Multimedia Tools and Applications (2026) 85:681
https://doi.org/10.1007/s11042-026-21822-5

A survey of implicit neural representations for video compression

Hannes Keunen[1] · **Maarten Wijnants**[1] · **Jori Liesenborgs**[1]

Received: 30 June 2025 / Revised: 19 June 2026 / Accepted: 19 July 2026

Abstract
Neural Video Representations (NVRs) have recently been proposed as a novel approach to the video compression problem. NVRs consist of one or multiple small neural networks that are overfitted on one specific video sequence, thereby encoding the video within the weights and biases of the network(s). In contrast to other learned video coding approaches, NVR-based codecs do not rely on large datasets and can achieve lower decoding complexity by using compact, video-specific models instead of large shared encoder-decoder architectures. Many works have focused on improving the compression performance of NVR-based codecs by enhancing the overall codec design, devising more performant and parameter-efficient model architectures, and incorporating more advanced model compression schemes such as weight pruning, quantization, and entropy minimization. We provide a systematic overview of representative work in the field and discuss common weaknesses and opportunities for future work, with a focus on practical deployment for video streaming. This paper serves as both an introduction for newcomers and a reference for existing researchers, highlighting the potential of neural video representations as an alternative to traditional codecs in video compression.

Keywords Implicit neural representations (INR) · Learned video compression · Model compression · Neural video representations (NVR) · Video streaming

✉ Hannes Keunen
hannes.keunen@uhasselt.be

Maarten Wijnants
maarten.wijnants@uhasselt.be

Jori Liesenborgs
jori.liesenborgs@uhasselt.be

[1] Digital Future Lab, UHasselt - Flanders Make, Diepenbeek, Belgium

Published online: 04 August 2026

1 Introduction

With video streaming accounting for more than 65% of global internet traffic in 2024,[1] the demand for better video codecs is growing rapidly. Traditional video codecs, such as H.266/VVC [1], still rely on hand-crafted features to exploit redundancies within and between frames. Although H.266/VVC achieves approximately 50% bit rate reduction over its predecessor, H.265/HEVC [2], the computational complexity for encoding and decoding also increases dramatically with each new generation of video codecs [3].

Recent advances in machine learning have been pushing towards the use of deep learning techniques for video compression, either by incorporating deep learning models in the traditional video compression pipeline, or by completely replacing the traditional pipeline by one that is trained end-to-end. DVC [4] first achieved this by replacing each component of the traditional video compression pipeline with components that can be optimized jointly. Subsequent work has expanded upon DVC, surpassing HEVC in terms of rate-distortion performance [5, 6].

The weakness of end-to-end trained neural compression methods, such as DVC, lies in the fact that they are general-purpose and thus have to perform well on a wide range of videos with heterogeneous characteristics (e.g., live action versus animation, low versus high camera motion, low versus high spatial complexity, and so on). To achieve this, they usually require training on large datasets. This also results in high parameter counts, and thus high decoding complexity and energy usage, making them impractical for deployment, especially on mobile devices.

Implicit neural representations (INRs) represent an alternative approach to video compression. Instead of fitting a large model to an extensive dataset, a much smaller model can be overfitted to memorize a single video. This allows a video to be encoded in the parameters of the network, eliminating the dependence on large datasets and significantly decreasing the decoding complexity. The INR paradigm was first introduced to represent spatial data as a continuous function $f_{\boldsymbol{\theta}} : \mathbb{R}^{\boldsymbol{I}} \rightarrow \mathbb{R}^{\boldsymbol{O}}$ [7–9]. This function can be approximated by a neural network parametrized by $\boldsymbol{\theta}$. Several works [10–12] have adapted the INR paradigm to image data by mapping spatial image coordinates to pixel intensities. SIREN [12] first extended this approach to video data by naively generating each pixel individually, indexed by its spatiotemporal coordinate (t, x, y). NeRV [13] was the first INR that specifically targeted video data, using convolution and upsampling operations to generate entire video frames from a temporal coordinate t. This inspired a new line of research on neural video representations (NVRs), improving NeRV for tasks such as video inpainting, frame interpolation, and video compression. The primary focus of this paper will be on the video compression task.

In the context of NVR, a video $\boldsymbol{V} \in \mathbb{R}^{T \times W \times H \times C}$, with T frames of size $W \times H$ and C channels per pixel, will be formalized as a set of N functions

$$\mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta}) = \{D_n(E_n(\boldsymbol{i}; \boldsymbol{\theta}^n_{enc}); \boldsymbol{\theta}^n_{dec})\}^N_{n=1}, \tag{1}$$

$$\boldsymbol{\Theta} = \{\boldsymbol{\theta}_n\}^N_{n=1} = \{\boldsymbol{\theta}^n_{enc}, \boldsymbol{\theta}^n_{dec}\}^N_{n=1} \tag{2}$$

[1] https://www.applogicnetworks.com/phenomena

where $\boldsymbol{i}$ represents some coordinate into $\boldsymbol{V}$, E_n is a function that generates an input embedding $\boldsymbol{e}_n$ for the region of the video indexed by $\boldsymbol{i}$, and D_n decodes the corresponding video data from that embedding. As we will explain in Section 2.2, each of the N functions in $\mathcal{F}$ encodes some part of $\boldsymbol{V}$ in its parameters $\boldsymbol{\theta}_n$. Each function D_n is approximated by a neural network parametrized by $\boldsymbol{\theta}^n_{dec}$. The embedding function E_n may also be in the form of a neural network parameterized by $\boldsymbol{\theta}^n_{enc}$, but can also be a handcrafted embedding. Section 3.4 contains an overview of the different forms of embedding functions. The complete set of parameters is denoted as $\boldsymbol{\Theta}$.

The contributions of this paper are as follows:

- We provide an overview of the existing techniques that can be used for designing approximators for (1) and their respective impact on reconstruction quality and compression performance.
- We discuss how an NVR-based video codec might be employed in the context of video streaming.
- Based on this practical scenario, we identify common challenges related to NVRs and provide opportunities for future work.

Gwilliam et al. [14] also provide a technical comparison of some of the NVRs discussed in this paper. However, this paper provides a more comprehensive taxonomy of existing work and focuses more on the practical implications and challenges for video compression and streaming. Huo et al. [15] provide a comprehensive overview of the evolution of block-based hybrid video compression towards autoencoder-based compression models. Jia et al. [16] provide a more recent overview of autoencoder-based learned video compression methods. Finally, Essakine et al. [17] provide a comprehensive review of implicit neural representations, but do not mention video-specific methods.

Section 2 first provides an overview of existing works by comparing different high-level design considerations for NVR-based video codecs. Section 3 discusses low-level design choices for NVR models. Section 4 discusses several model compression techniques used in NVRs. Combined, Section 2 to 4 organize existing approaches along multiple complementary axes, including modeling assumptions (e.g., instance-based vs. hybrid methods), architectural design choices (e.g., upsampling strategies, temporal modeling), and compression methods (e.g. pruning, quantization, entropy minimization), which results in a multi-dimensional NVR taxonomy. To identify common weaknesses and opportunities for future work, Section 5 finally discusses how the NVR paradigm might be employed in the practical context of video streaming.

2 High-level codec design

We first discuss several high-level design considerations for NVRs. Section 2.1 discusses the effect of *instance size*, or the amount of data generated in each forward pass. Section 2.2 discusses *model scope*, or the amount of data captured by each set of model parameters $\boldsymbol{\theta}_n$. Section 2.3 discusses the different types of *input embedding* that can be used as input to D_n. Finally, Section 2.4 discusses several ways in which temporal information can be exploited. An overview of all the works discussed in this section is provided in Table 1, along with

Table 1 Overview of neural video representation methods with an indication of the reconstruction quality achieved on the UVG [18] dataset

Year	Name	Instance	Scope	Input	Temporal context	Avg. PSNR
2021	SIREN [12]	Pixel	Full	/	Implicit	
	NeRV [13]	Frame	Full	PE	Implicit	31.11
2022	E-NeRV [20]	Frame	Full	PE, Grid	Implicit	31.51
	CNeRV [21]	Frame	Full	IE	Implicit	
	PS-NeRV [22]	Patch	Full	PE	Implicit	32.02
2023	NIRVANA [23]	3D Patch	GoP	PE	Conv3D	
	D-NeRV [24]	Frame	Full	PE	ME/MC	
	HNeRV [25]	Frame	Full	IE	Implicit	32.47
	DNeRV [19]	Frame	Full	IE, TE	Encoder	31.94
	FFNeRV [26]	Frame	Full	MTG	ME/MC	32.48
	Gomes et al. [27]	Frame	Full	Grid	Implicit	31.52
	C3 [28]	3D Patch	Instance	IE	Conv3D	
2024	COOL-CHIC video [29]	Frame	Instance	IE	ME/MC	
	HiNeRV [30]	Patch	Full	MTG	Implicit	35.27
	NeRV++ [31]	Frame	Full	PE	Implicit	32.35
	HNeRV-BOOST [32]	Frame	Full	IE	Implicit	33.89
	VQ-Nerv [33]	Frame	Full	IE	Implicit	**35.41**
	T-NeRV [34]	Frame	Full	PE, MTG, IE	Implicit	34.00
	NVRC [35]	Frame	Full	MTG	Implicit	
	TNeRV [36]	Frame	Full	IE, TE	Encoder	
	QS-NeRV [37]	Frame	Layer	IE	Implicit	33.63
	SNeRV [38]	Frame	Full	IE, TE	Implicit	<u>35.34</u>
	HFS-HNeRV [39]	Frame	Full	IE	Implicit	
2025	PNVC [40]	Frame	Instance	IE	ME/MC	

Note: The final column contains the same results as the final column in Table 2

an indication of the average reported reconstruction quality, where available. Table 2 provides a more detailed comparison of reconstruction quality reported in the literature. To improve comparability, only results obtained on the UVG dataset [18] with approximately 3M parameters and 300 training epochs are included. Nevertheless, since experiments have been performed by different researchers independently of one another, potentially with slightly different experimental settings, this only serves as an indication rather than a definitive performance evaluation. Although most works report results in PSNR, and in some cases also (MS-)SSIM, it is also important to note that these are signal fidelity metrics that do not always correspond well to the quality perceived by the human visual system. To the best of the authors' knowledge, only DNeRV [19] reports VMAF scores in addition to PSNR and SSIM. Also note that Table 2 only shows reconstruction quality, as compression performance will be considered in Section 4.

