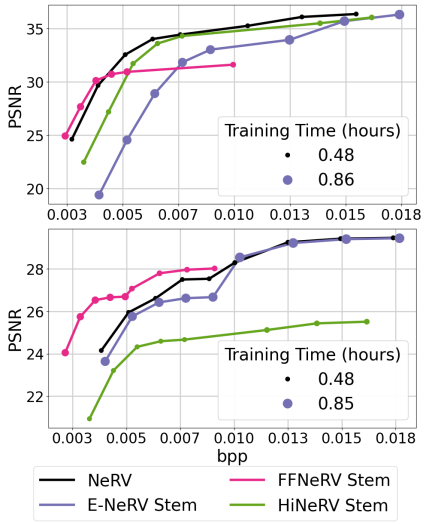

Figure 24. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **NeRV** for various **position-stem** combinations.

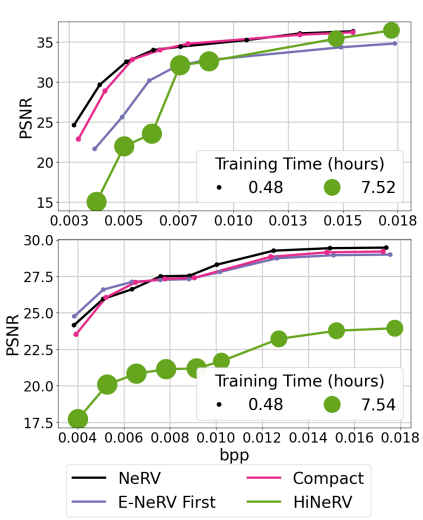

Figure 25. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **NeRV** with different **block designs**.

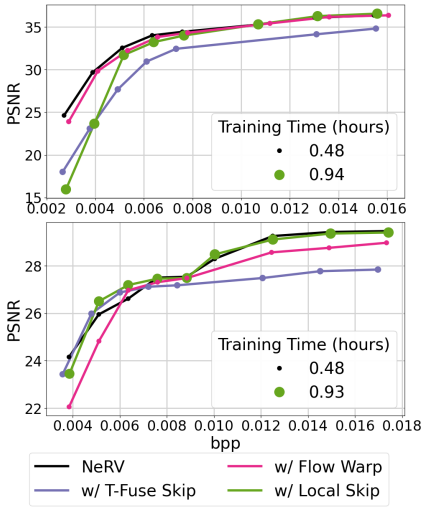

Figure 26. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p for **NeRV** with different **skip** connections and without **flow** warping.

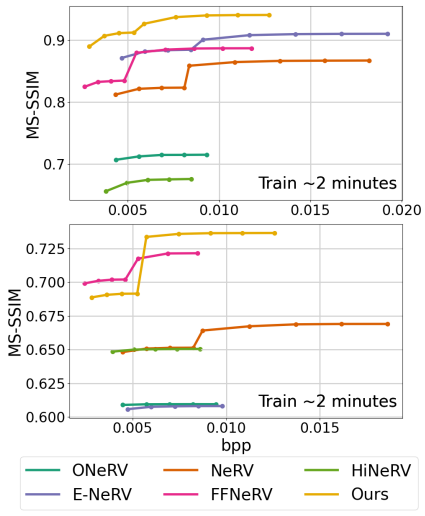

Figure 27. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p with "short" training time.

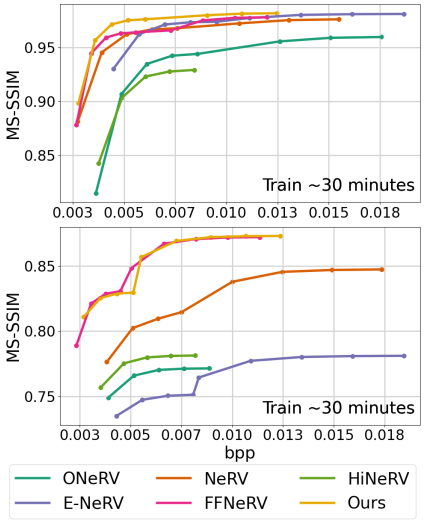

Figure 28. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p with "medium" training time.

