# ABSTRACT


Title of Dissertation: NEURAL NETWORKS AS DATABASES - FROM DATA TO MODEL COMPRESSION

Shishira Raghunath Maiya, 2025

Dissertation Directed by: Professor Abhinav Shrivastava
Department of Computer Science
University of Maryland, College Park



The past decade has witnessed an exponential increase in data and the rise of deep learning systems to handle it. These systems primarily analyze, learn and interpret the data by excelling in exploiting patterns. This leads us to the question - can we use these patterns to efficiently store the data as well ? Essentially this would mean moving from the paradigm of data compression towards model compression. This thesis introduces and investigates a variety of methods in the realm of model compression, with the eventual goal of replacing data compression itself.

To explore this, we start with the most information rich, yet sparse media form - videos. In the first part of this thesis, we introduce a framework for representing videos as continuous functions using Implicit Neural Representations (INRs), which aim to represent any given signal as a mapping between the spatial/temporal coordinate space to its values. We propose an auto-regressive framework that exploits the redundancies of a video for

efficient compression and real time decoding - a first in this field. We then build upon this framework by introducing a shared video prior that captures common patterns across video frames, significantly improving the encoding times by 10-20×, while improving the compression rates.

Despite these value additions, INR's aren't really representations of underlying data unless the resulting model weights also encode some semantics. Based on this intuition, we introduce a hypernetwork-based INR system enables us to perform semantic tasks like video retrieval and understanding along with compression.

Having established that the problem of data compression is actually a model compression problem, I will then present a case study on a widely used model compression method: pruning. We take a popular pruning method - the Lottery Ticket Hypothesis which offers extreme sparsity and study its effects on fundamental vision tasks like classification, detection and segmentation.

Finally, we take a look at the proposed future directions in which we explore enhanced network and algorithmic designs with random networks for greater compression and meta-learning for achieving faster video encoding.

# NEURAL NETWORKS AS DATABASES - FROM DATA TO MODEL COMPRESSION

by

Shishira Raghunath Maiya

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Abhinav Shrivastava, Chair/Advisor
Professor Hernisa Kacorri, Dean's Representative
Professor Christopher Metzler
Professor Pratap Tokekar
Professor Ruohan Gao

DEDICATION

To Amma, Appa and my little brother

for their unwavering love and support.

ಜ್ಞಾನದಿಂದಲಿ ಇಹವು| ಜ್ಞಾನದಿಂದಲಿ ಪರವು|

ಜ್ಞಾನವಿಲ್ಲದಲೆ ಸಕಲವೂ ತನಗಿದ್ದು|

ಹಾನಿ ಕಾಣಯ್ಯ ಸರ್ವಜ್ಞ| |

# Acknowledgments

People often remark - "PhD is a lonely path". I would push back. Every moment of this wonderful journey was shaped by those who mentored, nurtured, protected and pushed me to be the best version of myself. This wild ride of curiosity wouldn't be possible without the immense body of scientific literature and the work of doyens and giants of the field on whose shoulders we all stand. Their perseverance, dedication, rigor, and humility in pursuit of truth continue to inspire me.

The Sanskrit word "Guru" holds multitudes. The root meaning translates to "one who dispels darkness/ignorance" - an ode to transformative power of knowledge. I am forever indebted to my Guru, Prof Abhinav Shrivastava for giving me the opportunity to be under his tutelage. He took a chance on a curious boy with half-formed ideas and helped shape that curiosity into full-fledged enduring ideas. The strong foundation he has laid over these years will support me for the rest of my intellectual life. He has been more than a Professor - a mentor, friend, a brother and more recently a partner in building something bigger than both of us.

In a world inundated with AI slop, *taste* in both consumption and creation is what separates long lasting work from mediocrity. In a fast moving field like ours, this discernment is invaluable. "Research taste" is the art of aligning personal curiosities with meaningful hard problems to do fruitful research and I learned that from Max Ehrlich. His

calm, sagacious outlook on research and the instinct for choosing problems that matter will stay with me as a lifelong lesson.

This thesis wouldn't have been possible without the support of my friends in our lab. I am grateful for all the collaborations, conversations and support I have received from them over the years. A special shoutout to Saksham, Sharath, and Pulkit - labmates turned brothers - and to Sai, for being my home away from home in 4713. I thank Sukriti for her steadfast support, patience and belief in me. You've been my anchor in more ways than one.