2.1 Instance size

Instance size is the amount of data generated in a single forward pass of D_n. The format of the input coordinate $\boldsymbol{i}$ is determined by the size and shape of each instance.

Table 2 Indicative PSNR comparison on the UVG (1080p) dataset under approximately matched model size (3M parameters) and training budget (300 epochs)

Year	Model	Beauty	Bosphorus	Honey	Jockey	Ready	Shake	Yacht	Avg.
2021	NeRV [13]	33.14	32.74	37.18	30.99	23.97	33.06	31.11	31.11
2022	E-NeRV [20]	33.29	33.87	38.88	28.73	23.98	34.45	31.51	31.51
	PS-NeRV [22]	32.94	32.32	38.39	30.38	23.61	33.26	26.33	32.03
2023	HNeRV [25]	33.36	33.62	39.17	32.31	25.60	34.90	28.33	32.47
	DNeRV [19]	33.91	33.17	38.22	32.01	25.56	33.58	27.15	31.94
	FFNeRV [26]	33.62	33.72	38.82	32.80	25.82	34.19	28.37	32.48
	Gomes et al. [27]	33.46	33.76	38.70	28.89	23.88	34.32	27.66	31.52
2024	HiNeRV [30]	34.08	**38.68**	39.71	36.10	31.53	35.85	30.95	35.27
	NeRV++ [31]	33.57	34.59	38.76	32.46	25.29	34.11	27.65	32.35
	NeRV-BOOST [32]	33.55	34.51	39.04	32.82	26.08	34.54	28.76	32.76
	E-NeRV-BOOST [32]	33.75	35.62	39.61	32.39	27.75	35.48	29.23	33.40
	HNeRV-BOOST [32]	33.80	36.11	39.65	34.28	28.19	35.88	29.33	33.89
	VQ-NeRV [33]	34.15	36.89	39.41	**36.77**	31.74	**36.08**	**32.85**	**35.41**
	T-NeRV [34]	**34.34**	35.65	**40.26**	34.80	28.24	35.25	29.46	34.00
	QS-NeRV [37]	33.92	35.74	39.27	33.83	27.67	35.04	29.97	33.63
	SNeRV [38]	34.14	37.64	39.11	36.17	**33.21**	34.41	32.71	35.34

Note: The best result for each video is marked in bold. The second-best result for each video is underlined. Results are obtained from the respective publications. Despite matching dataset, model size, and training budget, differences in implementation and evaluation settings may still affect comparability

2.1.1 Pixel

Neural representations for images [12, 41] usually generate one pixel at a time, indexed by spatial coordinates (x, y). SIREN [12] extends this pixel-wise approach to videos, where each individual pixel of $\boldsymbol{V}$ is indexed by its spatiotemporal coordinate $\boldsymbol{i} = (t, x, y)$ where $t < T$, $x < W$, and $y < H$. Because pixel-wise methods do not explicitly exploit spatial redundancies, this leads to inferior results compared to larger instance sizes.

2.1.2 Frame

NeRV [13] showed that switching to a frame-wise approach with convolutional neural networks substantially reduces encoding and decoding complexity, and improves reconstruction quality. Frames are indexed by a single temporal coordinate $\boldsymbol{i} = t, t < T$. Many later works [19–21, 24–27, 29, 31–34, 36–40] followed the same frame-wise approach.

2.1.3 Patch

As a more generalized formulation, $\boldsymbol{V}$ can be subdivided into patches of size $T_p \times W_p \times H_p$, indexed by patch coordinate $\boldsymbol{i} = (t, x, y)$ where $t < \frac{T}{T_p}$, $x < \frac{W}{W_p}$, and $y < \frac{H}{H_p}$. As such, the pixel- and frame-wise approaches can both be seen as special cases of the patch-wise approach, where the pixel-wise approach uses a patch size of $1 \times 1 \times 1$ and the frame-wise approach uses a patch size of $1 \times W \times H$. A spatial patch-wise approach ($T_p = 1$) was first introduced by PS-NeRV [22] to better capture high-frequency detail, while still preserving the improvements in computational complexity offered by frame-wise methods over pixel-wise methods. This results in an average increase of 0.92dB in PSNR w.r.t. NeRV [13] on

the UVG dataset [18], as can be seen in Table 2. A second advantage of this approach is that the architecture of the model does not have to change when adapting to different spatial resolutions [23]. HiNeRV [30] additionally uses overlapping patches to avoid spatial inconsistencies between neighboring patches. NIRVANA [23] and C3 [28] use three-dimensional spatiotemporal patches to also exploit temporal redundancies within each patch.

2.2 Model scope

Model scope refers to the total amount of data captured by each $\boldsymbol{\theta}_n$. The scope of each individual model is therefore inversely proportional to the number of models N used to encode $\boldsymbol{V}$. As Section 5 will discuss in more detail, a larger scope results in higher encoding latency because the entire video segment must be encoded before anything can be transmitted to a decoder.

2.2.1 Full scope

Most works [13, 19–22, 24–27, 30–34, 36, 38, 39] do not explicitly limit the scope, but instead encode the entire input video with a single model. D-NeRV[2] [24] is specifically aimed at encoding longer and more diverse videos, by conditioning on keyframes.

2.2.2 GoP scope

NIRVANA [23], on the other hand, trains a separate network per Group of Pictures (GoP). In this way, each subsequent set of parameters $\boldsymbol{\theta}_n$ can be fine-tuned from $\boldsymbol{\theta}_{n-1}$, which reduces training time.

2.2.3 Instance scope

At the extreme end of the spectrum, COOL-CHIC video [29] trains a separate model for each frame. This results in an extremely low decoding complexity of only 900 multiplications per decoded pixel, and a lower decoding delay since each frame is compressed individually. When combined with a single-frame instance size, the downside of this approach is that temporal redundancies are no longer implicitly captured by the model, but must be exploited through some other means. COOL-CHIC video [29] and PNVC [40] are based on a more traditional method by generating I-, P-, and B-frames. Similarly to NIRVANA [23], PNVC additionally fine-tunes the parameters for each frame from those of the previous frame to reduce encoding complexity even further. C3 [28] uses a single-instance scope in combination with three-dimensional spatiotemporal patches.

2.2.4 Layer scope

QS-NeRV [37] takes a layered approach inspired by scalable video coding [42]. A base layer network based on HNeRV [25] is first used to encode a low-quality version of $\boldsymbol{V}$. Several enhancement layer networks then use feature maps from the previous layer to decode a residual that can be added to the output of the previous layer.

[2] Note the distinction between D-NeRV [24] and DNeRV [19], and between T-NeRV [34] and TNeRV [36]

2.3 Input embedding

Instead of decoding each instance from the input coordinate $\boldsymbol{i}$ directly, the embedding function E_n from (1) first converts $\boldsymbol{i}$ to some low-dimensional embedding $\boldsymbol{e}_n$ that is used as input to D_n, as depicted in Fig. 1. The resulting embedding is usually in the form of a feature grid with a low spatial dimension but a high number of channels per pixel, which can then be passed to a fully convolutional decoder D_n.

2.3.1 Positional encoding (PE)

In the original approach proposed by NeRV [13], frames are generated purely based on a frame index interpolated between 0 and 1. This frame index is first encoded using positional encoding [43]

$$\mathrm{PE}\ (t) = \left(\sin\left(b^0\pi t\right), \cos\left(b^0\pi t\right), ..., \sin\left(b^{l-1}\pi t\right), \cos\left(b^{l-1}\pi t\right)\right), \tag{3}$$

where l is the length of the embedding and b is a constant factor, commonly set to 1.25 . Mapping the input to a higher-dimensional space with high-frequency functions allows the network to better learn high-frequency information than when the frame index is taken directly as input [13, 43–45]. A multilayer perceptron (MLP) is then used to transform the input embedding to a feature grid as input to subsequent convolutional and upsampling layers. This form of input embedding was also adopted by PS-NeRV [22], D-NeRV [24] and NeRV++ [31].

2.3.2 Spatial grid

A common critique on the combination of PE and MLP described above is that the MLP contains a disproportionate number of parameters compared to the rest of the network [20, 27]. E-NeRV [20] uses a much smaller MLP to generate a one-dimensional temporal feature vector $\boldsymbol{e}_{n,t} \in \mathbb{R}^d$, combined with a grid of spatial coordinates that is also encoded with PE into a spatial feature grid $\boldsymbol{e}_{n,s} \in \mathbb{R}^{w \times h \times d}$, with $w < W$ and $h < H$. The spatial and temporal features are then combined through per-feature element-wise multiplication. The resulting feature grid can be passed to D_n directly. Gomes et al. [27] bypass the need for MLP layers in NeRV completely by generating a grid based on spatial and temporal coordinates and then applying PE to each element of that grid. Although a grid-based embedding reduces or bypasses the reliance on an expensive MLP, this is still a content-agnostic embedding, meaning that content-specific features are encoded solely by the network layers.

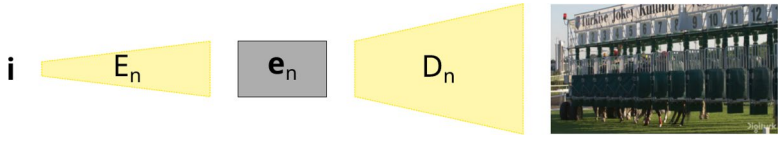

Fig. 1 Conventional NVR architecture, where instances are generated from an input coordinate

2.3.3 Multi-resolution temporal grid (MTG)

FFNeRV [26] and HiNeRV [30] use K learned feature grids $\{\boldsymbol{G}_k \in \mathbb{R}^{T_k \times W_g \times H_g \times C_g}\}_{k=1}^{K}$ with different temporal resolutions T_k, spatial resolution $W_g \times H_g$ and C_g features. Given a temporal coordinate t, a temporal feature is interpolated from each feature grid, and the results are concatenated along the channel dimension. The concatenated temporal features then serve as input to D_n. Due to interpolation, each of the T_k slices of the MTG affects several adjacent video frames if $T_k < T$. In other words, this type of embedding inherently models temporal redundancies [34]. Furthermore, treating the MTG like a set of learnable parameters makes it more powerful for learning spatial and temporal features than a fixed grid. Table 1 also indicates that methods that use the MTG embedding generally perform better than methods that use PE.