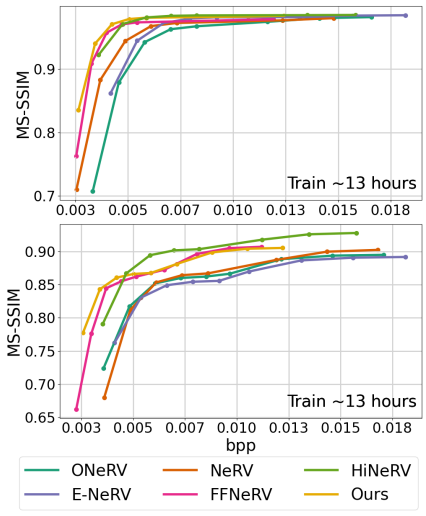

Figure 29. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p with "long" training time.

Weight masking. For weight masking, we use the following modified token settings while keeping the other settings same as the base training.

We have two configurations, a larger one, and a smaller one.

Larger weight masking model.

- HypoNeRV layers min token numbers: 1, 32, 4, 0
- HypoNeRV layers max token numbers: 2, 64, 8, 0
- HypoNeRV masking ratio: 0.5
- HypoNeRV token dimensions: 256, 144, 72, 0

Smaller weight masking model.

- HypoNeRV layers min token numbers: 0, 16, 2, 0
- HypoNeRV layers max token numbers: 0, 32, 4, 0
- HypoNeRV masking ratio: 0.5
- HypoNeRV token dimensions: 256, 144, 72, 0

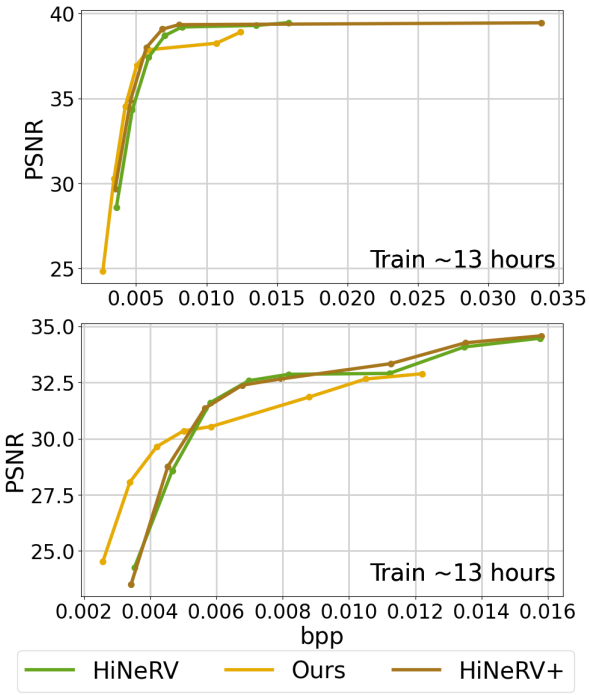

Figure 30. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p.

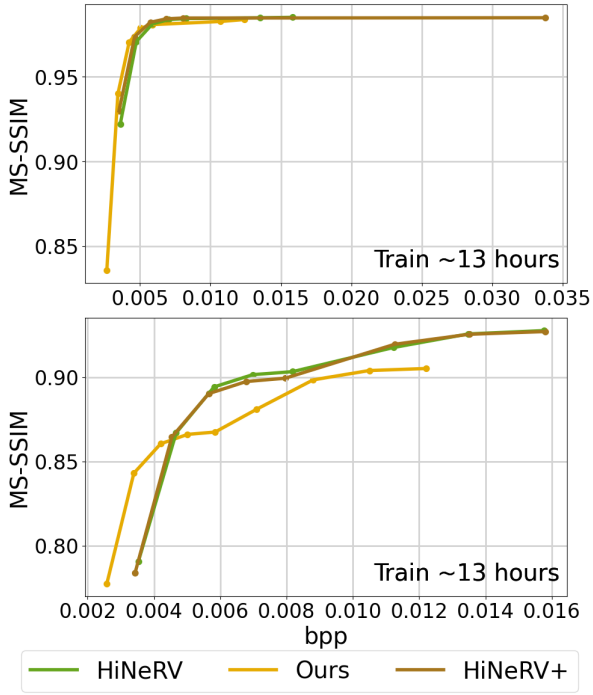

Figure 31. **Quality vs. size**, for HoneyBee (top) and Jockey (bottom) at 1080p.

