NerVast: Compression-Efficient Scaling of Implicit Neural Video Representations via Scene-based Parameter-sharing

Yunheon Lee[1] Juncheol Ye[1] Jaehong Kim[2*] Dongsu Han[1*]
[1] KAIST, Republic of Korea [2] Inha University, Republic of Korea
{yhlee37, juncheol, dhan.ee}@kaist.ac.kr, jaehong.kim@inha.ac.kr

Abstract

Implicit neural representation (INR) has emerged as a new data representation for compressing videos and now shows on-par performance with the conventional codecs. The next challenge in the field is to make INR scalable for its practical use. Existing works realize this by utilizing small INR models to scale for long and high-resolution video, which achieves better encoding and decoding speeds. However, they fail to fully exploit the temporal nature of video data when encoding it into multiple separate INRs across time, which leads to sub-optimal compression efficiency. In this work, we propose NerVast, a new encoding scheme for video INR, that improves compression efficiency while still enjoying the low computation and transfer costs of small INR models. When a video is represented in separate INR segments, NerVast effectively reduces the total volume required for representation by sharing the parameters between models during encoding. Without expensive training, NerVast selects the most efficient parameters to share. Then it jointly trains both shared and non-shared parameters in a way that minimizes the quality drop imposed by sharing. While maintaining real-time decoding speed (> 30 fps), NerVast provides better compression (39.9 % reduction in parameters on average) compared to the compute-efficient INR models. In other words, NerVast is better in encoding quality (1.57 dB higher in PSNR) with the same bitrate.

1. Introduction

Implicit neural representation (INR) is a novel data representation technique that has recently gained attention for its ability to represent complex signal data, including audio [32], images [10, 11], and even 3D scenes [3, 28]. INR represents the data as a form of learnable function that takes a specific coordinate as input and returns the target value of representing data ($f_\theta(\cdot) : \mathcal{R}^n \rightarrow \mathcal{R}^m$). This technique also shows great potential in video representation. NeRV [5] reconstructs video using INR with acceptable visual quality, while other recent works [6, 17, 21] achieve comparable or even better compression performance than existing codecs such as H.264 [42] and HEVC [36].

In addition to its compression performance, another important requirement of INR for its real-world deployment is the ability to scale to large data. In the case of video, using a large monolithic INR model to compress a long sequence video is impractical because it translates to higher computation costs in both encoding (training) and decoding (inference); especially, it leads to unacceptable frame rates for decoding (< 30 fps) even with the powerful GPUs. The approach also hinders its use in streaming scenarios as reconstructing a single video frame with INR requires an entire model to be transferred [7].

To address the issue, one straightforward approach is to utilize multiple small INR models, each representing a small part of the whole data. This design philosophy is actively being adopted in INR for static 3D scenes to successfully model massive urban scenes [37, 39] or speed up the inference [30, 31].

Likewise, we can split the video into multiple small INRs across time to get the same practical benefit for long videos. However, unlike static 3D scenes, videos bear strong temporal redundancy. Naively encoding adjacent video chunks over time into *independent* models fails to exploit this unique characteristic, which may lead to sub-optimal compression efficiency. For example, imagine a video where the same objects or background scenery repeatedly show up over a long time, but the video is encoded into multiple independent INRs across time. This can result in redundant parameters having similar weights between models, which implies that there is room for further reduction of the total number of parameters while still achieving comparable quality. Recent work named NIRVANA [26] also points out that temporal redundancy is overlooked when a video is represented into multiple INRs. It proposes a new model architecture that speeds up the encoding and decoding time by leveraging temporal redundancy and multiple smaller models. However, leveraging the temporal nature of video to

*Co-corresponding authors.

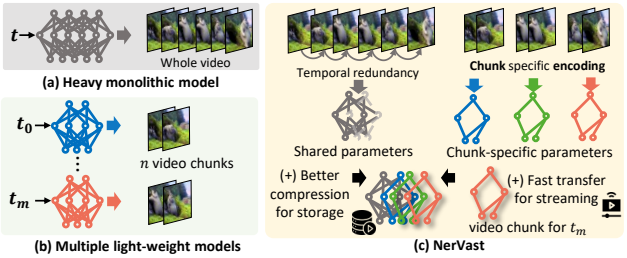


Figure 1. Overview of NerVast. Our work enables efficient scaling with small computation and transfer costs with better compression efficiency.

achieve better *compression efficiency* is an unexplored opportunity.

In this paper, we propose NerVast, a new encoding method that compresses a video into multiple small INR models by exploiting the strong temporal redundancy in the video. Our goal is to enhance the compression efficiency while taking advantage of the low computation cost and transfer overhead of small INRs, without modifying the model architecture itself. To achieve this, NerVast transforms repetitive information present in multiple video chunks into shared parameters across small INR models as shown in Fig. 1. Specifically, it carefully selects the sharable parameters and encodes them into the same weights to make them contain the long-term redundancy in a video. This strategy effectively reduces the overall parameter count required to represent the same video. Additionally, NerVast effectively handles the remaining non-shared parameters to express the residual information of each video chunk for better quality gains.

The approach is promising but careful consideration must be given to the following factors: 1) finding the extent of redundancy between video chunks that can be efficiently modeled using shared parameters, 2) identifying the parameters worthy of selection for sharing.

To address the first factor, we leverage the threshold-based scene change detection method to define a parameter-sharing window, which groups video chunks that will share the same parameter. For the second factor, we introduce Fisher's information score as a metric to select which parameters to share within the window. Furthermore, when training multiple small INRs for encoding, it is crucial to strike a balance between capturing temporal redundancy among video chunks and accurately representing each video chunk. Our proposed approach effectively addresses this challenge by employing partial joint training of INR.