I also thank Vishnu for being that friend who, with every conversation, left me with threads of curiosity to unravel. Namitha, for all the Bangalore banter when we missed *ooru*. And Shlok for being the elder brother who held me when I was down. This journey wouldn't have been the same without my amazing batchmates - Manas, Aadesh, Noor, Vasu, Sneha, Samayadeep, Aman, Naman, Pooja, Divya and Neha. A special thanks to the Graduate Hills gang - Navneeth, Suhas, Priyanka, Ashvi, Rachita, Anish, and Patil. You came into my life by chance, but made UMD feel like home.

Additionally, I thank everyone back home who has quietly rooted for my success - especially Sneha, Vishal, and Sudarshan - for their unwavering belief in me, even from afar. And to the quiz boys - Niyam, Nagendra, Tejas, Kishore, Nikhil, Pal, Subba, and Nanda - thank you for keeping my spirits up and my mind sharp through this long, winding trek.

I express my gratitude to the people I had the privilege to work with during my internships - Kwot Sin, Ric Poirson, and Sergey Tulyakov at Snap, and Rui Shen at Apple, for their mentorship, trust, and sharp technical guidance. The lessons I learned from you helped me hone my research instincts and will continue to guide me in the future. I also

want to thank Tom Hurst, Migo Gui and the entire administrative staff at UMIACS and UMD CS for their support to international students and making me feel like their own. I am also grateful to all my mentors at IISc - Dr Raghu Krishnapuram and all the Professors in RIT who nurtured the budding researcher in me and handed me pivotal opportunities at the right time.

Above all, I thank my parents - Raghunath and Vasanthi for being the bedrock of support, nurture, and home in every sense of the word. Thank you for bringing me to this world, raising me in a house full of love and setting me on a path grounded in curiosity and learning. I cannot thank by brother Sharath enough - you have been the one holding things together, taking care of them and shouldering the responsibility. This PhD wouldn't have been possible without your quiet support. I'd also like to thank my Uncle - Ravindranath who had an outsized influence on my life choices, goals and intellectual values. He was a lighthouse throughout my journey, pointing me toward what matters, shaping my taste, and nudging me to rise above the mediocre and pursue what's genuinely interesting.

And beyond people, certain circumstances and events deeply affected me and changed how I perceive the world, especially during times of despair. The ones that stand out are the Indian test series victory against Australia in 2020-21 and RCB finally lifting the trophy after 18 years. These moments taught me about the value of resilience and determination when faced with overwhelming odds, sometimes spanning decades.

To everyone mentioned here, and to those I've missed but carry with me, thank you for being part of this journey.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

*"Compression is*

*Understanding"*

---

*Ilya Sutskever's Ghost*

The invention of the transistor sparked a communication revolution in the world, which eventually led to creation of humongous amounts of data, which has only grown exponentially since then. This growth necessitated the need for algorithms that could efficiently store, retrieve and process this data. Historically data was stored on physical media like tapes, hard drives, CDs with specific compression algorithms - both lossy and lossless forms. However, these algorithms were often hand-crafted for the niche they serve and often did not exploit the underlying structure of the data. The deep learning revolution of the past decade aimed to challenge this paradigm. By training large models that could extract patterns in huge swathes of data, we were finally building efficient priors of the world which we could exploit. But we hit a roadblock - the models were too large to be stored on disk, too slow to decode and too power hungry to be deployed on edge devices. Consequently, traditional codecs like JPEG remains the king of image compression, while HEVC and MP4 continue to dominate video and audio compression respectively. In this work, I aim to revisit this paradox, albeit with a fresh perspective. What if,