Table 4. HyperNeRV compression results on UVG at 256×256 resolution. PSNR/MS-SSIM.

		Video						
Method	#Params	Beauty	Bosphorus	HoneyBee	Jockey	ShakeNDry	YachtRide	ReadySetGo
HyperNeRV	130k/24.1k	30.22/0.8370	26.52/0.7212	21.33/0.5223	25.14/0.7376	25.66/0.5715	25.36/0.6728	21.14/0.5821

8. NeRV Walkthrough

See a detailed diagram of the basic NeRV stem from HNeRV [5] in Figure 34. See a detailed diagram of the first NeRV block corresponding to that stem in Figure 35. From there, the features are upsampled by subsequent NeRV blocks, with upsampling dictated by the channel expansion (from convolution) and the PixelShuffle stride which converts channels to spatial dimensions. Once at full frame/image resolution, the features are processed by a final convolutional layer to convert from feature dimension to color channel dimension (3). The result is passed through a specialized activation, such as a sigmoid, tanh, or adding 0.5 to the output. This final output is equivalent to the original image/frame.

9. INR Hyper-Network Walkthrough

We provide a brief walkthrough of the hyper-network setup, although it is explained in prior work as well [6]. To understand the INR hyper-network setup, it is easiest to start with the prediction of the hyper-network: the hypo-network. For our purposes, this network is a simple sequence of 4 NeRV blocks, which take a time positional encoding as input, and upsample it (via convolution and PixelShuffle) until it is the size of the image the NeRV model is meant to represent. This NeRV model has a total of 4 learnable convolutional layers. So, our hyper-network must predict the parameters for 4 convolutional layers.

The model does so in two parts. First, it initializes and learns a set of base, or "shared" parameters, which correspond to the hypo-network exactly. That is, the base parameters contain all the weights and the biases corresponding to all 4 layers of the hypo-network. If we want to have an 85.6k parameter hypo-network, we learn 85.6k base parameters. Remember that the hyper-network learns at the dataset-level, not the video level. In a realistic setting, this would be installed as an encoder, and the base parameters would be installed as part of the decoder. They would not be transmitted over the wire in real-time, since they can be learned up front. Or else, they would be transmitted, but

Table 5. **NeRV-methods with hybrid encoders**. Both videos are at 1080p, all methods are trained for 300 epochs. Grids and decoder parameters are included in the # Params; we compute the size of hybrid encodings, # Encode, (which must be stored in the bitstream) as $fc_h \times fc_w \times e_{\text{dim}}$. We also include results from the original methods for reference. We do not include these comparisons in the main paper since the relatively large size of the content and difference embeddings make fair comparison very challenging.

		Storage Size		Video	
Method	Encoding	# Params	# Encode	HoneyBee	Jockey
E-NeRV	Fixed	1.5M	n/a	37.27/0.9790	26.44/0.7907
		3M	n/a	38.81/0.9835	29.02/0.8372
E-NeRV	HNeRV	1.5M	0.346M	35.17/0.9699	23.73/0.7291
		3M	0.346M	38.23/0.9822	27.21/0.8025
E-NeRV	DiffNeRV	1.5M	4.47M	35.46/0.9713	29.88/0.8754
		3M	4.47M	38.64/0.9834	31.91/0.9004
FFNeRV	Grid	1.5M	n/a	35.15/0.9681	28.93/0.8422
		3M	n/a	37.55/0.9797	31.41/0.8859
FFNeRV	HNeRV	1.5M	0.346M	36.18/0.9749	27.94/0.8216
		3M	0.346M	37.68/0.9806	30.08/0.8574
HiNeRV	Grid	1.5M	n/a	**39.21/0.9844**	32.25/0.8978
		3M	n/a	**39.39/0.9849**	34.71/0.9300
HiNeRV	HNeRV	1.5M	11.1M	38.49/0.9826	**34.62/0.9384**
		3M	11.1M	39.24/0.9846	**35.64/0.9447**
HiNeRV	DiffNeRV	1.5M	58.8M	37.99/0.9812	33.12/0.9170
		3M	58.8M	38.95/0.9840	34.12/0.9268
HNeRV	HNeRV	1.5M	1.38M	37.87/0.9805	29.90/0.8652
		3M	1.38M	38.72/0.9832	31.34/0.8841
DiffNeRV	DiffNeRV	1.5M	4.23M	37.91/0.9793	31.01/0.8845
		3M	4.23M	39.19/0.9829	32.41/0.9001