In summary, the contribution of our work is as follows: 1) NerVast demonstrates how exploiting the strong temporal redundancy inherent in videos can enhance the compression efficiency of the video when it is represented in small INR models. 2) NerVast enables efficient scaling to large video with small computation and transfer costs. 3) NerVast introduces the new encoding scheme for multiple small INRs to select and learn shared parameters, significantly enhancing the compression efficiency.

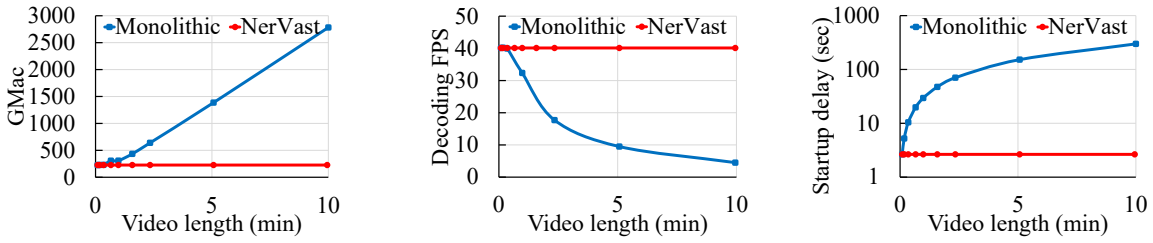
2. Background & Related Work

Implicit Neural Representation is an emerging paradigm for data representation using neural networks. In particular, coordinate-based neural representation techniques [28, 32] have gained significant attention, which takes input coordinates and outputs the corresponding target data value at that position (e.g.,$f(x, y, t) = (r, g, b)$) to represent complex signals such as audio [32], images [10, 11], and even 3D scene [3, 28]. In the domain of video compression, recent works such as [5, 6, 13, 14, 20, 21, 23, 26] have shown comparable or even better compression performance than conventional codecs. To represent large data such as high-resolution or long video sequences, these works can work by increasing the number of parameters (i.e., investing more bits). However, simply just enlarging their models proportional to the data size faces practical issues. Fig. 2 shows the impracticality of using the large monolithic INR model[1]: as the video length extends, the required computation (GMac) drastically increases which leads to unacceptable decoding fps even on a high-end GPU (NVIDIA RTX 3090). This also leads to significant startup delay when used in video streaming because the entire model should be downloaded to access a single frame.

Scaling INR for Large-Scale Data. To address the challenges of scaling INR, utilizing small INR has been actively adopted in INR for image reconstruction [18] and static 3D scenes [30, 31, 37, 39]. These works spatially partition the data into multiple small INRs and have proven effective for static domains—ENRP [18] uses independent models on image patches, KiloNeRF [31] and Rebain et al. [30] split 3D scenes into chunks for faster rendering, and Mega-NeRF [39] and Block-NeRF [37] scale to city-sized data. A similar chunking strategy applies to video by slicing along the time axis: splitting a long clip into fixed-length segments keeps per-segment compute and startup delay constant (Fig. 2). However, treating each segment as an independent model discards temporal redundancy and limits compression gains. NIRVANA [26] has attempted to utilize this redundancy to reduce the encoding time of video, but not for compression, and the compression efficiency remains the same. In contrast, NerVast combines chunk-wise partitioning with lightweight parameter sharing, exploiting inter-chunk similarity to improve compression efficiency without incurring additional training or inference overhead.

Parameter sharing is a technique that allows sharing of

[1]We use NerV [5] for the experiment. To preserve the performance in quality, we increase the model size proportional to the video length maintaining 5 Mbps (typical HD video bitrate). To measure the startup delay, we assume 10 Mbps bandwidth.



(a) GMac vs. video length **(b)** Decoding fps vs. video length **(c)** Startup delay vs. video length

Figure 2. The impracticality of a large monolithic INR model for its real-world deployment.

parameters between different tasks or models to leverage inter-task knowledge [8, 34, 40, 45, 46]. Furthermore, parameter sharing can be leveraged to achieve greater parameter efficiency. In the neural-enhanced video streaming, SRVC [19] and [24] have utilized parameter sharing between chunk-specific super-resolution networks to reduce the number of parameters required for streaming. However, these approaches are specific to the super-resolution task and require sequential training of the models with parameter freezing, limiting the possibility of better optimization of the INR video regression task.

In neural representation, the number of parameters directly translates to the size of the data representation in bits. Therefore, recent works adopt parameter sharing for efficient and compact data representation. For instance, [11] introduces modulation networks (∼1 K parameters) as image-specific (32x32 CIFAR 10) networks while sharing the base network weights (∼3.8 M parameters) trained on a general dataset. This approach directly minimizes the required data volume for representing each specific image. TINC [43] proposes a layer-wise parameter sharing architecture that utilizes hierarchical tree-based sharing to improve representation accuracy, while [7] shares executable sub-networks to represent different levels of data quality. Recent INR-focused approaches such as Parameter-Reuse-INR [12], SIEDD [2], and SR-NeRV [16] extend this idea. However, these methods either reuse entire layers [12], rely on a specialized global encoder paired with per-segment decoders [2], or share discrete embedding tokens across frames [16], all of which require specialized architectural components and do not explicitly exploit inter-chunk similarity. In this work, we study parameter sharing as a means to capture and share the long-term redundancy in a video.

3. Method

Our goal is to enhance the compression efficiency of video neural representations when a video is represented in multiple models for its scalability and data transfer. In other words, we target to reduce the overall volume of the video neural representation with a minimal quality drop in such setting. To achieve this, we start from segmenting a video V into N separate chunks ($V = \{C_n\}_{n=1}^{N}$), where each chunk consists of a fixed number of frames.[2] Each video chunk will eventually be encoded to its corresponding small INR model. To reduce the total volume (i.e., the number of parameters) of INR, we take parameter sharing as our high-level approach.