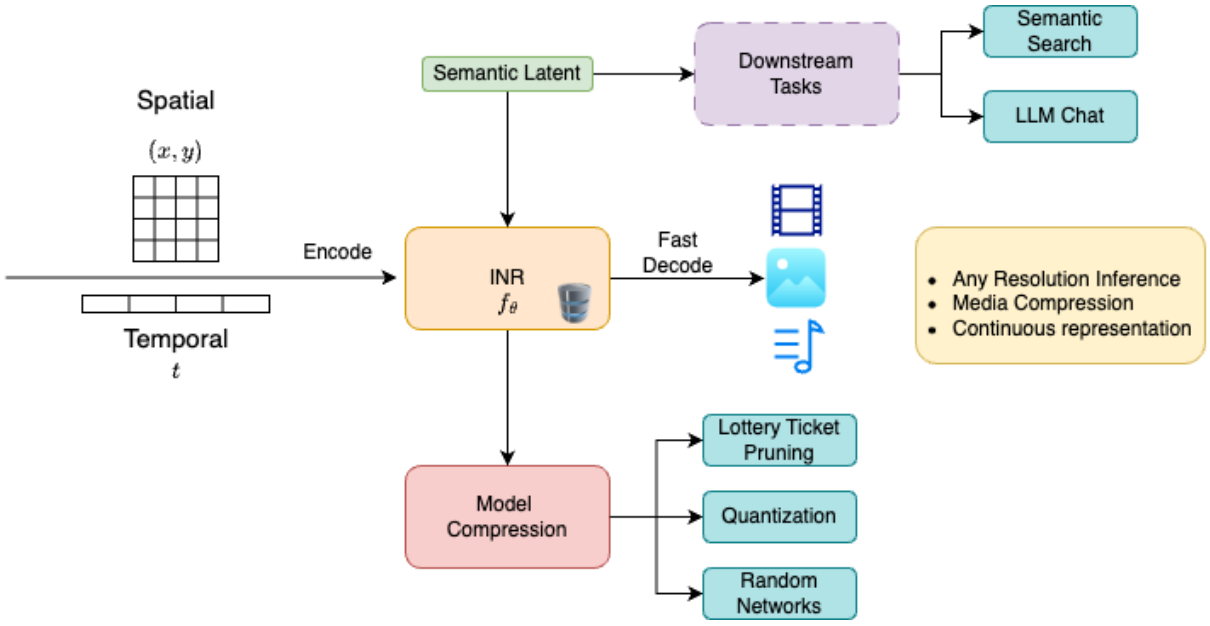



Figure 1.1: The figure shows the general framework of using INRs as efficient databases of media which not only offer compression, but also enable a variety of important semantic tasks.

instead of training large models, we could train much *smaller* models, which capture the local patterns of a particular data point in an implicit manner? From a signal processing standpoint this would mean training a neural network to be a continuous function mapping from the coordinate space to the signal space. Once trained, the network itself becomes our data, effectively transitioning us from a data compression paradigm to one of model compression. These class of networks - known as Implicit Neural Representations can be used to represent *any* kind of data, have excellent decoding speeds, and are also resolution agnostic. Moreover, compressing the data now becomes compressing its INR.

This shift in thinking triggered a series of works that provided a "neural" way of approaching a variety of problems. Works like [1] and [2] tackled image compression using this approach, while [3] developed the first scalable INR for videos. For videos, these existing approaches fell short on many areas. Prohibitively slow encoding times, fixed model architectures that prevent scaling to different video resolutions and inability

to encode videos of any length. In Chapter 2, we develop a scalable framework for Video-INRs that addresses many of these shortcomings. Our proposed method NIRVANA treats videos as groups of frames and fits separate small networks to each group, while performing patch-wise prediction. Each of these networks are trained auto-regressively by initializing from the previous trained model, drastically reducing video encoding times by over 12× compared to previous methods at similar compression levels while improving the encoding quality on benchmark datasets. We also quantize the parameters while training itself, thus not relying on any form of post-hoc pruning or quantization. This allows NIRVANA to achieve variable bitrate compression that adapts based on the video content - a key feature missing in existing methods. In contrast to prior video INR works which struggle with larger resolution and longer videos, we show that our algorithm scales naturally due to its patch-wise and auto-regressive design. NIRVANA also achieves 6× decoding speed and also exhibits linear scaling with GPUs, making it practical for various deployment scenarios.

NIRVANA introduced a new way of thinking about video INR architectures. It had components of a "shared prior" at a video level, while maintaining faithful reconstruction at the frame level. Eventhough this lead to massive improvements in encoding times, it is still far from practical for real-world applications. In Chapter 3, we propose SIEDD, a spiritual successor to NIRVANA, where we borrow the idea of a shared video prior and combine it with sampling and optimization tricks to achieve a 10 - 20× speedup in encoding times over existing methods. It is the first INR based codec that can encode UVG-4K videos under 60 mins with excellent quality and compression rates.