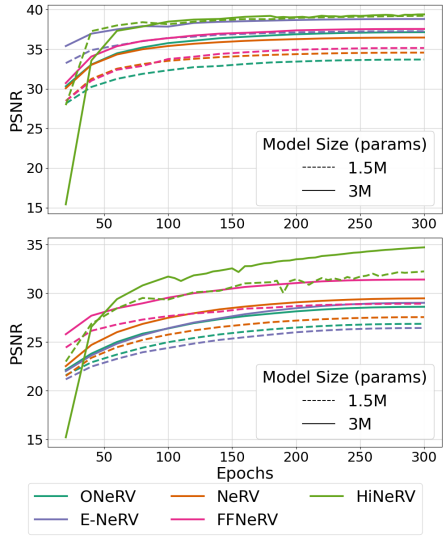

Figure 32. Reconstruction quality over training epochs, for UVG HoneyBee (top) and Jockey (bottom) at 1080p.

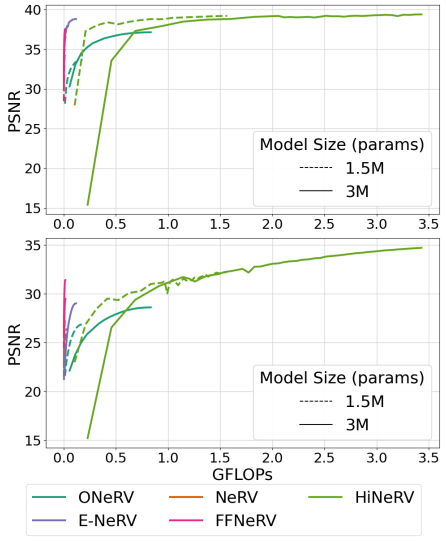

Figure 33. Reconstruction quality over GFLOPs, for UVG HoneyBee (top) and Jockey (bottom) at 1080p.

only a single time. So, we do not consider the base parameters when computing bpp or otherwise making compression storage considerations.

On their own, the base parameters cannot represent ev-

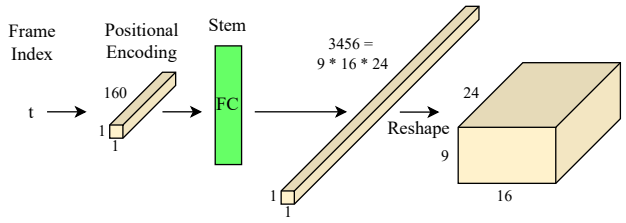

Figure 34. We show a detailed walkthrough of the NeRV stem, including sizes for a NeRV with 160-dimension positional encoding, $fc_{\text{dim}} = 12$, $fc_w = 16$, $fc_h = 9$. See Figure 35 for a walkthrough of the NeRV block, whose input is this stem's output.

ery single video. Instead, they are modulated, by the transformer backbone of the hyper-network, for any given input clip. To perform this modulation, we predict a set of "unique parameters." We set designate a certain amount of learnable weight tokens, corresponding to the different layers of the hypo-network, and give these as input to the hyper-network, along with the video tokens. We take the output weight tokens for each layer, and process them with fully-connected layers to convert from the transformer token dimension to a weight token dimension. We then repeat and reshape the post-FC tokens, perform an elementwise multiplication with the shared parameters, and normalize. The result is the hypo-network corresponding to the input clip. We illustrate this process for a single layer in Figure 36.