Problem definition. Given video chunks $C_1, C_2, \cdots C_N$, let the corresponding neural representation model parameter $\theta_1, \theta_2, \cdots, \theta_N$. θ_m^j indicates the j'th parameter in m'th chunk model. The models have the identical size of $S = ||\theta||_0$, and we define κ which denotes the target portion of shared parameters between each chunk. Simply, when κ is applied, it reduces $\kappa \times S$ of size for each model starting from the second chunk, which accomplishes our first goal: reducing the total volume (by $\kappa\,(n-1) \times S$). Next, to minimize the quality drop that may be imposed by sharing, we formulate our objective as follows:

$$min_\theta \sum_n \mathcal{L}_n(C_n;\theta_n),\ \ s.t.\ ||\theta_l - \theta_m||_0 \leq (1-\kappa)\cdot||\theta||_0\ ,\ \forall\, l, m \tag{1}$$

where $L_n(C_n;\theta_n)$ is loss of n'th chunk model for video chunk C_n under parameter θ_n. The objective minimizes the quality drop, by minimizing the joint loss.

Challenges & approach. To represent a video $V = \{C_n\}_{n=1}^{N}$, it requires overfitting model parameters θ_n to its given video data chunk C_n. This results in distinct parameter values across models ($\theta_m \neq \theta_n$) where we cannot ensure at least κ portion of parameters that have the "identical" parameters to be shared. As a result, we need a way to explicitly control the parameter values 1) to force the shared parameters to be updated towards the same values, and 2) to make the non-shared ones trained towards the values that best represent their corresponding model's assigned data C_n. Similarly, a number of works utilize a method called "Masking" to explicitly distinguish and control the parameter while training but for different purposes: they select and control parameters to make them zero for pruning [9, 29] or freeze their values for transfer learning [15, 35, 44].

Inspired by the idea, NerVast first creates a mask $M = \{m^i |\ m^i = \mathbf{I}_{Shared}(i)\}$, where $I_{Shared}(i) = 1$, if $\theta_m^i = \theta_l^i$

[2] This baseline setting is commonly employed in INR works that address scalability. Typical video streaming applications (e.g., YouTube) use 4-10 second video chunks.

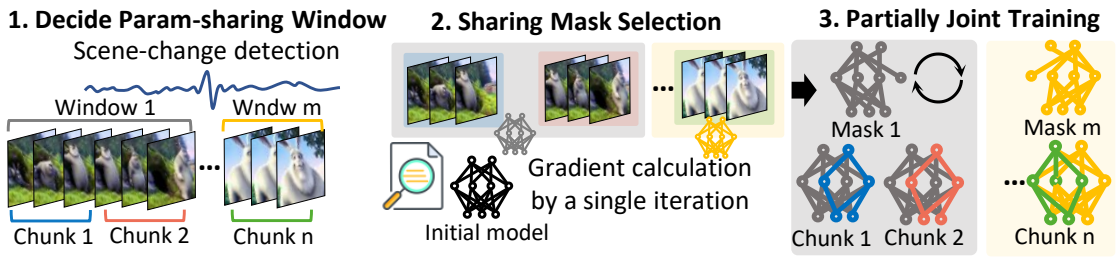


Figure 3. Overall workflow: When multiple video chunk data is given, (1) NerVast decides the parameter sharing window that includes temporally adjacent similar chunks, (2) finds the sharing mask within the parameter sharing window. (3) Finally, it trains multiple models to find the optimal weights for video representation under the new constraint of parameter sharing.

to indicate whether the i'th parameter is shared. Then, it trains the parameter shared models $\theta_1, \theta_2, \cdots, \theta_n$ with the chosen mask. The approach still leaves the following questions for each process: 1-1) For how long in a video should we share the parameters across models? Using a single sharing mask throughout the whole video sequence is not reasonable, and we define a parameter-sharing window in §3.1. 1-2) Which parameters should we select as our sharing mask? Careful selection of the shared parameters is necessary and we provide an effective selection method in §3.1, which requires just a single iteration for calculating gradients. 2) How should we train (encode) the shared parameters & non-shared ones? We propose a partial joint training algorithm (§3.2) that effectively satisfies our objective goal (1). The overall workflow of NerVast is illustrated in Fig. 3.

3.1. Shared Parameter Selection.

Param-sharing window. In scenarios where the content within a video dynamically changes over time, the use of a single sharing mask for an extended duration becomes impractical. To address this challenge, it is necessary to adapt the sharing mask as the redundancy of videos changes. We propose a simple technique that clusters video chunks based on explicit data relationships between frames. When a video requires encoding, we establish a chunk clustering window using the threshold-based scene change detection method commonly employed in video codecs [41]. We measure the difference of neighboring frames and select frame pairs that show significant difference values from adjacent difference values. In addition, to avoid excessively short clusters, we apply a minimum bound for window length. By adopting this method, we group video chunks together that exhibit redundancy with negligible overhead, enabling the shared parameters within each cluster to achieve enhanced compression efficiency (§4.5).