2.3.4 Instance embedding (IE)

CNeRV [21] and HNeRV [25] formulate E_n and D_n as encoder and decoder components of an autoencoder, as depicted in Fig. 2. When both parts are trained jointly, this essentially allows the network to learn the best input embedding autonomously, instead of using a hand-crafted one. This results in a hybrid between autoencoder-based compression models such as DVC [4], and INR-based methods. Since $\boldsymbol{e}_n$ is generated from the actual video, it cannot be generated during the decoding process. This means that each IE has to be encoded in the bitstream along with the model weights, resulting in a trade-off between the two. By using a learned embedding, HNeRV[25] achieves an average increase of 1.36dB in PSNR w.r.t. NeRV [13] on the UVG dataset [18], as can be seen in Table 2. More generally, the results in Table 2 indicate a trend toward higher reconstruction quality for methods that employ IE compared to those that use content-agnostic embeddings. DNeRV^2 [19], QS-NeRV [37], TNeRV^2 [36], HFS-HNeRV [39], and SNeRV [38] also follow this hybrid approach. VQ-NeRV [33] and PNVC [40] further extend the hybrid architecture similarly to U-shaped networks [46], by producing instance embeddings between each downsampling stage and its corresponding upsampling stage.

IE can be used alongside other embedding types to introduce more spatial or temporal context. HNeRV-BOOST [32] improves HNeRV [25] by conditioning on a positionally encoded temporal index in each upsampling block, as we will explain in more detail in Section 3.3. TNeRV [34] combines frame embeddings with the MTG embedding introduced by FFNeRV [26], based on the earlier observation that the MTG can model temporal redundancies, and also conditions each upsampling block on a positionally encoded temporal index.

2.4 Temporal context

Traditional video codecs employ motion estimation and motion compensation techniques to explicitly model temporal redundancies in an input video. Although INRs are able to encode

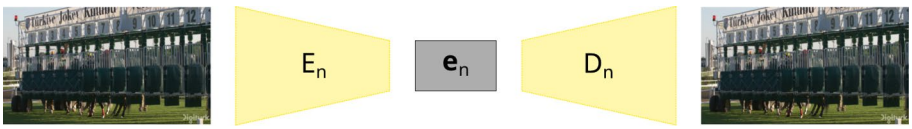

Fig. 2 Hybrid NVR architecture that combines INR- and autoencoder-based methods

redundancies into the network's parameters implicitly, some works explicitly exploit temporal information to achieve higher temporal consistency in the decoded output.

2.4.1 Motion estimation and compensation (ME/MC)

Some NVRs still perform ME/MC explicitly. D-NeRV [24] performs ME/MC with features extracted from key frames at multiple scales. As mentioned in Section 2.2, this technique allows the model to perform significantly better on longer and more diverse videos. FFNeRV [26] and T-NeRV^2 [34] generate independent reference frames, along with flow maps and interpolation weights, which are used to warp and interpolate features from adjacent independent frames into a final reconstructed frame. While this has been shown to perform better on videos with highly dynamic content, it also adds some computational overhead [14]. Similarly, COOL-CHIC video [29] generates flow maps, interpolation weights, and residuals that are fed to an inter coding module.

2.4.2 Temporal embedding (TE)

Some hybrid NVRs use adjacent frames to generate temporal embeddings alongside the instance embedding. DNeRV [19] encodes the differences between the current frame and the previous and next frames. Similarly, TNeRV [36] introduces a temporal diversity exploration module with 3D convolutions in the encoder to extract temporal information. SNeRV [38] uses embeddings from adjacent frames during the upsampling stages.

3 Low-level network design

The NeRV [13] decoder consists of five NeRV blocks that gradually upsample the input to obtain a final reconstructed frame. Each NeRV block uses a 3×3 subpixel convolution, consisting of a convolutional and a pixelshuffle layer [47], to scale the spatial resolution of the input feature map by a factor S. The first block increases the number of channels by some expansion factor, while subsequent blocks reduce the number of channels by a factor of 2. Each block is followed by a GELU activation function. Common upsampling factors are $(5, 2, 2, 2, 2)$ for 720p resolution and $(5, 3, 2, 2, 2)$ for 1080p resolution. Specifically, for an initial channel dimension C_{in} and a target channel dimension C_{out}, the NeRV block is shown in Fig. 3(a). The complete decoder architecture for 1080p resolution is shown in Fig. 4.

There are two common concerns with this architecture. First, the upsampling blocks have an uneven and inefficient distribution of parameters [20, 25, 26, 30]. Section 3.1 discusses how the decoder blocks can be designed to address this issue. As a general rule, when a large number of parameters can be eliminated at the cost of a slight decrease in performance, the saved parameter budget can be used to increase the channel dimension of each intermediate feature map, thereby increasing the representation capacity of the network as a whole. This is supported by Fig. 5, which shows that simply increasing the number of parameters in the same model significantly improves the reconstruction quality.

The second common concern with this architecture is that it often fails to preserve high-frequency details as a result of convolution operations [38, 39], causing a spectral bias

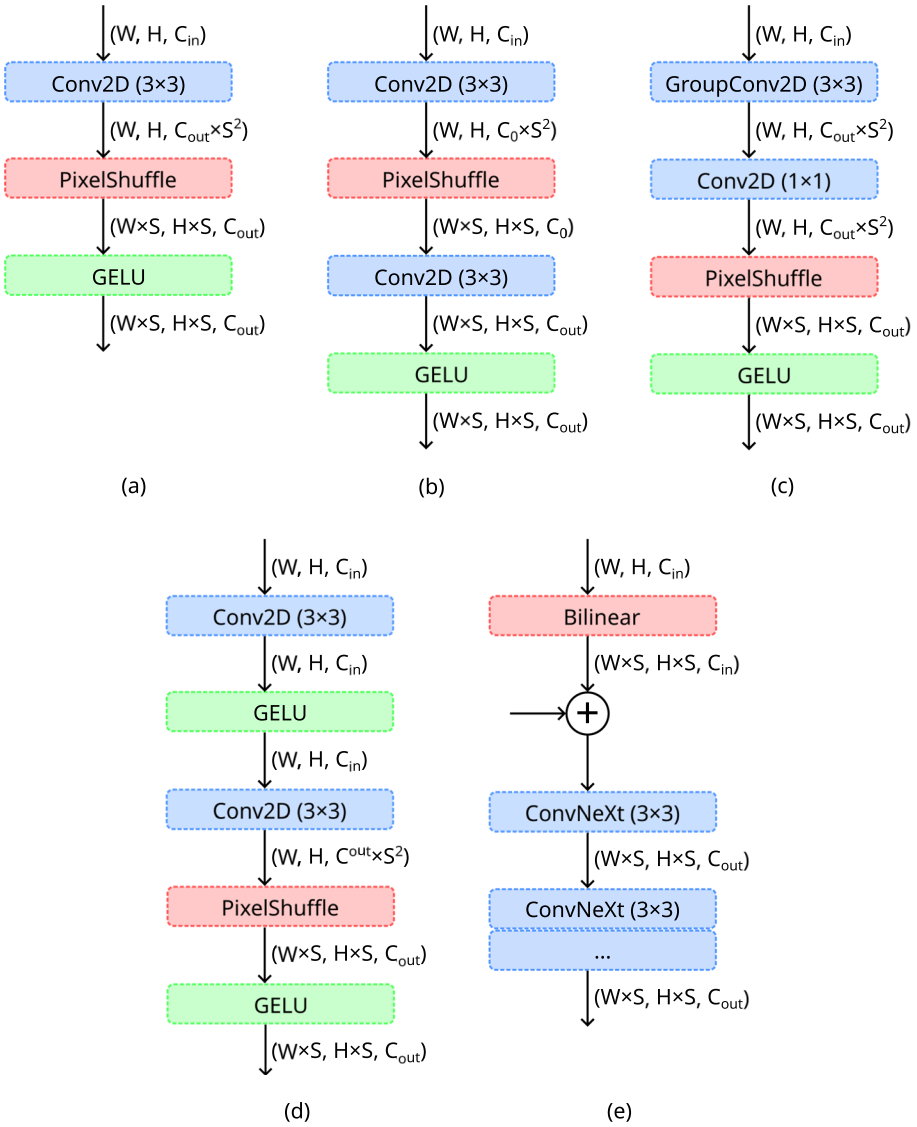

Fig. 3 Upsampling block architectures (**a**) NeRV block (**b**) First upsampling block in E-NeRV, with $C_0 = \min(C_{in}, C_{out})/4$ (**c**) FFNeRV block in stages 2-5 (**d**) T-NeRV block in stages 3-5 (**e**) HiNeRV block

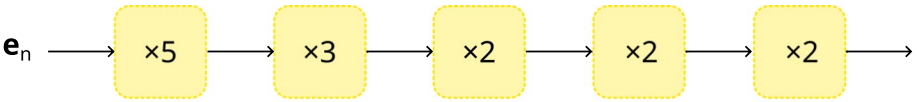

Fig. 4 NeRV decoder architecture for 1080p resolution

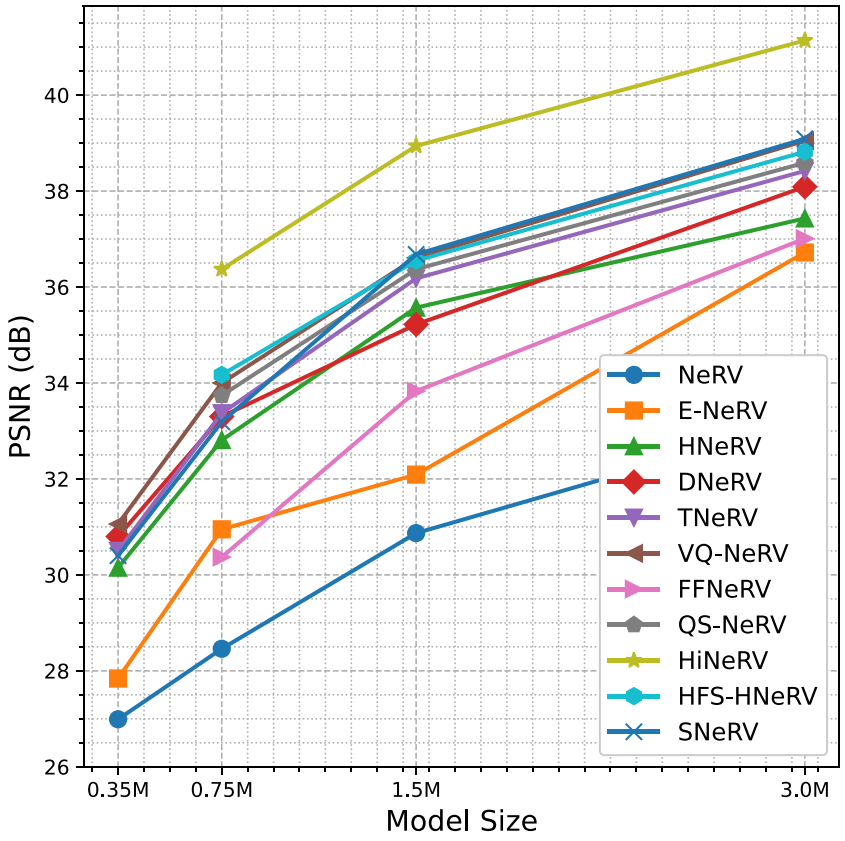

Fig. 5 PSNR results on Big Buck Bunny (132 frames) with different model sizes (in millions of parameters), as obtained from the respective papers

