NerVast: Scaling Neural Video Representation with Enhanced Compression Efficiency

Supplementary Material

A. Experiment details

A.1. Base model hyperparameters

NerVast can be seamlessly integrated with various base models and we demonstrate this using NeRV [1], E-NeRV [3], and HNeRV [2]. To control the bpp of base models, we tune the Convolution Input dimension and Lower width for each of these base models as shown in Tab. 1. For the remaining hyperparameters, we maintain the initial configurations given in the paper's official GitHub pages[1] [2] [3]. The notation (3M) and (6M) indicate the parameters number for the model is 3 million and 6 million.

	NeRV	**E-NeRV**	**HNeRV**
Conv	9 x 16 x 32 (3M)	9 x 16 x 34 (3M)	16 x 64 (3M)
Input dim	9 x 16 x 58 (6M)	9 x 16 x 92 (6M)	16 x 64 (6M)
Lower width	96	96	12

Table 1. Base model hyperparameters of NeRV, E-NeRV, HNeRV models

A.2. Training details

We introduce the partially joint training algorithm for NerVast, updating parameters based on both joint and chunk-specific losses independently. Considering the mini-batch Gradient Descent (GD) optimization where a single mini-batch includes frames from the same chunks, a natural approach would be training N chunk models and synchronizing the shared parameters of these models at every weight update cycle.

However, this approach produces a substantial number of copy operations among chunk models. Each iteration involves updating the shared parameters of N chunk models, resulting in a time complexity of O(N), where N is the number of chunks. Additionally, when aiming to utilize more advanced optimization techniques, like Adam optimization, which leverage momentum for faster convergence, it is essential to carefully manage the momentum of every chunk model and customize the update procedure accordingly.

To address the problems, we implement a partially joint training incorporating (N+1) distinct models. The first is dedicated to training shared parameters (shared model) while the remaining N models focus on training non-shared parameters (chunk models). The procedure for each iteration (update with mini-batch) involving data from chunk i is as follows: 1) Copy non-shared parameters from the i th chunk model to the shared model 2) Calculate the gradient using the shared model 3) Copy the non-shared parameters' gradient to the chunk model 4) Calculate momentum and weight delta, and update both the shared model and chunk model independently. After finishing the training, we copy the shared parameter from the shared model to non-shared models, which generate N chunk models with shared parameters.

Algorithm 1 Partially joint training

Require: Shared model θ, Chunk models $\theta_{\{i\}}$, Mask M, Dataset D

Randomly initialize θ^0
Synchronize $\theta_i^0 = \theta^0$
Initialize momentum $m, m_{\{i\}} = 0$
for e in Epochs **do**
  for $d \in D$ **do**
    $x, y_{gt} = d$
    $i = ChunkIndex(x)$ ▷ (1)
    $\theta.weight = M \cdot \theta.weight + (1 - M) \cdot \theta_i.weight$
    $y_{pred} = \theta(d.x)$ ▷ (2)
    $Loss = L(y_{gt}, y_{pred})$
    $Loss.backward()$
    $\theta_i.grad = (1 - M) \cdot \theta.grad$ ▷ (3)
    $\Delta\theta, m = Adam(\theta.grad, m)$ ▷ (4)-1 shared
    $\theta^t = \theta^{t-1} + \alpha \cdot \Delta\theta^t$
    $\Delta\theta_i, m_i = Adam(\theta_i.grad, m_i)$ ▷ (4)-2 chunk
    $\theta_i = \theta_i + \alpha \cdot \Delta\theta_i$
  end for
end for
for $i \in ChunkIndices$ **do** ▷ (5)
  $\theta_i.weight = M \cdot \theta.weight + (1 - M) \cdot \theta_i.weight$
end for

By centralizing the management of shared parameters in a single model, we reduce the need for copy operations to 2 per iteration: chunk model to shared model and vice versa. Therefore, we reduce the time complexity to O(1) per iteration. Moreover, the individual optimizers for the shared model and chunk models facilitate separate momentum calculations according to parameter types.

The detailed algorithm is described at Alg. 1

[1] https://github.com/haochen-rye/NeRV
[2] https://github.com/kyleleey/E-NeRV
[3] https://github.com/haochen-rye/HNeRV

B. Result on different epochs and sizes

We expand our analysis by testing NerVast with various model sizes and extended epochs. The comprehensive assessment demonstrates that NerVast presents benefits in diverse scenarios. In this experiment, we use 720p, 132 frames from the Big Buck Bunny video, and 4 chunk models.

Epochs	300	600	900	1200
Naive-split (3M)	34.99 dB	37.26 dB	38.23 dB	38.77 dB
NerVast (6M, κ=0.7)	**38.73 dB**	**40.78 dB**	**41.46 dB**	**41.81 dB**

Table 2. Results on different training epochs

We vary the training epoch of both Naive-split and NerVast to demonstrate the quality improvement of NerVast, while using the same number of parameters. Each method employs a total $\sim$ 12M parameters. Tab. 2 demonstrates NerVast achieves superior quality throughout all training epochs.