Parameter selection. Once the parameter-sharing window is determined, the next essential step is to decide the sharing mask M. However, optimal parameter selection involves tracing all possible parameter combinations until the end of the training, which is infeasible to accomplish. To address this challenge, we propose the use of Fisher information approximation (Eq. 2) as a metric for selecting shared parameters, bypassing the need for exhaustive empirical training.

$$\hat{F}_\theta = \frac{1}{N}\sum_n \mathrm{E}_{y\sim p_\theta(y|x_i)}(\nabla_\theta log p_\theta(y|x_i))^2 \qquad (2)$$

Fisher information can represent the influence of parameter changes on the model output, to be utilized in network pruning [25, 38], information measure [1] or deciding update parameter in transfer learning [35]. To minimize the possible quality drop due to our parameter sharing constraint, we selected the parameters that least affect the loss according to the Fisher information score to minimize quality degradation. As a result, it can be leveraged as a viable proxy metric for capturing the redundancy within video chunks. In practice, we compute approximated Fisher information [35] for the parameter by calculating the gradient of the whole dataset and employing it for sharing mask selection. As shown in §4.5, our parameter selection approach using Fisher information achieves a minimal quality drop compared to various other proxy metrics.

3.2. Encoding Procedure

Problem definition & objective. After deciding the parameter-sharing mask, our goal can be narrowed down to maximize the quality under the given mask M. Thus the problem changes to minimize objective in equation 1 with changed constraint, $\theta^i_m = \theta^i_l$, if $M^i = 1$. To solve this problem with the gradient descent algorithm, we calculate the gradient of each INR parameter, θ_m as follows:

$$\nabla_{\theta^i_m}(\sum_n L_n(C_n;\theta_n)) = \sum_n \nabla_{\theta^i_m} L_n(C_n;\theta_n) \qquad (3)$$

For shared parameters, we can derive the above gradient as $\nabla_{\theta^i_m} L_n(C_n;\theta_n) = \nabla_{\theta^i_n} L_n(C_n;\theta_n)$. This is because of the mask constraint forces that $\theta^i_m = \theta^i_n$ for shared parameters. On the other hand, for non-shared parameter $\nabla_{\theta^j_m} L_n(C_n;\theta_n) = 0 \; for \; n \neq m$ since chunk specific parameter does not affect the other chunks models' output.

Finally, we can reconstruct the gradient descent algorithm for a partial parameter shared model as like below:

$$\theta^i_m \leftarrow \theta^i_m - \alpha \sum_n \nabla_{\theta^i_n} L_n(C_n), \quad \theta^j_m \leftarrow \theta^j_m - \alpha \nabla_{\theta^j_m} L_m(C_m) \tag{4}$$

where θ^i_m, θ^j_m are shared and non-shared parameters in m'th chunk model and α is learning rate. This can be intuitively explained, that the shared parameter is updated toward minimizing joint loss to capture the common information across video chunks, while the non-shared parameters are updated towards separate directions that are optimal for chunk-specific loss. We can obtain $\sum_n \nabla_{\theta^i_n} L_n(C_n)$ with forward & backward each chunk models and accumulate gradients. By incorporating joint loss for shared parameters and chunk-specific loss for non-shared parameters, we effectively minimize the total loss associated with parameter sharing across multiple INRs.

Partial joint training algorithm. NerVast employs Algorithm 1 to facilitate the partial joint loss above to train the multiple INRs. The algorithm first iterates over the video chunks and calculates the parameter gradients through forward & backward propagation (lines 3-4). Throughout the iteration, the algorithm updates the non-shared parameters that embrace the specific features of each video chunk (line 5). Simultaneously, it accumulates the gradient for the shared parameters until the completion of the iteration (line 6). At the end of the iteration, the accumulated gradient for the shared parameter is updated accordingly (line 8).

Algorithm 1 Partially joint training

Require: θ, C, M
$\theta = \{\theta_1, \theta_2 \cdots \theta_n\}$; multiple INRs for each video chunk, $C = \{C_1, C_2 \cdots C_n\}$; set of video chunks, M; mask for shared parameters

1: Randomly initialize θ
2: Synchronize $\theta^{share} = M \odot \theta_m$
3: **for** $C_i \in C$ **do**
4: calculate $\nabla_{\theta_i} L(C_i; \theta_i)$ ▷ Calculate chunk-wise gradient
5: $\theta_i \leftarrow \theta_i - \alpha \cdot (1 - M) \odot \nabla_{\theta_i} L_i(C_i; \theta_i)$ ▷ Update gradient for non-shared parameters
6: $shared\_grad \mathrel{+}= M \odot \nabla_{\theta_i} L_i(C_i; \theta_i)$ ▷ Accumulate gradient for shared parameters
7: **end for**
8: $\theta^{share} \leftarrow \theta^{share} - \alpha \cdot shared\_grad$ ▷ Update accumulated gradient $\sum_n \nabla_{\theta^{share}_n} L_n(C_n; \theta_n)$

By using the partial joint training, NerVast achieves superior video representation quality compared to SRVC [19], a method that solely adjusts the non-shared parameters after freezing the shared parameters (§4.5).

4. Evaluation

4.1. Evaluation Settings

Datasets. We use the sampled frames (720x1280, 132 frames, 5sec) from "Big Buck Bunny" video and UVG frames (1080x1920, 3900 frames, 2 mins) from UVG dataset [27] which is used in NeRV [5], ENeRV [23] and HNeRV [6]. In addition, to show NerVast can effectively support long videos, we use "Elephant Dream" (1080x1920, 1440 frames, 1min) from the Xiph dataset. For long video evaluation, we also sample more HD frames from "Big Buck Bunny" video (1080x1920, 7200 frames, 5 mins).

Neural representation models. We utilize medium-sized NeRV [5], ENeRV [23], and HNeRV [6] models of ∼6M parameters unless otherwise noted. We train each model for 300 epochs with a learning rate of $5e^{-4}$ and a cosine annealing learning rate scheduler, following the original papers [5, 6, 23].

Baselines. Previous studies [18, 26, 31, 37] utilize multiple small independent models to improve scalability. We consider this approach as a baseline and name it "Naive-split". Unless otherwise noted, we chunk the video into 5-second segments according to the chunk size for HTTP adaptive streaming [33], and use an INR model (chunk model) for each chunk. Specifically, we use 12 chunks for "Elephant Dream" where each chunk model contains 120 frames, considering the "Elephant Dream" is 24fps (120 frames = 24fps x 5sec). Similarly, 26 chunk models are used for the UVG videos, and 60 models for "Big Buck Bunny".