Despite these improvements, INR based methods have a long way to go before they can be considered as credible alternatives for traditional video codecs. One glaring issue

is that these trained networks are not really "representations" of the underlying videos, as we cannot use them for anything apart from faithful reconstruction. In Chapter 4, we propose Latent-INR, an architecture that not only encodes the video, but in the process also creates latents which are used for various downstream tasks like video retrieval, interpolation and video-LLM based chat. In our model we have a dictionary of per-frame latents along with video specific hypernetworks that modulate the INR weights to reconstruct a particular frame. This framework not only retains the compression efficiency, but the learned latents can be aligned with features from large vision models, which grants them discriminative properties. We align these latents with CLIP and show good performance for both compression and video retrieval tasks. By aligning with VideoLlama, we are able to perform open-ended chat with our learned latents as the visual inputs. Additionally, the learned latents serve as a proxy for the underlying weights, allowing us perform tasks like video interpolation. These semantic properties and applications, existing simultaneously with ability to perform compression, interpolation, and super-resolution properties, are a first in this field of work. These collective advancements finally bring in "semantics" into the pipeline of INR based codecs, paving way for compressed yet meaningful video *representations*.

On an orthogonal note, once we are fully entrenched in this paradigm where model compression and data compression are interchangeable, it is important to investigate various techniques. One such area is the interesting field of model pruning and sparsity which involves gains in storage and compute by driving most parameters in a trained model to zeros. In Chapter 5 we present a case study of a popular pruning method - the Lottery Ticket Hypothesis (LTH) and its effects on downstream tasks like object detection, instance

segmentation, and keypoint estimation. The learnings from these empirical experiments can be directly applicable to compressing INRs as well.

This thesis provides a glimpse at a future where neural networks evolve from being analyzers of data, to efficient data warehouses that allow us to interact with them in a friction-less manner. In the final chapter, I discuss some proposed future directions and existing problems with these systems which need to be solved before we can build a real-time, *universal data codec* that is both compressed and semantically meaningful.

# Chapter 2: NIRVANA: A Scalable Framework for Video-INRs

This chapter [1] introduces the paradigm of using compressed models as proxy for the underlying data, using Implicit Neural Representations (INRs). Neural networks are usually used to either model the underlying data distribution or analyze the given data and perform specific predictions. INRs on the other hand construct a coordinate-signal mapping which serves as a continuous function of the data, enabling efficient storage and retrieval. But building such functions to represent video data can be challenging. Existing methods struggle either in fidelity, long encoding times, inability to represent videos of arbitrary length and scale. Our method NIRVANA is designed precisely to avoid these pitfalls. We introduce patch based prediction and an autoregressive design allowing us to encode any length videos with fast encoding times, while achieving compression and greatly improved inference speed.

## 2.1 Introduction

In the information age today, where petabytes of content is generated and consumed every hour, the ability to compress data fast and reliably is important. Not only does compression make data cheaper for server hosting, it makes content accessible to popula-

[1] Joint work with Sharath Girish. I was responsible for primary architecture design, implementation, and experiments. Sharath Girish was responsible for neural entropy coding.