10. XINC Analysis Walkthrough

10.1. Adapting XINC

XINC, introduced in [38], is a framework designed to investigate how neurons in an image or video INR encode signals they are trained to represent. By dissecting the contributions of each neuron in every layer of the network to each output pixel, the framework aids in understanding the behavior of these INRs. We adapt this framework to analyze the last (head) layer of the various NeRV variants we study in this work, as well as the HypoNeRV model. Specifically, we generate contribution maps for the kernels in the head layer of these models (Figures 11, 37 and 40) and leverage these maps for motion analysis (Figures 12 and 41).

For the ONeRV, NeRV, HNeRV, E-NeRV and FFNeRV networks, integrating XINC is relatively straightforward since their head layers consist of a single learnable convolutional layer, akin to the NeRV model studied in [38], followed by an optional activation function. Given an input v_{in} of shape $h \times w \times ch_{in}$ and ch_{out} output channels, each of the $ch_{in} * ch_{out}$ convolutional kernels is treated as a neuron with a distinct contribution map. We compute these maps by independently convolving each kernel over v_{in} and storing the outputs to produce a set of $h \times w \times (ch_{in} * ch_{out})$ maps. These maps are subsequently passed through an optional activation layer to obtain final kernel contributions.

For HiNeRV, we begin by following the approach outlined above. However, since HiNeRV is a patch-based method, we stitch together contributions to pixels in different patches of a frame to construct the full-frame kernel contributions.

Adapting XINC for the HypoNeRV requires additional considerations due to the presence of a PixelShuffle operation in the head layer, which redistributes channel information into the spatial domain. To correctly interpret kernel contributions at the output resolution, we account for this rearrangement by following the procedure outlined in [38]. We omit applying proximity correction since there are no downstream convolutional layers that follow the head layer, as was originally applied in XINC.

10.2. XINC on HyperNeRV

Figure 37 presents the XINC contribution maps for the last layer of HypoNeRV, which consists of a convolutional layer followed by PixelShuffle. Since the head layer contains over 750 neurons, we sort them by total contribution magnitude and display a subset of uniformly sampled representative kernels. Unlike the NeRV variants shown in Figure 11, HypoNeRV's head layer features a higher number of kernels, necessitated by the channel-to-space rearrangement imposed by PixelShuffle to output the 3 channel frame. The selected contributions span a diverse range, from high (blue/purple) to low (dark red), highlighting the variation in neuron responses across different parts of the scene. Figure 38 illustrates how PixelShuffle modifies the contribution patterns of individual kernels in the head layer. The left panel shows a single kernel's contribution map, where the stride-based shuffling operation enforces a sparse activation pattern with specific zeroed-out locations. The right panel provides a reference map indicating valid contribution sites within a PixelShuffle group, revealing the structured nature of these transformations. Each kernel in a group of $stride^2$ kernels operated on by PixelShuffle can contribute to exactly one position within the $stride \times stride$ local window. Due to the interpolation artifacts that may arise, this visualization is best examined at full resolution.

Figure 41 analyzes how HypoNeRV neurons adjust their contributions over time by comparing two types of transitions: (1) between the first frames of consecutive clips and (2) between consecutive frames within the same clip. In clip-to-clip transitions, contributions shift primarily in dynamic regions, reflecting scene changes while largely disregarding static backgrounds like grass. However, for frame-to-frame transitions within a clip, neurons unexpectedly redistribute their contributions even in stationary regions, suggesting an internal representation that does not strictly adhere to temporal consistency. This behavior indicates that