4.2. Compression Efficiency of NerVast

Effectiveness across diverse video durations. Table 1 shows that NerVast consistently achieves substantial compression across videos of varying lengths, from short clips such as Shake (300 frames, from UVG) to long sequences such as Big Buck Bunny (7200 frames). Across all datasets, NerVast reduces parameter volume by 35–46% compared to the naive-split baseline, while maintaining nearly identical reconstruction quality (within 0.2dB PSNR and comparable MS-SSIM). These consistent gains highlight that the proposed parameter-sharing strategy effectively removes redundancy and adapts well regardless of video duration by encoding temporally repetitive information.

Effectiveness across videos of diverse dynamics. To examine whether motion dynamics influence compression efficiency, we compare two representative cases: Honeybee (low temporal variation) and Jockey (high temporal variation), as shown in Table 5. Despite large differences in motion dynamics, NerVast achieves over 42% Bjøntegaard Delta Bit Rate (BD-Rate) [4] reduction in both cases, with slightly larger gains observed for the more static Honeybee video. In contrast, BD-PSNR values differ more sub-

Dataset	Method	Volume (# of params)	PSNR	MS-SSIM
Shake	Naive-split	13.14M	34.96	0.9513
(300 frames)	NerVast	**8.54M (↓35%)**	34.76	0.9516
Elephant Dream	Naive-split	78.8M	39.64	0.9847
(1440 frames)	NerVast	**42.7M (↓45.8%)**	39.57	0.985
UVG dataset	Naive-split	170.8M	34.60	0.95
(3900 frames)	NerVast	**108.4M (↓36.5%)**	34.50	0.95
Big Buck Bunny	Naive-split	394.2M	38.11	0.981
(7200 frames)	NerVast	**246.4M (↓37.5%)**	38.99	0.984

Table 1. Compression over different videos. Volume, quality of Naive-split and NerVast over videos with different durations. NerVast compresses total volume by 36% with a marginal quality drop, or even better quality compared to the Naive-split. Additional encoding time (7%) exists due to our joint training algorithm, while GMac and decoding time are identical to the baseline.

Method	PSNR	Compression Rate
Naive-split (6M)	34.60	0%
Naive-split (3M)	32.57	48.8%
NerVast (κ=0.5)	**34.50**	**37.5%**
NerVast (κ=0.7)	**34.14**	**52.5%**
NerVast (κ=0.9)	**32.78**	**67.5%**

Table 2. Compression over different κ. The trade-off between volume and quality according to the different κ values over the UVG dataset. Larger κ indicates more shared parameters for a high compression rate, at the expense of lower quality.

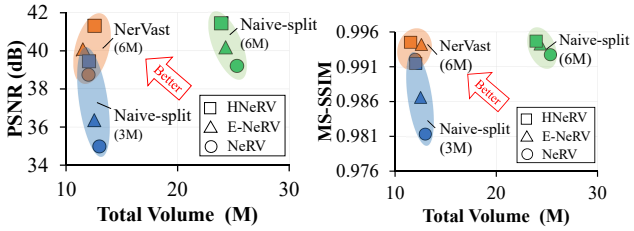


Figure 4. Performance of NerVast with different NeRV variants on the Big Buck Bunny dataset. Colors denote methods and marker shapes denote base models (NeRV[5], E-NeRV[23], HNeRV[6]). NerVast consistently improves video quality while reducing total model volume, demonstrating seamless applicability across architectures.

Metric	**Jockey**	**HoneyBee**
Temporal Information (Mean-avg.)	12.60	2.16
Temporal Information (Optical Flow)	4.25	0.15
BD-Rate [%]	-42.26	-43.81
BD-PSNR [dB]	2.106	0.524

Table 5. Robustness of NerVast across static (HoneyBee) and dynamic (Jockey) videos from the UVG dataset.

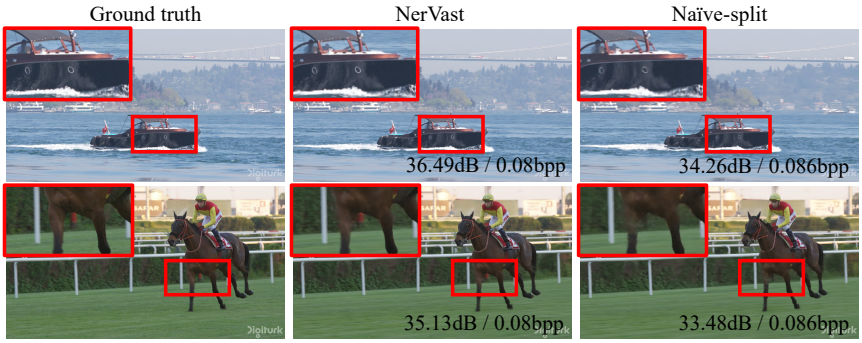
stantially across videos, as the absolute quality scale inherently depends on video content. This indicates that the proposed parameter-sharing strategy adapts robustly across both static and dynamic scenarios.

Effectiveness across NeRV variants. Fig. 4 demonstrates the applicability of NerVast to three different INR models: NeRV, E-NeRV, and HNeRV, using the Big Buck Bunny video as a test case. Across all variants, NerVast consistently improves compression efficiency. Compared to the "Naive-split (6M)" baseline that uses medium-sized chunks (6M parameters each), NerVast achieves smaller model volume with negligible quality loss. Moreover, relative to the "Naive-split (3M)" baseline that employs smaller chunks with a parameter budget comparable to NerVast, our method yields superior quality gains of +1.87dB to +3.74dB in PSNR. These results highlight that NerVast is not only effective across different NeRV variants but also broadly applicable to future INR models.