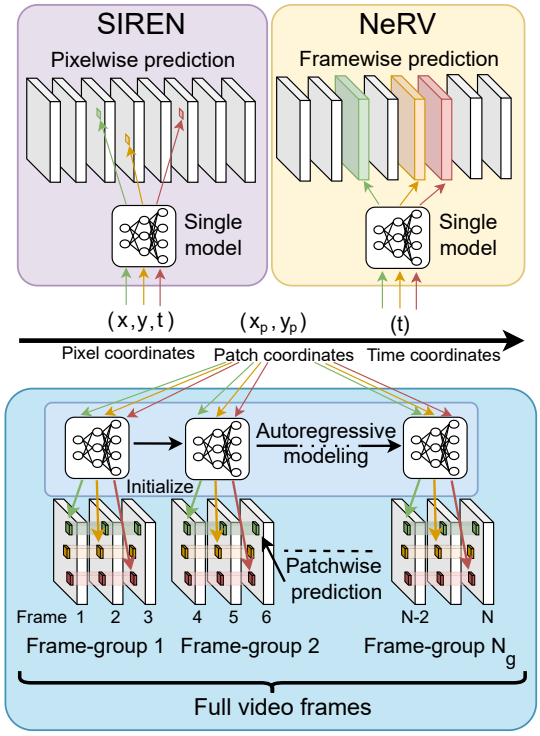



Figure 2.1: **Overview of NIRVANA:** Prior

tion/regions with low-bandwidth. Conventionally, such compression is achieved through codecs like JPEG [4] for images and HEVC [5], AV1 [6] for videos, each of which compresses data via targetted hand-crafted algorithms. These techniques achieve acceptable trade-offs, leading to their widespread usage. With the rise of deep learning, machine learning-based codecs [7, 8, 9, 10] showed that it is possible to achieve better performance in some aspects than conventional codecs. However, these techniques often require large networks as they attempt to generalize to compress all data from the distribution. Furthermore, such generalization is contingent on the training dataset used by these models, leading to poor performance for Out-of-Distribution (OOD) data across different domains [11] or even when the resolution changes [12]. This greatly limits its *real-world practicality* especially if the input data to be compressed is significantly different from what the model has seen during training. In recent years, a new paradigm, Implicit Neural Representations (INR),

emerged to solve the drawbacks of model-learned compression methods. Rather than attempting to generalize to all data from a particular distribution, its key idea is to train a network that specifically fits to a signal, which can be an image [13], video [3], or even a 3D scene [14]. With this conceptual shift, a neural network is no longer just a predictive tool, rather it is now an efficient storage of data. Treating the *neural network as the data*, INR translate the data compression task to that of model compression. Such a continuous function mapping further benefits downstream tasks such as image super-resolution [15], denoising [16], and inpainting [13].

Despite these advances, videos vary widely in both spatial resolutions and temporal lengths, making it challenging for networks to encode videos in a practical setting. Towards solving this task, SIREN [13], attempted to learn a direct mapping from 3D spatio-temporal coordinates of a video to each pixel's color values. While simple, this is computationally inefficient and does not factor in the spatio-temporal redundancies within the video. Later, NeRV [3] proposed to map 1D temporal coordinates in the form of frame indices directly to generate a whole frame. While this improves the reconstruction quality, such a mapping still does not capture the temporal redundancies between frames as it treats each frame individually as a separate image encoding task. Finally, mapping only the temporal coordinate also means one would need to modify the architecture in order to encode videos of different spatial resolutions.

To address the above issues, we propose NIRVANA, a method that exploits spatio-temporal redundancies to encode videos of arbitrary lengths and resolutions. Rather than performing a pixel-wise prediction (*e.g.*, SIREN) or a whole-frame prediction (*e.g.*, NeRV), we predict *patches*, which allows our model to adapt to videos of different spatial resolutions

without modifying the architecture. Our method takes in the centroids of patches (patch coordinates) $(x_p, y_p)$ as inputs and outputs a corresponding patch volume. Since patches can be arranged for different resolutions, we do not require any architectural modification when the input video resolution changes. Furthermore, to exploit the temporal nature of videos, we propose to train individual, small models for each group of video frames ("clips") in an autoregressive manner: the model weights for predicting each frame group is initialized from the weights of the model for the previous frame group. Apart from the obvious advantage that we can scale to longer sequences without changing the model architecture, this design exploits the temporal nature of videos that, intuitively, frame groups with similar visual information (*e.g.*, static video frames) would have similar weights, allowing us to further perform residual encoding for greater compression gains. This *adaptive* nature, that static videos gain greater compression than dynamic ones, is a big advantage over NeRV where the compression for identical frames remain fixed as it models each frame individually. To obtain further compression gains, we employ recent advances in the literature to add entropy regularization for quantization, and encode model weights for each video during training [17]. This further adapts the compression level to the complexity of each video, and avoids any post-hoc pruning and fine-tuning as in NeRV, which can be slow.

Finally, we show that despite its autoregressive nature, our model is linearly parallelizable with the number of GPUs by chunking each video into disjoint groups to be processed. This strategy largely improves the speed while maintaining the superior compression gains of our method, making it practical for various deployment scenarios.