towards lower frequency components. In fact, the spectral bias issue is not exclusive to NVR, but also occurs in other applications of neural networks [44]. Section 3.2 explains how the spectral bias issue can be alleviated by adding inductive biases towards high-frequency information to the decoder. Section 3.3 then explores several ways to incorporate temporal information in the upsampled feature maps. For hybrid approaches, the encoder architecture also plays a role in compression performance. This is discussed in Section 3.4. Finally, Section 3.5 discusses several loss functions that are used for training NVRs, and how these can also be used towards solving the spectral bias issue.

3.1 Upsampling architecture

The convolutional layer in each NeRV block contains $C_{in} \times S^2 \times C_{out}$ kernels of size $k \times k$, resulting in a total of $k^2 \times C_{in} \times S^2 \times C_{out}$ parameters. This is mostly problematic in the first upsampling block, where $S = 5$. E-NeRV [20] improves this by introducing an intermediate channel dimension $C_0 = \min(C_{in}, C_{out})/4$ in the first upsampling block. The improved block structure is shown in Fig. 3(b). This results in two convolutional layers with $k^2 \times C_{in} \times S^2 \times C_0$ parameters and $k^2 \times C_0 \times C_{out}$ parameters, respectively, or

a total parameter count of $k^2 \times C_0 \times (C_{in} \times S^2 + C_{out})$ for the first upsampling block. This amounts to at least four times fewer parameters than the NeRV block if $C_{in} \leq C_{out}$. A detailed calculation of this ratio can be found in [20].

FFNeRV [26] follows yet another approach by substituting the convolutional layer for a depthwise separable convolution [48], composed of a grouped convolution followed by a pointwise convolution, as shown in Fig. 3(c). With g channel-wise groups that are processed separately, the first convolutional layer has a parameter count of $k^2 \times \frac{C_{in} \times S^2 \times C_{out}}{g}$. The point-wise convolution, equivalent to a pixel-wise linear layer, then contains $S^4 \times C_{out}^2$ parameters. Together, this results in a ratio of $\frac{1}{g} + \frac{S^2 \times C_{out}}{k^2 \times C_{in}}$ with respect to the NeRV block. From this ratio, we can see that when $S > k$ and $C_{out} \geq C_{in}$, the depthwise separable convolution actually contains more parameters than a regular convolution. Therefore, this structure is only used in the second upsampling block and onward, where $S \leq k$ and $C_{out} \leq C_{in}$.

HNeRV [25] does not change the architecture of the upsampling blocks, but instead varies the kernel size from 1 in the first upsampling stage, to 3 in stage 2, to 5 in stages 3 and onward. Furthermore, the channel dimension is reduced by a factor of 1.2 in each layer, instead of 2 as in the original NeRV [13] block. This combination of larger convolution kernels and a smaller channel dimension scale factor places more importance in the later stages of the network, allowing for a more even distribution of parameters across the network. TNeRV [36] uses the same upsampling block architecture as HNeRV. T-NeRV [34] also uses a similar block architecture, but the 5×5 convolutions are replaced by two 3×3 convolutions and an additional activation layer, as in Fig. 3(d). This significantly reduces the number of parameters, while still maintaining the same spatial range as the original 5×5 convolution.

Instead of the parameter-heavy subpixel convolution, HiNeRV [30] uses a much simpler bilinear interpolation for upsampling. It then uses the saved parameter budget in a more powerful architecture based on the ConvNeXt [49] block. A local feature grid, similar to the multiresolution temporal embeddings, is first added to the upsampled feature map to preserve high-frequency details that would otherwise be lost due to interpolation. The result is then passed through a series of ConvNeXt blocks. The activation layer at the end of the block is omitted since each ConvNeXt block already contains an activation layer. This architecture, which is illustrated in Fig. 3(e), makes HiNeRV significantly more powerful than its predecessors that rely on subpixel convolution, but also increases the computational cost. In fact, Gwilliam et al. [14] show that HiNeRV, despite its higher representation capacity, performs significantly worse than other NVR models at shorter training times. Gwilliam et al. therefore synthesise a model that optimizes the trade-off between encoding time and reconstruction quality by combining components from existing models.

More recently, PNVC [40] has introduced weight reparameterization and knowledge distillation schemes in its upsampling blocks to reduce decoding complexity. Weight reparameterization was originally introduced in [50], where complex layers are substituted with simpler operations during inference. Weight reparameterization is implemented by decomposing a single fully connected layer into several parallel stacks of fully connected layers. Similar to weight reparameterization, knowledge distillation schemes train a small student model to mimic a larger teacher model [51]. This is implemented in PNVC with a mask decaying strategy [52, 53], where a mask $\boldsymbol{M}$ is initialized as $\mathbf{1}$ and gradually decayed

towards $\mathbf{0}$ during training. This can be interpreted as an interpolation weight that gradually pushes more importance towards the student model. A detailed description of the methods used by PNVC can be found in [40].

3.2 Preservation of high-frequency components

In order to address the issue of spectral bias mentioned in the introduction of this section, HFS-HNeRV [39] extends the HNeRV [25] block with a high-frequency spectrum convolution module that extracts more high-frequency features. Feature maps are first transformed with the discrete wavelet transform (DWT). The DWT decomposes a feature map into four sets of wavelet coefficients $\boldsymbol{C}_{LL}$, $\boldsymbol{C}_{LH}$, $\boldsymbol{C}_{HL}$, and $\boldsymbol{C}_{HH}$, with low-frequency components, and horizontal, vertical, and diagonal high-frequency features, respectively. The input feature maps are then augmented with frequency-domain features extracted from the wavelet coefficients.

In contrast, SNeRV [38] uses a small network to extract the $\boldsymbol{C}_{LH}$, $\boldsymbol{C}_{HL}$ and $\boldsymbol{C}_{HH}$ components after the final upsampling stage, and then performs an inverse DWT to restore the final frame. The final upsampled feature map serves as the $\boldsymbol{C}_{LL}$ component in the IDWT, and also as input to extract the aforementioned high-frequency components. Table 2 shows that on average, SNeRV achieves an increase of almost 3dB in PSNR w.r.t. HNeRV, on which it is based.

3.3 Temporal information fusion

Note from Section 3.1 that the upsampling operations used by NeRV [13] exploit spatial information exclusively, whereas temporal information is only introduced via input embedding. Further research has shown that the network's performance can be improved by fusing temporal information in the decoder. Inspired by style transfer networks, E-NeRV [20] and T-NeRV^2 [34] both use an AdaIN [54] layer at the start of each upsampling block to fuse a temporal feature vector with each intermediate feature map. AdaIN combines instance normalization with an affine transformation, and is defined as

$$\mathrm{AdaIN}(\boldsymbol{x}_t, \boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) = \boldsymbol{\sigma}_t \left(\frac{\boldsymbol{x}_t - \mu(\boldsymbol{x}_t)}{\sigma(\boldsymbol{x}_t)} \right) + \boldsymbol{\mu}_t, \tag{4}$$

where $\mu(\boldsymbol{x}_t)$ and $\sigma(\boldsymbol{x}_t)$ are the channel-wise mean and variance extracted from $\boldsymbol{x}_t$, and $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$ are channel-wise shift and scale parameters extracted from a positionally encoded temporal feature vector through a small MLP layer, as shown in Fig. 6(a). The complete decoder architecture for E-NeRV is shown in Fig. 7. PS-NeRV [22] follows a similar approach, but the AdaIN layer is applied after the upsampling step instead. Because AdaIN performs normalization, which is often used to prevent overfitting, Zhang et al. [32] argue that this might not be ideal for INR. HNeRV-BOOST [32] instead performs a similar operation to AdaIN, but without the instance normalization step, resulting in a simple affine transformation as shown in Fig. 6(b).

Instead of using purely content-agnostic temporal features, some hybrid approaches use a separate encoder branch to generate content-specific temporal embeddings, as summarized in Section 2.4. DNeRV [19] and TNeRV [36] treat this temporal embedding as the hidden

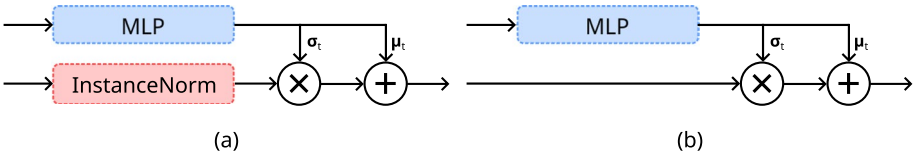

Fig. 6 Temporal information fusion blocks (**a**) AdaIN block (**b**) Affine transformation used by HNeRV-BOOST

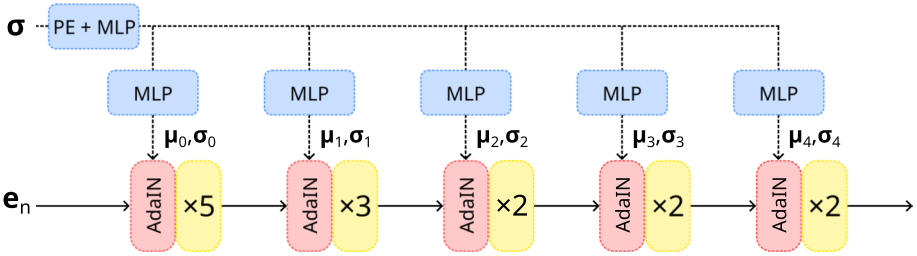

Fig. 7 Content-agnostic temporal information fusion with AdaIN

state in a recurrent neural network (RNN) by fusing the spatial and temporal embeddings in a convolutional gated recurrent unit (ConvGRU) (Fig. 8) [55]. Additionally, TNeRV first modulates both spatial and temporal embeddings with a content-agnostic temporal feature, similar to the methods described earlier in this section.