Comparison with NIRVANA [26]. NerVast and NIRVANA pursue fundamentally different objectives: NerVast explicitly targets compression efficiency through cross-chunk parameter sharing, whereas NIRVANA aims to accelerate training (encoding) by sequentially fitting INR models. As shown in Table 3, NIRVANA achieves 34.71 dB at 0.32 bpp, while NerVast reaches nearly identical quality (34.50 dB) at only 0.10 bpp (↓67%), indicating far superior compression efficiency. While NerVast incurs higher encoding time, this is essentially a one-time cost that can be amortized, whereas compression efficiency directly affects recurring storage and transmission costs. Accordingly, our design places emphasis on reducing recurring costs, which are more critical for practical downstream deployment.

4.3. Computation Analysis

NerVast trades a small, amortizable encoding overhead for large compression savings while leaving decoding cost (compute/fps) unchanged. Figure 6 compares total encoding time and decoding throughput for Monolithic and Naive-split, and decomposes NerVast into three stages: 1) param-sharing window discovery, 2) parameter selection, and 3) partial joint training. Relative to Naive-split, NerVast requires only a marginal increase in encoding time to reach the same final quality (+7.5%: 30.60 s vs. 28.44 s), whereas the monolithic model is far heavier (54.9 s) and decodes much slower (≈17.7 fps vs. ≈48 fps). In the breakdown, *chunk model training* accounts for most of the overhead, with window discovery (0.3%) and gradient accumulation (5%) contributing smaller shares. Despite a modest 7% increase in encoding time, NerVast delivers much higher compression (≈48%) with no impact on decoding speed, which matters for repeated delivery.



Method	PSNR (dB)	BPP
SIREN	27.20	0.28
NIRVANA	34.71	0.32
NerVast (Ours)	34.50	0.10 (↓67%)

Table 3. Compression efficiency compared with SIREN [32] and NIRVANA [26].

Figure 5. Qualitative comparison between NerVast and Naive-split.

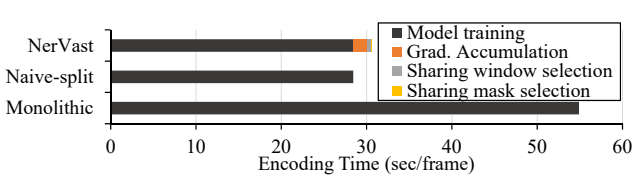
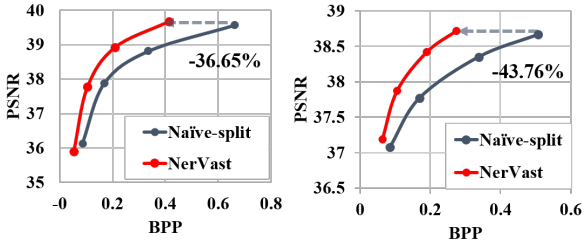


Figure 6. Detailed analysis of encoding time. Despite a modest 7% increase in encoding time, NerVast delivers much higher compression (↓48%) with no impact on decoding speed (48 fps) compared to the naive-split. The monolithic model incurs the highest encoding time and the slowest decoding.



(a) RD curve vs. model size **(b)** RD curve vs. video chunk length

Figure 7. RD curves for NerVast vs. Naive-split under (a) model size (knob 1) and (b) chunk length (knob 2). NerVast achieves BD-Rate reductions of 36.65% and 43.76%.

4.4. NerVast with Different Configurations

NerVast can adjust the bitrate of video with three key parameters: 1) the INR model size for each video chunk, 2) the length of the video chunk, and 3) the parameter sharing rate. We examine the performance of NerVast under varying configurations using the UVG dataset [27].

Model size. Fig. 7(a) reports compression results across model sizes (knob 1) using static (HoneyBee) and dynamic (Jockey) video from the UVG dataset. As the model size increases, total bits per pixel (BPP) also increases. In both static and dynamic scenarios, NerVast mitigates quality drops and achieves lower BPP than Naive-split, yielding a better rate–distortion (RD) curve and average BD-Rate reduction of 36.65%.

Chunk length. Fig. 7(b) evaluates the effect of chunk length (knob 2) on compression efficiency using static (HoneyBee) and dynamic (Jockey) video from the UVG dataset. We sweep lengths from 1.6 to 10 seconds per video chunk. Shorter chunks require more models and thus increase total BPP (with modest quality gains), whereas longer chunks reduce BPP. Across both dynamic and static settings, NerVast consistently outperforms Naive-split. NerVast achieves reductions of 43.76% in BD-Rate relative to Naive-split.

Sharing portion κ. NerVast offers the ability to control the trade-off between total volume and quality by manipulating the parameter sharing portion, κ. Table 2 shows that a larger value of κ leads to increased compression at the expense of quality degradation. Notably, when compared to the Naive-split approach utilizing smaller model sizes for compression, NerVast achieves superior quality ($\geq$1.5dB) even with a higher compression rate ($52.5\% \geq 48.8\%$).

Furthermore, there is another noticeable trend in the experiment. Videos with a stable nature, exhibiting more redundant information between adjacent chunks (e.g., Honeybee), demonstrate relatively smaller quality drop ($\sim$ 0.5 dB). Conversely, dynamic videos (e.g., Jockey) exhibit larger differences between chunks, resulting in a higher quality drop ($\sim$ 2.6 dB). It shows that there is an opportunity to aggressively compress the video through NerVast when a greater amount of redundancy exists.

In summary, our findings demonstrate that NerVast consistently achieves higher compression efficiency than the Naive-split method across various settings.