We evaluate NIRVANA on the benchmark UVG dataset [18]. We show that NIRVANA reaches the same levels of PSNR and Bits-per-Pixel (BPP) compression rate with

almost 12× the encoding speed of NeRV. We verify that our algorithm adapts to varying spatial and temporal scales by providing results on videos in the UVG dataset with 4K resolution at 120fps, as well as on significantly longer videos from the YouTube-8M dataset, both of which are challenging extensions which have not been attempted on for this task. We show that our algorithm outperforms the baseline with much smaller encoding times and that it naturally scales with no performance degradation. We conduct ablation studies to show the effectiveness of various components of our algorithm in achieving high levels of reconstruction quality and understand the sources of improvements.

Our contributions are summarized below:

- We present NIRVANA, a patch-wise autoregressive video INR framework which exploits both spatial and temporal redundancies in videos to achieve high levels of encoding speedups (12×) at similar reconstruction quality and compression rates.
- We achieve a 6× speedup in decoding times and scale well with increasing number of GPUs, making it practical in various deployment scenarios.
- Our framework adapts to varying video resolutions and lengths without performance degradations. Different from prior works, it achieves adaptive bitrate compression based on the amount of inter-frame motion for each video.

## 2.2 Related Work

**Implicit Neural Representations (INR)** are a novel family of methods designed to map a set of coordinates to a specific signal - such as a single image or video - using a neural network as a function for such mappings. SIREN [13] demonstrates that by

utilizing periodic activation functions in MLPs, such a function can be fit and used to map a wide array of signals, including images, 3D shapes and videos. As an alternative, [19] shows than an INR network with standard activations can be trained by utilizing random Fourier features. [20] and [21] propose adaptive block-based approaches whose complexity mirrors the underlying signals. Frequency-based approaches are proposed in [22, 23, 24] that enable multi-scale representations. The first image specific INR method is COIN [1], which is extended to encode multiple images through network modulations in COIN++ [2]. A method for learning local implicit functions is proposed in [25] that results in smoother super-resolution outputs. Several methods [26, 27, 28] have explored meta-learning approaches to reduce the long encoding times of image INRs; [29] further shows that directly initializing a meta-sparse network not only gives a good initialization but also helps with model compression.

**INR for videos.** Despite the significant advances in INR methods for image compression, videos present a more challenging task for INR methods. For example, if we naively add time as an extra dimension to image-based methods, such as in SIREN [13], the resulting outputs are grainy. In [30, 31], INR methods that utilize flow-based information to encode videos are introduced; however, they cannot scale beyond short low-resolution videos. NeRV [3] is the first method to scale video compression using INRs. They modify the implicit mapping function to learn a direct mapping from a video frame index to an entire frame. Further extensions of NeRV, such as [32, 33], provide minor improvements over the original architecture. [34] propose learning a single shared NeRV for a diverse set of videos but are closer to explicit methods. Despite good reconstruction, the problems of long encoding times, the lack of inter-frame encoding, and the inability to adapt to video content

act as major drawbacks for widespread adoptions. [35] use learnable features instead of coordinates as input but also do not scale to higher video resolution and duration.

**Model compression** is typically achieved by pruning or quantization of network weights. A plethora of works perform model pruning with minimal loss of performance [36, 37, 38, 39]. Pruned models contain a majority of zeros and can be stored in sparse matrix formats [40] for reduction in model size. Alternatively, quantization works [41, 42, 43, 44] to reduce the number of bits needed to store each model weight, resulting in reduced disk space. As implicit neural networks represent the data using their model weights, data compression translates to model compression. In this work, we adopt the works of [17, 45] for model compression by maintaining a set of quantized parameters during training which are then stored on disk. These recent methods have shown to achieve high levels of compression through entropy regularization without sacrificing downstream network performance. We perform quantization and model compression *during training*, unlike the post-hoc pruning and quantization in NeRV [3].

## 2.3 Approach

### 2.3.1 Background

Given a video $\boldsymbol{V} \in \mathbb{R}^{N \times H \times W \times 3}$ consisting of $N$ frames, each with spatial resolution $H \times W$, an INR defines a mapping from the spatio-temporal coordinate $X = (x, y, t); X \in \mathbb{R}^3$ to the pixel value $p \in \mathbb{R}^3$. Thus, it implicitly represents a function $h_\theta : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ parameterized by $\theta$. The function is typically trained by minimizing the MSE-loss $||h_\theta(X) - \boldsymbol{V}||_2$. While this is a straightforward extension of image-based INR

methods to the spatio-temporal domain, it fails to exploit the spatial and temporal consistency in videos. Pixel-wise prediction leads to redundant computation and long encoding times while also producing blurry outputs[13],[1]. To mitigate this issue, NeRV [3] directly encodes the frame index $t \in \mathbb{R}$ as a positional embedding input to a model which outputs the entire image frame $\mathbb{R}^{H \times W \times 3}$. NeRV consists of several MLP layers followed by convolution layers which upsamples the latent representation to the target frame's spatial resolution. While this formulation improves upon the naive pixel-based formulation, it does not adapt to arbitrary resolutions, and does not capture the temporal dependencies between frames as it effectively acts as only an image encoder for each frame.