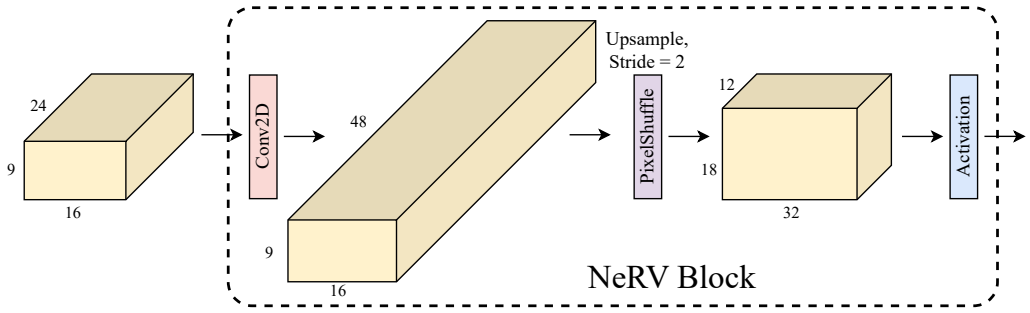

Figure 35. We show an example NeRV block with upsample stride $s = 2$ and channel reduction $r = 2$, for the features generated by the positional encoding and stem in Figure 34. See Appendix 8 for more details.

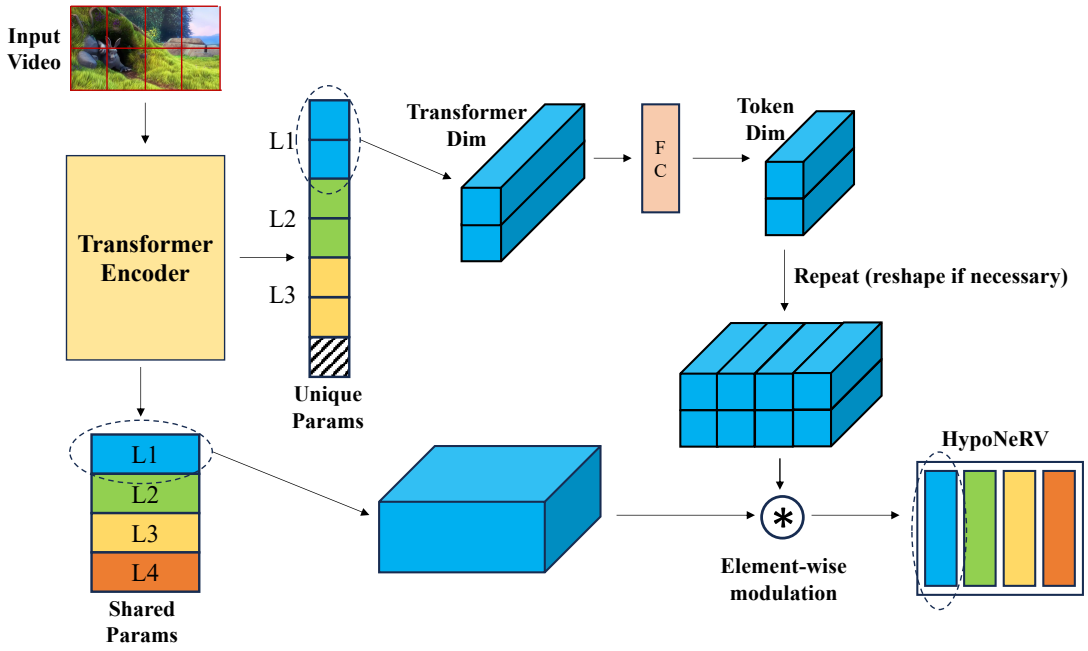

Figure 36. We show example token predictions, FC processing, repeat/reshape, and modulation with the "shared" parameters for a single layer of the hypo-network. The unique parameters (the predictions of the weight token FC layer, just before repeat/reshape), are what we store, quantize, encode, and measure for bpp.

while the model effectively captures broader scene dynamics, its learned representations may not maintain stable spatial allocations for finer-grained motion and preserve local temporal coherence.

10.3. Supplementary Results on NeRV variants

Figure 40 and Figure 41 supplement Figure 11 and Figure 12 respectively to extend the analysis to additional contribution maps and motion analysis results for NeRV, HNeRV and E-NeRV. NeRV has more structured changes driven by motion, whereas the weaker E-NeRV model demonstrates less pronounced adaptation. These findings further highlight the differences in representational capacity among the NeRV variants.