NerVast with pruning. Although NerVast reduces the number of parameters, it is different from pruning. NerVast shares parameters containing redundant information across the chunk models, while pruning selects parameters containing less information for each chunk model. Therefore, the parameters to be pruned and shared parameters are orthogonal. NerVast is orthogonal to model pruning and can be effectively combined with it. Fig. 8 demonstrates this. Pruning significantly reduces the number of parameters of NerVast with a very minor impact on quality ($\leq$0.2dB drop for 20% sparsity, similar to the original NeRV model).

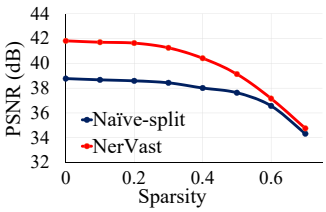


Figure 8. Pruning result.

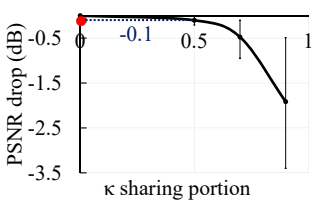


Figure 9. Effect of κ on quality.

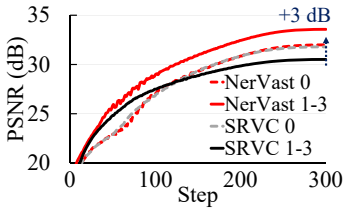


Figure 10. Partially joint training.

	PSNR	Volume
No-share	33.60	13.44M
Random	32.56	8.4M
Variance	32.63	8.4M
Difference	32.65	8.4M
NerVast	**33.16**	**8.4M**

Table 4. Sharing mask selection.

4.5. Ablation Studies

Parameter-sharing window. To see the impact of the parameter-sharing window, we divided "Big Buck Bunny" 1440 frames into 45 frame-length video chunks and applied the scene detection method. It results in configurations of 4 parameter sharing windows, each window consisting of $\{4, 10, 10, 8\}$ chunks. The utilization of the window yields a significant quality improvement, resulting in an increase of approximately 0.6 dB compared to sharing parameters across the entire 32 chunks. By setting the window differently, we effectively avoid sharing parameters between chunks that are less likely to exhibit redundancy.

Sharing portion κ selection. As evidenced by the results presented in Table 2, we observe a clear trend where the quality decreases as the shared portion is increased to reduce the total volume. We further show the tradeoff by varying the value between 0 and 0.9, sweeping the entire space. We report the quality drop from sharing compared to the Naive-split. Fig. 9 shows that the shared fraction of approximately 50 % yielded a negligible quality drop $\leq$ 0.1 dB, while significantly reducing the total volume. Thus, we used $\kappa = 0.5$ for our main evaluation. Notice that there exists an opportunity to further reduce the total volume by applying a larger κ, based on the user's preference.

Sharing mask selection. Table 4 presents results, highlighting how different masking methods affect the overall quality under the same sharing rate. Specifically, we conduct an experiment considering a sharing portion of approximately 50% for the UVG Jockey dataset. The "Difference" method [19, 22] observes how much the parameter changed in a single epoch of training by calculating the absolute difference between the training steps. On the other hand, the "Variance" selects parameters that exhibit the least variance across models after single training epochs. Out of all the selection schemes, our parameter selection method exhibits the least amount of quality degradation (0.5 dB higher than the second best) showing the effectiveness of NerVast masking method. The result proves that careful parameter selection is essential to mitigate the quality drop and maximize the benefit of parameter sharing.

Partially joint training. In Fig. 10, we compare the training curve over time between "Sequential training" and our "Partially joint training" under the same parameter-sharing ratio κ. SRVC [19] sequentially trains chunk-wise neural networks to exploit temporal redundancy in video datasets. For the first chunk, both sequential training and joint training yield comparable quality because all parameters have complete freedom to optimize. However, in subsequent chunks, the quality of sequentially trained models decreases due to the frozen parameters that were optimized using previous chunk data. Notice that NerVast's training method achieves the same quality much faster, and with the same training budget, it yields 3 dB higher in quality than the sequential training. Sequential training involves freezing parameters to enforce identical values in specific portions of the network. This freezing technique restricts the optimization space within gradient descent and may not be well-suited for the context of INR, where overfitting is desired to achieve optimal compression performance.

5. Conclusion

In this paper, we point out the scalability challenges associated with representing the video using neural representation. To overcome this problem, we introduce NerVast that efficiently compresses videos into multiple small INRs. By exploiting the strong temporal redundancy present in videos, NerVast splits the video into multiple small INR models and carefully selects shared parameters to capture the redundancy across video chunks. First, NerVast decides the parameter-sharing window using a scene change detection method. Next, it identifies the suitable parameters, employing Fisher information score. Finally, through partial joint training, it achieves the balance between capturing temporal redundancy and accurately representing each video chunk. We demonstrate significant gains in compression by reducing the parameter count required for representation while maintaining the video quality compared to Naive-split method. Consequently, NerVast enables compact scalable video INR to the large video with small computation and transfer costs.

Acknowledgment. We thank anonymous reviewers for providing helpful feedback and suggestions to improve our work. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.RS-2024-00340099) and was supported by INHA UNIVERSITY Research Grant. Jaehong Kim and Dongsu Han are the corresponding authors.