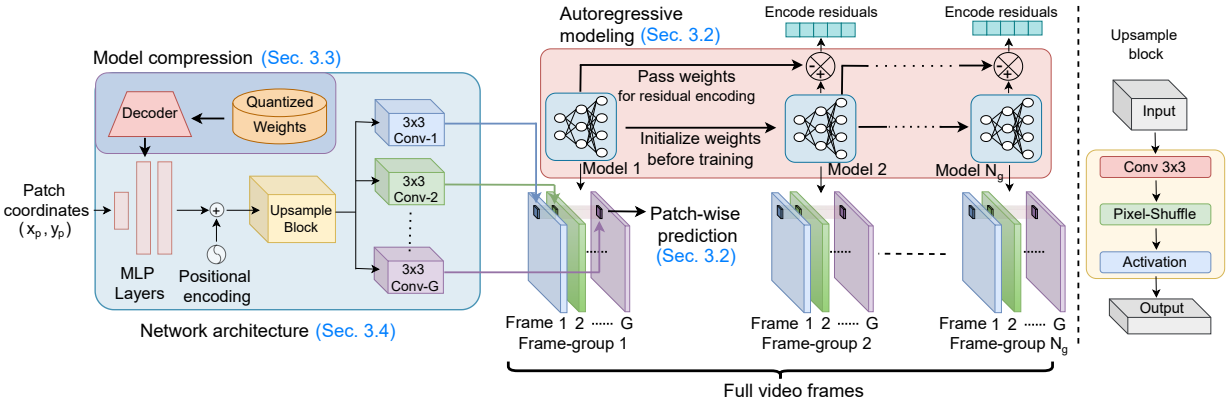



Figure 2.2: **Overview of NIRVANA:** (Left) We propose an autoregressive video INR framework which performs patch-wise prediction of groups of frames by fitting separate networks to each group. Each network is initialized with the previous group's network weights. We store quantized weights which are optimized during training. (Right) Our architecture consists of several SIREN MLP layers followed by convolutional blocks. It takes patch coordinates as inputs and outputs patches across a group of G frames.

### 2.3.2 Autoregressive Patch-wise Modeling

Patch prediction. The two dominant INR paradigms for video encodings, SIREN [13] and NeRV [3] represents two extreme ends of a spectrum: the former predicts every pixel in a video volume independently, while the latter predicts the pixels for a single frame

simultaneously. Through pixel-wise prediction, SIREN accommodates varying the output image's spatial resolution but does not exploit the spatial consistency of the image. In contrast, while NeRV exploits the spatial consistency of an image through convolutions, it cannot vary the image's spatial resolution. We adopt a middle ground between these two extremes by adopting a modeling approach that instead predicts *patches* of an image. This gives us the best of both approaches, since our model utilizes the spatial consistency of an image while still naturally scaling to varying image resolutions.

We push further in this direction by exploiting the temporal consistency in videos to predict a *volume of patches* across neighboring frames. We thus predict a patch group $\boldsymbol{P} \in \mathbb{R}^{G \times H_p \times W_p \times 3}$, where G is the number of frames in a group and $(H_p, W_p)$ is the patch size. This method enables us to reduce the amount of redundant computation in both the spatial and temporal dimensions, leading to significantly shorter encoding times compared to NeRV or SIREN.