3.4 Encoder architecture

For hybrid approaches, encoder architecture also plays a role in compression performance. CNeRV [21] uses a tiny single-layer encoder, while HNeRV [25], as well as the base layer of QS-NeRV [37], use a slightly larger encoder architecture composed of five ConvNeXt [49] blocks. Because the encoder can be discarded after training, a larger encoder does not increase the size of the bitstream, but can potentially generate more powerful embeddings. Based on this observation, T-NeRV [34] uses a much larger ConvNeXt encoder to generate frame embeddings. A downside of this approach is that a larger encoder also increases the computational cost and energy usage during training.

The temporal difference encoder employed by DNeRV [19] is also built of a sequence of ConvNeXt blocks that encode the differences between the previous and current frame, and the current and next frame. TNeRV [36] uses a slightly more complex temporal diversity encoder that first extracts features from each frame separately with a series of ConvNeXt

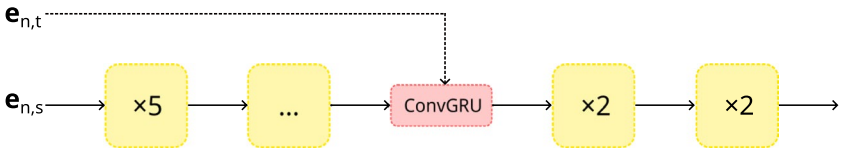

Fig. 8 Temporal information fusion with ConvGRU

blocks. The extracted features are then concatenated and processed by a 3D convolutional layer. The result is finally encoded by a series of ConvNeXt blocks into a temporal diversity feature map.

3.5 Loss function

NeRV [13] is trained with a linear combination of spatial-domain L1 and SSIM losses, with a trade-off parameter α commonly set to 0.7:

$$L(\boldsymbol{V}, \boldsymbol{i}; \boldsymbol{\Theta}) = \alpha||\boldsymbol{v_i} - \mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta})|| + (1-\alpha)(1 - \text{SSIM}(\boldsymbol{v_i}, \mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta}))), \tag{5}$$

where $\boldsymbol{v_i}$ represents the region of the ground-truth video $\boldsymbol{V}$ indexed by $\boldsymbol{i}$ and $\mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta})$ is the prediction of the same region of $\boldsymbol{V}$.

Several works have addressed the spectral bias issue discussed in Section 3.2 by incorporating frequency domain terms in the loss function. In addition to the spatial L1 loss, HNeRV-BOOST [32] also adds a frequency domain L1 loss term:

$$\begin{aligned} L(\boldsymbol{V}, \boldsymbol{i}; \boldsymbol{\Theta}) = &||\text{FFT}(\boldsymbol{v_i}) - \text{FFT}(\mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta}))|| \\ &+ \lambda\alpha||\boldsymbol{v_i} - \mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta})|| \\ &+ \lambda(1-\alpha)(1 - \text{SSIM}(\boldsymbol{v_i}, \mathcal{F}(\boldsymbol{i}; \boldsymbol{\Theta}))), \end{aligned} \tag{6}$$

where λ is another hyperparameter to balance the trade-off between the different parts of the loss function. HFS-HNeRV [39] addresses the same issue by formulating the loss function as a combination of mean squared error (MSE) loss in the spatial domain, and the MSE of a high-pass-filtered version of the reconstructed and ground truth frames. SNeRV [38] follows yet another approach by calculating the loss function from (5) over the spatial images, as well as the four wavelet coefficients.

4 Model compression

By encoding a video into the parameters of a neural network, NVR-based codecs convert the video compression problem to a model compression problem. NeRV [13] performs video compression in four sequential steps: training, model pruning, weight quantization, and entropy coding. This is still very much a hand-crafted compression pipeline, where the training step can be seen as a transformation from one domain to another, not unlike the discrete cosine transform in the traditional image and video compression pipeline. This raises the need for more advanced model compression techniques. We start by discussing weight pruning and quantization techniques in Sections 4.1 and 4.2, respectively. Section 4.3 finally discusses how rate-distortion optimization can be used to build a fully end-to-end optimized compression scheme. Table 3 provides a brief overview of works with an explicit focus on model compression.

4.1 Pruning

NeRV [13], as well as several consecutive works [19–22, 24–26, 33, 36–38], uses a simple global unstructured pruning method, where all weights smaller than some threshold θ_q are

Table 3 Overview of compression methods used in NVRs

Year	Name	Pruning	Quantization	RDO
2021	NeRV [13]	✓	SQ (Fixed)	
2023	NIRVANA [23]		SQ (Learned)	✓
	FFNeRV [26]	✓	QAT	
	Gomes et al. [27]		SQ (Learned)	✓
	C3 [28]		SQ (Learned)	✓
2024	COOL-CHIC video [29]		SQ (Learned)	✓
	HiNeRV [30]	✓	QAT	
	HNeRV-BOOST [32]		SQ (Learned)	✓
	VQ-Nerv [33]	✓	VQ	
	T-NeRV [34]		SQ (Learned)	✓
	NVRC [35]		SQ (Learned)	✓
2025	PNVC [40]		SQ (Learned)	✓

Note: This table only includes works with an explicit focus on model compression

pruned, with θ_q chosen as the q-th percentile value over all weights in $\boldsymbol{\theta}_n$. HiNeRV [30] uses a slightly more advanced pruning method, where parameters in larger layers have a higher chance to be pruned than parameters in smaller layers. This is based on the intuition that larger layers contain more redundancy, and thus pruning has less negative impact than in smaller layers.

More advanced pruning methods such as filter pruning [56] and structured sparsity [57–59] exist, both of which can also reduce computational cost. To the best of our knowledge, these methods have not yet been explored in the context of NVR.

4.2 Quantization

Since entropy coding relies on a discrete set of values to be encoded, the continuous network weights and embeddings have to be quantized before encoding.

4.2.1 Scalar quantization (SQ)

Early works on NVR [13, 19, 21, 24, 25, 36–38] use a fixed SQ step after training to quantize each weight or embedding to a fixed number of bits. SQ is implemented as an affine transformation with scale and shift parameters α and β so that

$$Q(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{x} + \beta}{\alpha} \right\rceil, \tag{7}$$

$$Q^{-1}(\boldsymbol{x}) \approx \boldsymbol{x} \times \alpha - \beta, \tag{8}$$

where $\lfloor \cdot \rceil$ represents the rounding operator. As a result of rounding, the inverse quantization function Q^{-1} is always an approximation of the original value.

4.2.2 Vector quantization (VQ)

VQ [60] presents a more general approach to quantization by grouping continuous values into several clusters, for example with k-means clustering. Each value is then represented by

the corresponding cluster centroid. VQ-NeRV [33] implements VQ by discretizing residual embeddings in a codebook. However, because of the higher complexity of VQ, SQ is often the preferred method.

4.2.3 Quantization aware training (QAT)

Instead of performing quantization as a separate step after training, several works [61, 62] have shown that the quantization error can be reduced by applying some quantization method during the training process. With QAT, the model weights are quantized and rounded before each forward pass. This approach was integrated into NVR in combination with scalar quantization by FFNeRV [26] and HiNeRV [30].

A common difficulty with quantization during training is that the gradient of the rounding operation is zero almost everywhere. This is usually avoided in one of two ways. The *straight-through estimator* (STE) [63] applies rounding during the forward pass, while acting like an identity operation during gradient computation. Alternatively, adding a *uniform noise term* with the same width as the quantization bins to the weights during training can be used as an approximation for the quantization error [64]. FFNeRV [26] uses the STE, while HiNeRV [30] uses a noise term.

4.3 Rate-distortion optimization

With fixed quantization methods such as those described in Section 4.2, finding the right quantization parameters for some rate-distortion objective always requires manual fine-tuning. Furthermore, it can be seen from Table 2 that there is no single model architecture that performs best for every possible video. Similar to picking the right quantization parameters, it is infeasible to manually tune the model architecture for a specific video. Both of these problems can be solved by treating NVR as a rate-distortion optimization (RDO) problem where a decrease in distortion always results in an increase in bitrate, and vice versa. This translates to a loss function of the form

$$L_{RDO} = L_{\text{distortion}} + \lambda L_{\text{rate}}, \tag{9}$$

where $L_{\text{distortion}}$ represents any measure of distortion as outlined in Section 3.5, L_{rate} represents an approximation of the length of the bitstream encoding the network parameters $\boldsymbol{\theta}_n$ and embeddings $\boldsymbol{e}_n$, and λ is a hyper-parameter that controls the trade-off between the two.

4.3.1 Rate minimization for latent representations

RDO has been widely used in autoencoder-based compression models for images [65, 66] and videos [4–6], where the input is first converted to a latent representation $\boldsymbol{y}$, which is then quantized to $\hat{\boldsymbol{y}}$ and encoded with entropy coding. According to Shannon's source coding theorem [67], the minimal length of a bit stream encoding $\hat{\boldsymbol{y}}$ is equal to the entropy of $\hat{\boldsymbol{y}}$, based on the underlying probability distribution p. In practice, p is unknown but can be approximated by some estimate $q_{\boldsymbol{\phi}}$ of the true probability distribution, with a separate set of parameters $\boldsymbol{\phi}$. Therefore, the rate and distortion can be optimized jointly by defining L_{rate} as the cross-entropy between p and $q_{\boldsymbol{\phi}}$,

$$L_{\text{rate}} = \mathbb{E}_{\hat{\boldsymbol{y}} \sim p}[-\log_2 q_{\boldsymbol{\phi}}(\hat{\boldsymbol{y}})]. \tag{10}$$

C3 [28] and COOL-CHIC video [11] incorporate this method into an NVR scheme to minimize the rate consumed by the instance embeddings. However, both models fail to include the rate penalty caused by the model parameters in the loss function, causing them to not be fully end-to-end optimized.