References

[1] Alessandro Achille, Giovanni Paolini, and Stefano Soatto. Where is the information in a deep neural network? *arXiv preprint arXiv:1905.12213*, 2019. 4

[2] Anirudh Rangarajan, Pulkit Tandon, Kedar Kshitij Patil, and Ser–Nam Lim. SIEDD: Shared-implicit encoder with discrete decoders. *arXiv preprint arXiv:2506.23382*, 2025. 3

[3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 1, 2

[4] Gisle Bjontegaard. Calculation of average psnr differences between rd-curves. *ITU SG16 Doc. VCEG-M33*, 2001. 5

[5] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser Nam Lim, and Abhinav Shrivastava. Nerv: Neural representations for videos. *Advances in Neural Information Processing Systems*, 34:21557–21568, 2021. 1, 2, 5, 6

[6] Hao Chen, Matt Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. Hnerv: A hybrid neural representation for videos. *arXiv preprint arXiv:2304.02633*, 2023. 1, 2, 5, 6

[7] Junwoo Cho, Seungtae Nam, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Streamable neural fields. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XX*, pages 595–612. Springer, 2022. 1, 3

[8] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3150–3158, 2016. 3

[9] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017. 3

[10] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 1, 2

[11] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. Coin++: Data agnostic neural compression. *arXiv preprint arXiv:2201.12904*, 2022. 1, 2, 3

[12] Fan Zhang, Ho Man Kwan, Ge Gao, Andrew Gower, and David Bull. Releasing the parameter latency: Layer-wise parameter reuse for implicit video coding. *arXiv preprint arXiv:2410.01654*, 2024. 3

[13] Ge Gao, Ho Man Kwan, Fan Zhang, and David Bull. PNVC: Towards practical INR-based video compression. *arXiv preprint arXiv:2409.00953*, 2024. 2

[14] Ge Gao, Siyue Teng, Tianhao Peng, Fan Zhang, and David Bull. GIViC: Generative implicit video compression. *arXiv preprint arXiv:2503.19604*, 2025. 2

[15] Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*, 2020. 3

[16] Taiga Hayami, Kakeru Koizumi, and Hiroshi Watanabe. Sr-nerv: Improving embedding efficiency of neural video representation via super-resolution. *arXiv preprint arXiv:2505.00046*, 2025. 3

[17] 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. *arXiv preprint arXiv:2303.14124*, 2023. 1

[18] Milad Soltany Kadarvish, Hesam Mojtahedi, Hossein Entezari Zarch, Amirhossein Kazerouni, Alireza Morsali, Azra Abtahi, and Farokh Marvasti. Ensemble neural representation networks. *arXiv preprint arXiv:2110.04124*, 2021. 2, 5

[19] Mehrdad Khani, Vibhaalakshmi Sivaraman, and Mohammad Alizadeh. Efficient video compression via content-adaptive super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4521–4530, 2021. 3, 5, 8

[20] Ho Man Kwan, Ge Gao, Fan Zhang, Andrew Gower, and David Bull. NVRC: Neural video representation compression. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024. arXiv:2409.07414. 2

[21] Joo Chan Lee, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Ffnerv: Flow-guided frame-wise neural representations for videos. *arXiv preprint arXiv:2212.12294*, 2022. 1, 2

[22] Xiaoqi Li, Jiaming Liu, Shizun Wang, Cheng Lyu, Ming Lu, Yurong Chen, Anbang Yao, Yandong Guo, and Shanghang Zhang. Efficient meta-tuning for content-aware neural video delivery. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVIII*, pages 308–324. Springer, 2022. 8

[23] Zizhang Li, Mengmeng Wang, Huaijin Pi, Kechun Xu, Jianbiao Mei, and Yong Liu. E-nerv: Expedite neural video representation with disentangled spatial-temporal context. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXV*, pages 267–284. Springer, 2022. 2, 5, 6

[24] Jiaming Liu, Ming Lu, Kaixin Chen, Xiaoqi Li, Shizun Wang, Zhaoqing Wang, Enhua Wu, Yurong Chen, Chuang Zhang, and Ming Wu. Overfitting the data: Compact neural video delivery via content-aware feature modulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4631–4640, 2021. 3

[25] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021. 4

[26] 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. *arXiv preprint arXiv:2212.14593*, 2022. 1, 2, 5, 6, 7

[27] 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*, page 297–302, New York, NY, USA, 2020. Association for Computing Machinery. 5, 7

[28] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1, 2

[29] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 3

[30] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14161, 2021. 1, 2

[31] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1, 2, 5

[32] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020. 1, 2, 7

[33] Thomas Stockhammer. Dynamic adaptive streaming over http– standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011. 5

[34] Tianxiang Sun, Yunfan Shao, Xiaonan Li, Pengfei Liu, Hang Yan, Xipeng Qiu, and Xuanjing Huang. Learning sparse sharing architectures for multiple tasks. In *Proceedings of the AAAI conference on artificial intelligence*, pages 8936–8943, 2020. 3

[35] Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021. 3, 4

[36] Vivienne Sze, Madhukar Budagavi, and Gary J Sullivan. High efficiency video coding (hevc). In *Integrated circuit and systems, algorithms and architectures*, page 40. Springer, 2014. 1

[37] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022. 1, 2, 5

[38] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018. 4

[39] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12922–12931, 2022. 1, 2

[40] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014. 3

[41] S Wee. Dynamic threshold method for scene change detection. In *2003 International Conference on Multimedia and Expo. ICME'03. Proceedings (Cat. No. 03TH8698)*, pages 337–340. IEEE Computer Society, 2003. 4

[42] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003. 1

[43] Runzhao Yang, Tingxiong Xiao, Yuxiao Cheng, Jinli Suo, and Qionghai Dai. Tinc: Tree-structured implicit neural compression. *arXiv preprint arXiv:2211.06689*, 2022. 3

[44] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021. 3

[45] Fuzhen Zhuang, Xiaohu Cheng, Ping Luo, Sinno Jialin Pan, and Qing He. Supervised representation learning with double encoding-layer autoencoder for transfer learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(2):1–17, 2017. 3

[46] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020. 3