Model Size	Small (3M)		Medium (6M)		Large (12M)	
	PSNR (dB)	Volume	PSNR (dB)	Volume	PSNR (dB)	Volume
Naive-split	34.99	12.96M	39.22	25.12M	42.04	50.24M
NerVast (κ=0.7)	34.88	**8.1M**	39.05	**12.57M**	41.98	**31.41M**

Table 3. Results on different model sizes

We vary the base model size of both Naive-split and NerVast to demonstrate the influence of the model size on compression efficiency. As shown in Tab. 3, NerVast can effectively reduce the number of parameters with minimal degradation in quality (approximately 0.1dB in PSNR).

C. NerVast with Downstream Tasks

NerVast is a model-agnostic parameter-sharing method that retains the inherent structure of the original model. This property makes it possible for NerVast to robustly execute diverse video tasks, preserving the original model's capability. To validate this preservation, we assess the effectiveness of NerVast in performing a video denoising task and a video inpainting task. In this experiment, we utilized 720p, 132 frames from the bunny video, and 4 chunk models.

C.1. Denoising

The NeRV [1] model denoises videos when the given input is noisy. To demonstrate that NerVast preserves the denoising property of the NeRV model, we add random salt and pepper noise to the bunny dataset and perform training for both NeRV (3M) and NerVast (6M, κ=0.7). Each method uses total $\sim$ 12M parameters. Table 4 shows that NerVast produces better denoising results. We attribute this to the improved video reconstruction quality by NerVast at the same compression rate.

	Noisy	Naive split (3M)	**NerVast (6M)**
PSNR	24.85 dB	31.91 dB	**35.45 dB**

Table 4. Denoising result of the NerVast, using NeRV base model

C.2. Inpainting

The HNeRV[2] model is capable of performing video inpainting, because of its robustness. To demonstrate NerVast's preservation of the HNeRV model's inpainting ability, we add 50x50 pixel masks to the frame, as in the original paper. NerVast's superior inpainting results over the Naive split (3M) are presented in Tab. 5. Similar to denoising tasks, we analyze that this result is attributed to the superior quality of video reconstruction by NerVast at the identical compression rate.

	Naive split (3M)	**NerVast (6M)**
PSNR	36.98 dB	**38.1 dB**

Table 5. Inpaint result of the NerVast, using HNeRV base model

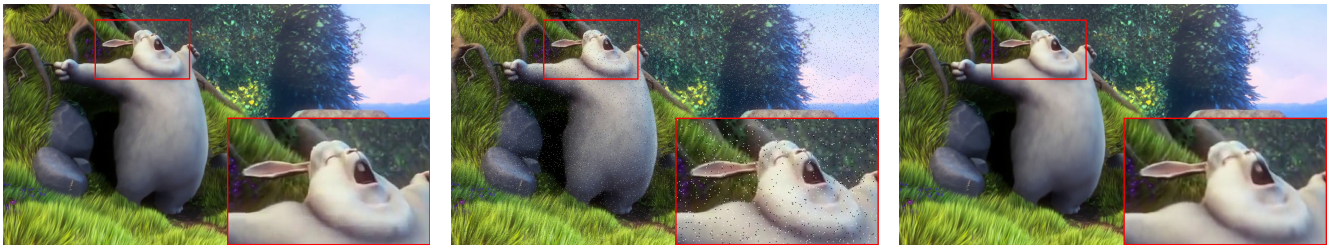
D. Comparison with Monolithic model

In order to realize practical applications, leveraging multiple INR models is necessary. Encoding an entire video sequence within a single monolithic model leads to a substantial increase in model size, affecting encoding, decoding, and transfer processes. The following experiment exemplifies this problem. This increase in computational overheads not only impedes encoding and decoding speeds, preventing real-time decoding even for state-of-the-art models but also introduces unbearable startup delays during streaming scenarios.

Tab. 6 compares two approaches, NerVast and the Monolithic method (a single model without chunking). The monolithic model achieves the best quality with a similar number of parameters. However, it requires approximately twice the training time, lacks real-time decoding capability ($\sim$ 17 fps), and suffers from long startup delays (35 seconds for a 2-3 minute video). These limitations make it impractical for real-world applications. On the contrary, while Naive-split can address these practical issues, it results in decreased compression efficiency and leads to the quality of the video dropping to around $\sim$ 32.57 dB. NerVast, however, achieves a balance by providing faster encoding time and decoding speed ($>$30 fps), along with a reasonable startup delay (a few seconds), while significantly reducing the possibility of sharp quality drops ($\sim$1.5dB higher than Naive-split).

	# of distinct params	GMac	Encoding Time (sec/frame)	Decoding Speed	Startup delay	PSNR(dB)
NerVast (6M, κ=0.7)	82.93 M	228	**30.15 (∼1.8x)**	**40 fps (∼2.2x)**	**2.7 sec (∼13x)**	34.14 dB
Monolithic (87.8M)	87.83M	693	54.9	17.72 fps	35 sec	35.15 dB

Table 6. NerVast achieves much faster encoding, decoding, and startup delay compared to the monolithic method.



(a) Ground truth (b) Noisy frame (c) NerVast

Figure 1. **Denoising** result of salt and pepper noise. We input the noisy frames and NerVast successfully denoises the noisy frames.

(a) Ground truth (b) Masked frame (c) NerVast

Figure 2. **Inpainting** result of fixed mask. We input the masked frames and NerVast successfully inpaints the masked patches.

References

[1] 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

[2] 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

[3] 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. 1