4.3.2 Rate minimization for model parameters

Oktay et al. [68] were the first to adapt RDO to a general model compression scheme. During each forward pass, a learned transformation $f_{\boldsymbol{\phi}}$ is applied to decode the model parameters $\boldsymbol{\theta}$ from a quantized latent space $\hat{\boldsymbol{\theta}}$. Since $f_{\boldsymbol{\phi}}$ is used as an inverse quantization function, Gomes et al. [27], NIRVANA [23], HNeRV-BOOST [32] and T-NeRV [34] all formulate $f_{\boldsymbol{\phi}}$ as in (8), but more complex transformations are also possible [35]. Similar to the quantized latent representation $\hat{\boldsymbol{y}}$ in [65] and (10), a learned entropy model $q_{\boldsymbol{\phi}}$ is used to tune $f_{\boldsymbol{\phi}}$ to the rate-distortion objective. This results in two additional sets of parameters, $\boldsymbol{\phi} = \{\boldsymbol{\phi}_{\text{quant}}, \boldsymbol{\phi}_{\text{em}}\}$. Finally, $\hat{\boldsymbol{\theta}}$ is entropy coded and transmitted along with $\boldsymbol{\phi}$.

Recall from Section 4.2 that quantization uses the non-differentiable rounding operator. When calculating the rate term, the rounding error is approximated by a noise term. When calculating the distortion term, the STE is used in order to avoid using noisy parameters for inference.

In practice, separate quantization and entropy models are learned for each layer of the network to allow the entropy model to better capture the distribution of each parameter group independently. This also allows the model to autonomously assign more or fewer bits to different parts of the network, based on the encoded content. For example, experiments by the authors of both T-NeRV [34] and NVRC [35] show that multiresolution temporal grid embeddings take up more bits per parameter for dynamic content than for static content to better exploit their temporal information. This suggests that RDO makes it possible to build a single NVR model that can automatically tune itself to many diverse input videos.

NVRC [35] extends the reparameterization approach with more complex quantization and entropy models. Because this results in more overhead from encoding $\boldsymbol{\phi}$, $\boldsymbol{\phi}$ itself is quantized as $\hat{\boldsymbol{\phi}} = \{\hat{\boldsymbol{\phi}}_{\text{quant}}, \hat{\boldsymbol{\phi}}_{\text{em}}\}$ and encoded with another set of learned quantization and entropy models parameterized by $\boldsymbol{\psi} = \{\boldsymbol{\psi}_{\text{quant}}, \boldsymbol{\psi}_{\text{em}}\}$. Finally, $\boldsymbol{\psi}$ is simply quantized into full/half precision as $\hat{\boldsymbol{\psi}} = \{\hat{\boldsymbol{\psi}}_{\text{quant}}, \hat{\boldsymbol{\psi}}_{\text{em}}\}$.

Comparisons between different NVR-based codecs, and HEVC HM and VVC VTM can be found in NVRC [35] and PNVC [40]. Both methods report Bjøntegaard Delta-rate (BD-rate) savings on the UVG [18] and MCL-JCV [69] datasets. As in Table 2, the exact numbers are not directly comparable due to differences in color space, anchor configuration, and evaluation settings. NVRC reports a 50-70% reduction in BD-rate on UVG in RGB space compared to HM, and a 20-50% reduction compared to VTM, clearly outperforming earlier approaches such as FFNeRV [26], HiNeRV [30], HNeRV-BOOST [32], and C3 [28] due to the more complex coding structure used by NVRC. The latter three approaches fall in a similar performance band as HM, but this highly depends on the dataset, configurations, and distortion metrics used. PNVC, evaluated against HM in YUV space, achieves around

35–40% BD-rate reduction in random-access mode, which is competitive with NVRC, as well as autoencoder-based methods like DCVC-HEM [70] and DCVC-DC [71].

5 Practical considerations for video streaming

After having reviewed the existing NVR techniques, we wish to identify common weaknesses and opportunities for future work. We will consider potential applications in an over-the-top (OTT) video streaming context, in which video compression plays a crucial role. Several factors must be considered before NVR-based video codecs can be deployed in a practical video streaming scenario. Adaptive bitrate (ABR) streaming paradigms such as MPEG-DASH [72] require videos to be split into short segments and encoded at several fixed bitrates, allowing streaming clients to select the appropriate bitrate for the current network conditions. Other obvious factors that determine the final Quality of Experience (QoE) are video reconstruction quality, decoding complexity, and, in the case of live streaming, encoding delay. The first requirement is easy to satisfy with an NVR-based codec, since NVR models are best trained on short video segments. In classical codecs, decoding can start almost immediately after receiving the bitstream, as frames are progressively reconstructed using predictive coding. In contrast, NVR-based codecs require all model parameters to be available before decoding can begin, introducing an initialization delay, especially when the scope is large. In other words, it is beneficial to use a smaller scope, or even a single-frame scope as proposed by COOL-CHIC video [29] and PNVC [40] for streaming. Sections 5.1, 5.2, 5.3, and 5.4 discuss common challenges related to the requirements of reconstruction quality, rate control, client-side feasibility, and encoding complexity, respectively.

5.1 Spectral bias

The spectral bias towards lower frequencies is the most common issue related to the reconstruction quality of NVRs and INRs in general. We already discussed several different methods to alleviate the spectral bias issue throughout this paper. PS-NeRV [22] has shown that a patch-wise approach generally preserves more high-frequency features than a frame-wise approach (Section 2.1.3). HiNeRV [30] additionally adds high-frequency information after each upsampling step (Section 3.1). HFS-HNeRV [39] and SNeRV [38] use wavelet transforms in the decoder network to add inductive biases towards high-frequency components (Section 3.2). Lastly, several works also add frequency-domain components to the loss function [32, 38, 39] (Section 3.5). For general INRs, Shi et al. [73] have introduced a weight reparameterization scheme that can also alleviate the spectral bias issue. Since spectral bias is inherently related to neural networks [44], NVR models should always compensate for this explicitly.

5.2 Rate control and explainability

For ABR streaming to work correctly, each segment must be encoded at fixed bitrates. We have already discussed several levers that can control the final bitrate of an NVR model. On the one hand, the number of parameters in the model can be adjusted manually, for example, by changing the number of channels in each intermediate feature map (Section 3 and Fig.

7). On the other hand, the rate-distortion tradeoff parameter λ can adjust the relative weight of the rate and distortion terms in the rate-distortion objective (Section 4.3). Increasing the scope of each model also reduces the bitrate but causes higher latency for streaming (Section 2.2).

Despite these levers, existing NVR approaches lack explicit rate control mechanisms that reliably target a desired bitrate. As a result, it is still unclear how to find the best configuration for any combination of input video and target bitrate. This remains an important direction for future work and relates to the broader challenge of explainability in deep learning, where it is often unclear which combination of hyperparameters and network configurations is optimal for a certain objective. Addressing this challenge starts with understanding which parts of the network contribute most to the output. A concrete step towards explainability in NVRs is taken by XINC [74], a promising tool that visualizes the contributions of different network weights to individual output pixels.

5.3 Feasibility on client devices

Since decoding is typically performed on resource-constrained client devices, the feasibility of NVR-based codecs depends on several factors, including decoding complexity, memory footprint, energy consumption, and hardware support. Existing work has mostly focused on designing more parameter-efficient models and compression methods that reduce the footprint of stored model weights. However, improved parameter efficiency often also introduces higher computational complexity [30]. Future work should also focus on incorporating existing methods for complexity reduction in the NVR pipeline. As mentioned in Section 4.1, advanced pruning methods, such as filter pruning and structured sparsity, remain underexplored for NVR models. Girish et al. [59] have already shown how structured sparsity can be used in a scheme that jointly minimizes bitrate and computational cost. Other techniques for reducing model complexity include weight reparameterization and knowledge distillation, both of which were briefly discussed in Section 3.1. PNVC [40] incorporates both of these techniques in an NVR framework. Unfortunately, both reparameterization and knowledge distillation schemes increase the computational complexity during training.

Energy consumption is another critical factor, particularly for battery-powered devices. Traditional codecs benefit from highly optimized hardware decoders designed for minimal power consumption, whereas NVR-based decoding relies on neural network inference, which is generally more energy-intensive on general-purpose hardware. While smaller models and reduced-precision inference (e.g., 8- or 16-bit weights) can mitigate this issue, energy efficiency remains a potential bottleneck for widespread mobile deployment.

In terms of memory requirements, NVR-based codecs differ significantly from traditional codecs. Traditional codecs require only a limited decoded picture buffer and small amounts of side information, resulting in a relatively low memory footprint. In contrast, NVR-based codecs require the full neural network to be stored in memory during decoding. This can lead to significantly higher memory usage, particularly for high-capacity models. On the other hand, NVR models are typically more lightweight than autoencoder-based neural video codecs, as they rely on compact, video-specific representations rather than large encoder and decoder networks. As such, NVR-based codecs occupy an intermediate position between traditional codecs and autoencoder-based approaches in terms of memory footprint.

Notwithstanding the ongoing NVR research efforts (as outlined above) with respect to model complexity reduction, energy expenditure, and memory footprint, the feasibility of deploying NVR-based codecs on mobile and edge devices remains largely underexplored. In particular, systematic evaluations of latency, energy consumption, and real-time decoding performance on commodity hardware are needed to assess their practical viability. While some works report metrics such as decoding throughput and computational complexity (e.g., multiply-accumulate operations), these evaluations remain relatively scarce and are often conducted on different hardware platforms, making direct comparisons difficult [30]. This suggests that deployment-oriented experiments should receive greater attention in future evaluations.

Despite these challenges, NVR-based codecs may also offer an advantage in terms of hardware flexibility. Conventional codecs depend on specialized hardware implementations that require standardization and long deployment cycles. In contrast, NVR-based codecs can be executed on general-purpose GPUs and increasingly available neural processing units (NPUs) in modern mobile systems-on-chip, enabling deployment without dedicated hardware support. Moreover, since model parameters are transmitted as part of the bitstream, updating the codec effectively reduces to transmitting a new model, with minimal additional overhead. This flexibility may enable faster innovation cycles compared to traditional codecs.