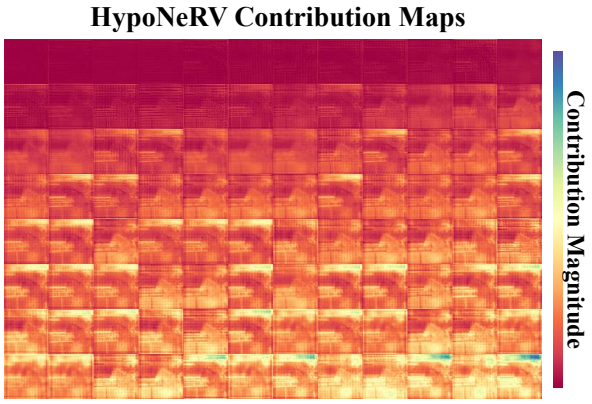

Figure 37. **XINC contribution maps** on the last (head) layer of HypoNeRV, for Jockey at 256×256 resolution. We sort kernels by total contribution magnitude and select a subset of uniformly sampled kernels in the head layer. For the sake of interpretability, we remove locations at which kernel contributions are rendered zero due to the strided PixelShuffle following the convolutional layer. HypoNeRV kernels exhibit the full spectrum of contribution magnitudes for various parts of the scene, ranging from low (dark red) to high (blue/purple).

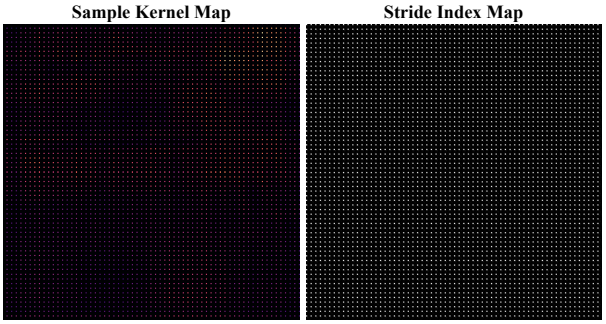

Figure 38. **Visualization of PixelShuffle**'s effect on kernel contribution patterns in the last (head) layer of HypoNeRV, for Jockey at 256x256. Left: Contribution map for a single kernel, showing the sparse activation pattern created by PixelShuffle's channel-to-space rearrangement. Black regions indicate locations where the kernel's contributions are zeroed out. Right: Reference map showing the potential contribution locations (white pixels) available to a kernel within a PixelShuffle group. Best viewed at full scale, otherwise the reader's PDF viewer may perform interpolation and render strange artifacts and make it appear as if the values on the right hand plot are not uniform. Similar distortions can occur for the plot on the left.

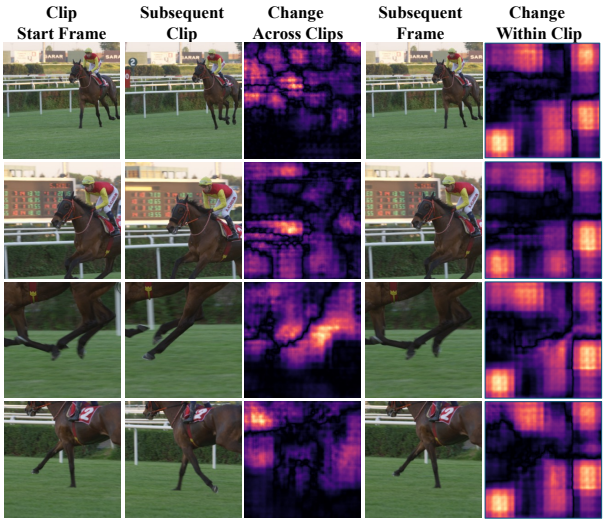

Figure 39. XINC motion analysis for fluctuation in total contribution of HypoNeRV neurons corresponding to two types of transitions: between first frames of consecutive clips (middle column) and between consecutive frames within the same clip (rightmost column). For transitions between clips, neurons rearrange their contributions in broader regions around areas of motion while ignoring static elements such as grass. For smaller motions between consecutive frames within a clip, neurons unexpectedly rearrange their contributions in static regions, showing little temporal faithfulness in their inner representation.