5.4 Encoding complexity

For live streaming, encoding complexity is of even greater importance than for non-live streaming since training NVR models consumes considerable time, computational resources, and energy. When training multiple subsequent NVR models, encoding time can be reduced by fine-tuning each set of parameters $\boldsymbol{\theta}_n$ from the previous parameters $\boldsymbol{\theta}_{n-1}$ [23, 40]. Alternatively, hyper-learning has been applied to general INRs [75, 76], as well as NeRV [77] to speed up encoding. Instead of optimizing the model weights with gradient descent, a hypernetwork is trained to predict the model weights directly for some input video. Hyper-learning significantly reduces encoding times, but also reintroduces the reliance on large datasets for training. Further research is needed to investigate how hyper-learning can be applied to more complex, end-to-end optimized NVR pipelines. Finally, research in areas outside INR has shown that the computational complexity of neural networks can be reduced by using weights in 16-bit [78] or even 8-bit [61] precision during training, with little impact on performance. With many optimization strategies still underexplored, future research has yet to point out whether real-time encoding with NVR-based codecs is achievable.

6 Conclusion

NVR-based video codecs present an alternative to both traditional video codecs and autoencoder-based learned video compression schemes. Learned video compression schemes generally have more potential for optimization than traditional video codecs because they do not rely on hand-crafted heuristics. Because of their overfitting nature, NVR-based codecs can be fine-tuned for specific videos without relying on large datasets and have a lower decoding complexity compared to autoencoder-based compression schemes. This survey provides a

systematic overview of the representative works, and discusses several techniques related to model design and compression. While this paper has focused on natural video content in the context of on-demand streaming, NVR-based codecs may also be applicable to other domains such as medical, scientific, or industrial imaging.

Despite the advantages, we have also identified several limitations that hinder the practical deployment of NVRs for on-demand video streaming, suggesting several directions for future research. The primary challenge lies in the high encoding complexity caused by per-video optimization. It is still an open question whether real-time encoding can be achieved with NVRs. On the decoding side, the lower complexity of NVRs relative to autoencoders is attractive for resource-constrained clients, though the practical feasibility of decoding on mobile and edge devices remains underexplored. Furthermore, improving the explainability of NVR models can aid in developing better model architectures and in configuring hyperparameters to achieve the precise bitrate targets required for adaptive streaming. Addressing spectral bias also remains important, particularly for high-quality reconstruction at low bitrates. Finally, future evaluations should move beyond PSNR as the primary measure of reconstruction quality. While PSNR remains the most widely reported metric in the NVR literature, it does not always correlate well with human visual perception. Quality metrics such as MS-SSIM, which evaluates structural similarity across multiple image scales, and VMAF, which combines multiple perceptual features into a machine learning-based quality score, can provide a more representative assessment of the quality perceived by the human visual system. Such metrics would provide a more realistic evaluation of NVR-based codecs, particularly for streaming applications that aim to maximize subjective Quality of Experience.

Author Contributions Hannes Keunen: Literature review, writing original draft. Maarten Wijnants and Jori Liesenborgs: Review and editing.

Funding This work is supported by the Special Research Fund of Hasselt University under grant number BOF24OWB27.

Data Availability Not applicable.

Declarations

Ethical Approval The authors declare that this work has not been published elsewhere, and has not been submitted to more than one journal for simultaneous consideration.

Consent to Participate All authors consent to submit this work in its current form.

Consent to Publish All authors consent to publish this work.

Competing Interests The authors declare that they have no conflict of interest.

References

1. Bross B, Wang Y-K, Ye Y, Liu S, Chen J, Sullivan GJ, Ohm J-R (2021) Overview of the versatile video coding (VVC) standard and its applications. IEEE Trans Circ Syst Video Techn 31(10):3736–3764. https://doi.org/10.1109/TCSVT.2021.3101953
2. Sullivan GJ, Ohm J-R, Han W, Wiegand T (2012) Overview of the high efficiency video coding (HEVC) standard. IEEE Trans Circ Syst Video Techn 22(12):1649–1668. http://doi.ieeecomputersociety.org/10.1109/TCSVT.2012.2221191
3. Pakdaman F, Adelimanesh MA, Gabbouj M, Hashemi MR (2020) Complexity analysis of next-generation VVC encoding and decoding. In: IEEE Int Conf Image Process (ICIP 2020), pp 3134–3138. https://doi.org/10.1109/ICIP40778.2020.9190983
4. Lu G, Ouyang W, Xu D, Zhang X, Cai C, Gao Z (2019) DVC: an end-to-end deep video compression framework. In Proc IEEE conf comput vis pattern recognit (CVPR 2019), pp 11 006–11 015
5. Li J, Li B, Lu Y (2024) Neural video compression with feature modulation. In Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2024), pp 26 099–26 108. https://doi.org/10.1109/CVPR52733.2024.02466
6. Mentzer F, Toderici G, Minnen D, Caelles S, Hwang SJ, Lucic M, Agustsson E (2022) VCT: a video compression transformer. In: Adv neural inf process syst 35 (NeurIPS 2022)
7. Chen Z, Zhang H (2019) Learning implicit fields for generative shape modeling. In: Proc. IEEE/CVF conf comput vis pattern recognit (CVPR 2019), pp 5939–5948
8. Mescheder LM, Oechsle M, Niemeyer M, Nowozin S, Geiger A (2019) Occupancy networks: Learning 3D reconstruction in function space. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2019), pp 4460–4470
9. Park J-J, Florence P, Straub J, Newcombe RA, Lovegrove S (2019) DeepSDF: Learning continuous signed distance functions for shape representation. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2019), pp 165–174
10. Dupont E, Goliński A, Alizadeh M, Teh YW, Doucet A (2021) COIN: compression with implicit neural representations. arXiv:2103.03123
11. Ladune T, Philippe P, Henry F, Clare G, Leguay T (2023) COOL-CHIC: coordinate-based low complexity hierarchical image codec. In: Proc IEEE/CVF int conf comput vis (ICCV 2023), pp 13 469–13 476. https://doi.org/10.1109/ICCV51070.2023.01243
12. Sitzmann V, Martel JNP, Bergman AW, Lindell DB, Wetzstein G (2020) Implicit neural representations with periodic activation functions. In: Adv neural inf process syst 33 (NeurIPS 2020)
13. Chen H, He B, Wang H, Ren Y, Lim S-N, Shrivastava A (2021) NeRV: neural representations for videos. In: Adv neural inf process syst 34 (NeurIPS 2021), pp 21 557–21 568
14. Gwilliam M, Zhang R, Padmanabhan N, Du H, Shrivastava A (2026) How to design and train your implicit neural representation for video compression. In: Proc. IEEE/CVF winter conf applications comput vis (WACV 2026), pp 729–739
15. Huo S, Liu D, Zhang H, Li L, Ma S, Wu F, Gao W (2024) Towards hybrid-optimization video coding. ACM Comput Surv 56(9). https://doi.org/10.1145/3652148
16. Jia C, Ye F, Ma S, Gao W, Sun H, Chiariglione L (2025) Emerging advances in learned video compression: models, systems and beyond. In: Proc int jt conf artif intell (IJCAI 2025). https://doi.org/10.24963/ijcai.2025/1165
17. Essakine A, Cheng Y, Cheng C, Zhang L, Deng Z, Zhu L, Schönlieb C, Aviles-Rivero A (2025) Where do we stand with implicit neural representations? A technical and performance survey. Transactions on Machine Learning Research, vol 2025-February, pp 1–25
18. Mercat A, Viitanen M, Vanne J (2020) UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In: Proc 11th ACM multimed syst conf, pp 297–302. https://doi.org/10.1145/3339825.3394937
19. Zhao Q, Asif MS, Ma Z (2023) DNeRV: modeling inherent dynamics via difference neural representation for videos. In: Proc. IEEE/CVF conf comput vis pattern recognit (CVPR 2023), pp 2031–2040. https://doi.org/10.1109/CVPR52729.2023.00202
20. Li Z, Wang M, Pi H, Xu K, Mei J, Liu Y (2022) E-NeRV: Expedite neural video representation with disentangled spatial-temporal context. In: Proc 17th Eur conf comput vis (ECCV 2022), pp 267–284. https://doi.org/10.1007/978-3-031-19833-5_16

21. Chen H, Gwilliam M, He B, Lim S-N, Shrivastava A (2022) CNeRV: content-adaptive neural representation for visual data. In: Br machine vis conf (BMVC 2022), p 510
22. Bai Y, Dong C, Wang C, Yuan C (2023) PS-NeRV: patch-wise stylized neural representations for videos. In: IEEE int conf image process (ICIP 2023), pp 41–45. https://doi.org/10.1109/ICIP49359.2023.10222144
23. Maiya SR, Girish S, Ehrlich M, Wang H, Lee KS, Poirson P, Wu P, Wang C, Shrivastava A (2023) NIRVANA: neural implicit representations of videos with adaptive networks and autoregressive patch-wise modeling. In: Proc. IEEE/CVF conf comput vis pattern recognit (CVPR 2023), pp 14 378–14 387
24. He B, Yang X, Wang H, Wu Z, Chen H, Huang S, Ren Y, Lim S-N, Shrivastava A (2023) Towards scalable neural representation for diverse videos. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2023), pp 6132–6142. https://doi.org/10.1109/CVPR52729.2023.00594
25. Chen H, Gwilliam M, Lim S-N, Shrivastava A (2023) HNeRV: a hybrid neural representation for videos. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2023), pp 10 270–10 279. https://doi.org/10.1109/CVPR52729.2023.00990
26. Lee JC, Rho D, Ko JH, Park E (2023) FFNeRV: flow-guided frame-wise neural representations for videos. In: Proc 31st ACM int conf multimed (MM 2023), pp 7859–7870. https://doi.org/10.1145/3581783.3612444
27. Gomes C, Azevedo R, Schroers C (2023) Video compression with entropy-constrained neural representations. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2023), pp 18 497–18 506. https://doi.org/10.1109/CVPR52729.2023.01774
28. Kim H, Bauer M, Theis L, Schwarz JR, Dupont E (2024) C3: High-performance and low-complexity neural compression from a single image or video. In: Proc. IEEE/CVF conf comput vis pattern recognit. (CVPR 2024), pp 9347–9358. https://doi.org/10.1109/CVPR52733.2024.00893
29. Leguay T, Ladune T, Philippe P, Déforges O (2024) Cool-chic video: learned video coding with 800 parameters. In: Proc data compression conf (DCC 2024), pp 23–32. https://doi.org/10.1109/DCC58796.2024.00010
30. Kwan HM, Gao G, Zhang F, Gower A, Bull D (2023) HiNeRV: video compression with hierarchical encoding-based neural representation. In: Adv neural inf process syst 36 (NeurIPS 2023)
31. Ghorbel A, Hamidouche W, Morin L (2024) NeRV++: an enhanced implicit neural video representation. In: Proc IEEE int conf vis commun and image process (VCIP 2024). IEEE, pp 1–5. https://doi.org/10.1109/VCIP63160.2024.10849795
32. Zhang X, Yang R, He D, Ge X, Xu T, Wang Y, Qin H, Zhang J (2024) Boosting neural representations for videos with a conditional decoder. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2024), pp 2556–2566. https://doi.org/10.1109/CVPR52733.2024.00247
33. Zhang G, Tang L, Zhang X (2024) VQ-NeRV: vector quantization neural representation for video compression. In: Proc IEEE int symp circuits syst (ISCAS 2024), pp 1–5. https://doi.org/10.1109/ISCAS58744.2024.10558613
34. Saethre JE, Azevedo R, Schroers C (2024) Combining frame and GOP embeddings for neural video representation. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2024), pp 9253–9263. https://doi.org/10.1109/CVPR52733.2024.00884
35. Kwan HM, Gao G, Zhang F, Gower A, Bull D (2024) NVRC: Neural video representation compression. In: Adv neural inf process syst 37 (NeurIPS 2024), pp 132 440–132 462
36. Wang J, Liu Y, Zhu S, Feng C (2024) Temporal enhanced hybrid neural representation for video compression. In: Proc picture coding symp (PCS 2024), pp 1–5. https://doi.org/10.1109/PCS60826.2024.10566352
37. Wu C, Quan G, He G, Lai X-Q, Li Y, Yu W, Lin X, Yang C (2024) QS-NeRV: real-time quality-scalable decoding with neural representation for videos. In: Proc 32nd ACM int conf multimed (MM 2024), pp 2584–2592. https://doi.org/10.1145/3664647.3680586
38. Kim J, Lee J, Kang J-W (2024) SNeRV: spectra-preserving neural representation for video. In: Proc. 18th Eur conf comput vis (ECCV 2024), pp 332–348. https://doi.org/10.1007/978-3-031-73001-6_19
39. Zhao J, Li XJ, Chong PHJ (2024) HFS-HNeRV: high-frequency spectrum hybrid neural representation for videos. In: Proc. 6th ACM int conf multimed Asia (MMAsia 2024). https://doi.org/10.1145/3696409.3700250
40. Gao G, Kwan HM, 0017 FZ, 0001 DB (2025) PNVC: towards practical inr-based video compression. In: AAAI 2025, pp 3068–3076. https://doi.org/10.1609/aaai.v39i3.32315
41. Strümpler Y, Postels J, Yang R, Gool LV, Tombari F (2022) Implicit neural representations for image compression. In: Proc 17th Eur conf comput vis (ECCV 2022), pp 74–91. https://doi.org/10.1007/978-3-031-19809-0_5
42. Boyce JM, Ye Y, Chen J, Ramasubramonian AK (2016) Overview of SHVC: scalable extensions of the high efficiency video coding standard. IEEE Trans Circ Syst Video Techn 26(1):20–34. https://doi.org/10.1109/TCSVT.2015.2461951

43. Tancik M, Srinivasan PP, Mildenhall B, Fridovich-Keil S, Raghavan N, Singhal U, Ramamoorthi R, Barron JT, Ng R (2020) Fourier features let networks learn high frequency functions in low dimensional domains. In: Adv neural inf process syst 33 (NeurIPS 2020)
44. Rahaman N, Baratin A, Arpit D, Dräxler F, Lin M, Hamprecht FA, Bengio Y, Courville AC (2019) On the spectral bias of neural networks. In: Proc 36th int conf machine learn (ICML 2019), vol 97, pp 5301–5310
45. Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R (2020) NeRF: representing scenes as neural radiance fields for view synthesis. In: Proc 16th Eur conf comput vis (ECCV 2022), vol 12346, pp 405–421. https://doi.org/10.1007/978-3-030-58452-8_24
46. Ronneberger O, Fischer P, Brox T (2015) U-Net: convolutional networks for biomedical image segmentation. In: Med image comput comput-assisted intervention (MICCAI 2015), pp 234–241. https://doi.org/10.1007/978-3-319-24574-4_28
47. Shi W, Caballero J, Huszar F, Totz J, Aitken AP, Bishop R, Rueckert D, Wang Z (2016) Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: Proc IEEE/CVF conf. comput vis pattern recognit (CVPR 2016). http://doi.ieeecomputersociety.org/10.1109/CVPR.2016.207
48. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) MobileNets: efficient convolutional neural networks for mobile vis. applications. arXiv:1704.04861
49. Liu Z, Mao H, Wu C-Y, Feichtenhofer C, Darrell T, Xie S (2022) A ConvNet for the 2020s. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2022), pp 11 966–11 976
50. Zagoruyko S, Komodakis N (2017) DiracNets: training very deep neural networks without skip-connections. arXiv:2103.03123
51. Gou J, Yu B, Maybank SJ, Tao D (2021) Knowledge distillation: a survey. Int J Comput Vis 129(6):1789–1819. https://doi.org/10.1007/s11263-021-01453-z
52. Wang G-H, Li J, Li B, Lu Y (2023) EVC: towards real-time neural image compression with mask decay. In: Proc int conf learn representations (ICLR 2023)
53. Peng T, Gao G, Sun H, Zhang F, Bull DR (2024) Accelerating learnt video codecs with gradient decay and layer-wise distillation. In: Picture coding symp (PCS 2024), pp 1–5. https://doi.org/10.1109/PCS60826.2024.10566283
54. Huang X, Belongie SJ (2017) Arbitrary style transfer in real-time with adaptive instance normalization. In: Proc IEEE int conf comput vis (ICCV 2017), pp 1510–1519. https://doi.org/10.1109/ICCV.2017.167
55. Ballas N, Yao L, Pal CJ, Courville AC (2016) Delving deeper into convolutional networks for learning video representations. In: Int conf on learn representations (ICLR 2016)
56. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2017) Pruning filters for efficient convnets. In: Proc 5th int conf learn representations (ICLR 2017)
57. Zhou H, Alvarez JM, Porikli F (2016) Less is more: towards compact CNNs. In: Proc 14th Eur conf comput vis (ECCV 2016), vol 9908, pp 662–677. https://doi.org/10.1007/978-3-319-46493-0_40
58. Wen W, Wu C, Wang Y, Chen Y, Li H (2016) Learning structured sparsity in deep neural networks. In: Adv neural inf process syst 29 (NeurIPS 2016), pp 2074–2082
59. Girish S, Gupta K, Singh S, Shrivastava A (2023) LilNetX: lightweight networks with extreme model compression and structured sparsification. In: Proc int conf learn representations (ICLR 2023), Kigali, Rwanda
60. Yang S, Mao Y (2022) Vector quantization of deep convolutional neural networks with learned codebook. In: Proc. 17th can workshop inf theory (CWIT 2022), pp 39–44
61. Banner R, Hubara I, Hoffer E, Soudry D (2018) Scalable methods for 8-bit training of neural networks. In: Adv neural inf process syst 31 (NeurIPS 2018), pp 5151–5159
62. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard AG, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proc IEEE conf comput vis pattern recognit (CVPR 2018), pp 2704–2713
63. Bengio Y, Léonard N, Courville A (2013) Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432
64. Stock P, Fan A, Graham B, Grave E, Gribonval R, Jégou H, Joulin A (2021)Training with quantization noise for extreme model compression. In: 9th Int conf on learn representations (ICLR 2021). OpenReview.net
65. Ballé J, Laparra V, Simoncelli EP (2017) End-to-end optimized image compression. In: Int Conf on Learn Representations (ICLR 2017)
66. Ballé J, Minnen D, Singh S, Hwang SJ, Johnston N (2018) Variational image compression with a scale hyperprior. In: Int conf on learn representations (ICLR 2018)
67. Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27(3):379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

68. Oktay D, Ballé J, Singh S, Shrivastava A (2020) Scalable model compression by entropy penalized reparameterization. In: Proc. 8th int conf learn represent (ICLR 2020)
69. Wang H, Gan W, Hu S, Lin JY, Jin L, Song L, Wang P, Katsavounidis I, Aaron A, Kuo CCJ (2016) MCL-JCV: a JND-based H.264/AVC video quality assessment dataset. In: Proceedings of the 2016 IEEE international conference on image processing (ICIP 2016), Phoenix, AZ, USA, pp 1509–1513
70. Li J, Li B, Lu Y (2022) Hybrid spatial-temporal entropy modelling for neural video compression. In: Proc 30th ACM int conf multimed (MM 2022)
71. Li J, Li B, Lu Y (2023) Neural video compression with diverse contexts. In Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2023)
72. Sodagar I (2011) The MPEG-DASH standard for multimedia streaming over the internet. IEEE Multimedia 18(4):62–67. http://doi.ieeecomputersociety.org/10.1109/MMUL.2011.71
73. Shi K, Zhou X, Gu S (2024) Improved implicit neural representation with fourier reparameterized training. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2024), pp 25 985–25 994. https://doi.org/10.1109/CVPR52733.2024.02455
74. Padmanabhan N, Gwilliam M, Kumar P, Maiya SR, Ehrlich M, Shrivastava A (2024) 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), pp 10 957–10 967
75. Chen Y, Wang X (2022) Transformers as meta-learners for implicit neural representations. In: Proc 17th Eur conf comput vis (ECCV 2022), vol 13677, pp 170–187. https://doi.org/10.1007/978-3-031-19790-1_11
76. Kim C, Lee D, Kim S, Cho M, Han W-S (2023) Generalizable implicit neural representations via instance pattern composers. In: Proc IEEE/CVF conf comput vis pattern recognit (CVPR 2023), pp 11 808–11 817. https://doi.org/10.1109/CVPR52729.2023.01136
77. Chen H, Xie S, Lim S-N, Shrivastava A (2024) Fast encoding and decoding for implicit video representation. In: Proc 18th Eur conf comput vis (ECCV 2024), pp 402–418. https://doi.org/10.1007/978-3-031-72933-1_23
78. Micikevicius P, Narang S, Alben J, Diamos GF, Elsen E, García D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, Wu H (2018) Mixed precision training. In: Proc 6th int conf learn represent (ICLR 2018)