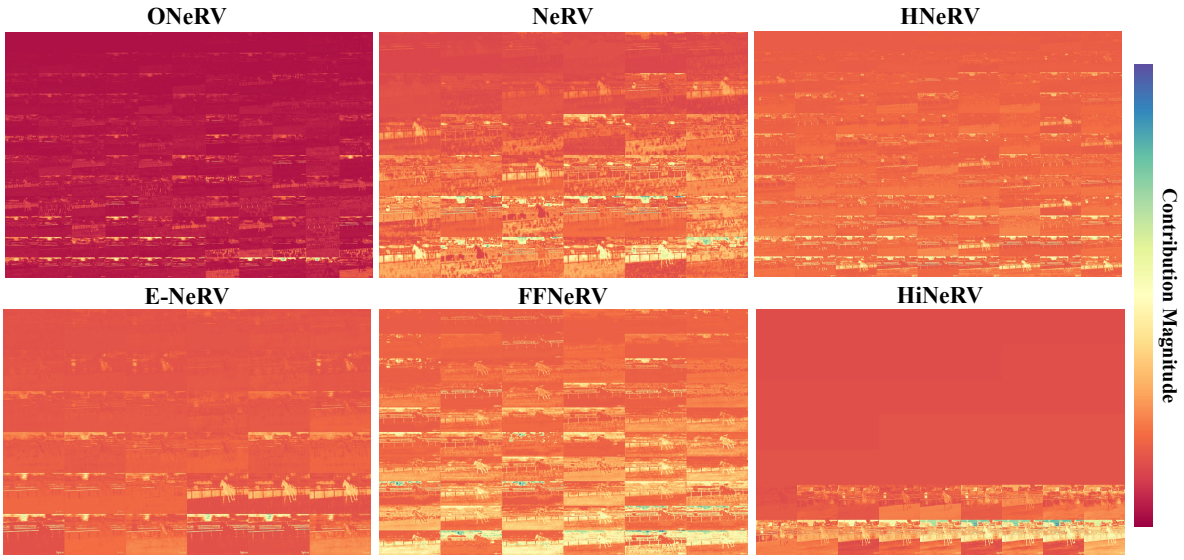

Figure 40. **XINC contribution maps** from the last (head) layer on Jockey at 1080p. We supplement Figure 11 by showing kernel contribution maps sorted by magnitude for additional NeRV variants - NeRV, HNeRV and E-NeRV.

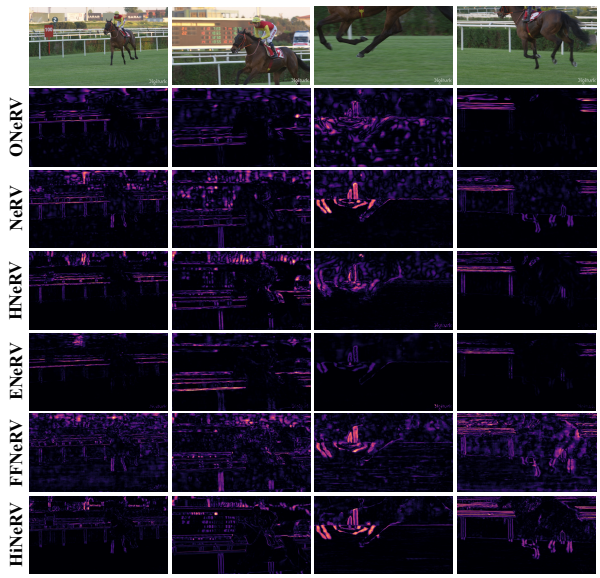

Figure 41. XINC motion analysis for the last (head) layer of all different NeRV variants, for Jockey at 1080p. We supplement Figure 12 by visualizing the fluctuation in total contribution for kernels of additional NeRV variants - NeRV, HNeRV and E-NeRV.