Autoregressive networks. While it is straightforward to input a 3D patch coordinate $(x_p, y_p, g)$ (where $(x_p, y_p)$ are the patch centroids within a frame and $g$ is the frame index within each group) and output a patch volume using a single network, it still suffers from adaptability to varying video content, resolutions, or durations as mentioned in Section 2.3.1. To overcome these challenges, we propose to autoregressively fit separate networks to each frame group. Each network is fed with the centroids of patches $(x_p, y_p)$, and it outputs the corresponding 3D patch volume of the group. Every subsequent network is finetuned from the previous one, leading to shorter encoding times. As video content does not change much over a short time period of multiple frames, the difference in weights (or

weight residuals) after fine-tuning is small. Thus, encoding the weight residuals instead of the weights themselves leads to higher compression rates. The design of the algorithm 1 allows us to encode multiple chunks of a long video in a parallel manner, a key feature missing from existing methods. This means NIRVANA can scale linearly with the number of GPUs without any drop in performance (see Section 2.4.6).

### 2.3.3 Model Compression and Weight Storage

Since the network weights are the latent representations for the video, network size directly translates to bitrate of the video encoding. To reduce network size, we adapt existing works which perform model weights/latent representation compression [17, 45]. For a weight parameter $\boldsymbol{W} \in \theta$, where $\theta$ is the set of model parameters, we maintain a corresponding quantized latent weight $\widetilde{\boldsymbol{W}}$. The continuous weight parameter $\boldsymbol{W}$ is then obtained as $\boldsymbol{W} = f_\phi(\widetilde{\boldsymbol{W}})$ where $f_\phi$ is a learned linear transform. The entire setup is trained end to end, without any post-hoc quantization and fine-tuning. As previously explained, we encode the quantized latent residuals instead of the latents themselves to achieve higher levels of compression. We encode these residuals using arithmetic coding [46], a lossless entropy-based compression algorithm. In order to encourage the latents to have lower entropy, we add an entropy regularization term to our loss function. This term encourages the network to have a lower entropy and hence a lower bitrate.

When decoding, each weight is obtained sequentially by cumulatively adding residuals. This approach helps in making the bitrate of NIRVANA adaptive to the video content: for frame-groups that have little motion, they are already closer to convergence and thus

have small differences in their network weights, leading to sparser residuals and subsequently, lower bitrates. This feature is missing in other methods as the models are fixed for a given video.

### 2.3.4 Network Architecture

In this section, we describe the network architecture which is used for each frame group, as illustrated in Figure 2.2. For a group of $G$ frames, we segment patch volumes of shape $(H_p, W_p, G)$. The input to the network is the 2D patch coordinate $(x_p, y_p) \in \mathbb{R}^2$ and the output is the corresponding RGB patch volume $\boldsymbol{P} \in \mathbb{R}^{G \times H_p \times W_p \times 3}$. We stack multiple MLP layers with SIREN activations to obtain an output feature representation vector $s_p \in \mathbb{R}^d$ of dimension $d$. We replicate $s_p$ by $G$ times, and add positional encoding vectors based on the position of the frame within the group, using the following embedding function:

$$\tau(t, 2i) = \sin\left(\frac{t}{f^{2i}}\right) \tau(t, 2i+1) = \cos\left(\frac{t}{f^{2i}}\right), i \in [0, d) \tag{2.1}$$

where $t \in [0, G)$ represents the position of the frame within the group of $G$ frames. We then add a decoder block as in [3] followed by a $3 \times 3$ convolutional layer to output $G$ patches. For a video with $N$ frames, we segment it into $N_g$ frame-groups with each group consisting of $G$ frames ($N = N_g \times G$). For the $g^{\text{th}}$ frame-group ($g \in [0, N_g)$), the corresponding network is represented as $h_{\theta_g}$ consisting of parameters $\theta_g$. The overall loss objective for

**Algorithm 1:** Sequential Video INR

**Init:** Randomly initialize network $h_{\theta_0}$ with initial weights $\theta_0$ and training iterations $T$. The video contains $N$ frames segmented into $N_g$ frame groups of size G each.;
**for** $g$ *in* $0, 1, 2, \cdots, N_g - 1$ **do**
  **if** $g = 0$ **then**
    Train $h_{\theta_0}$ for $T$ iterations for all patches;
    Store weights $h_{\theta_0}$;
  **else**
    Initialize weights $h_{\theta_k} \leftarrow h_{\theta_{k-1}}$;
    Finetune $h_{\theta_k}$ for $T_r$ iterations for all patches;
    Store quantized latent of residuals $h_{\theta_k} - h_{\theta_{k-1}}$;
  **end**
**end**

training the network for the $g^{\text{th}}$ frame-group is therefore

$$\mathcal{L}_g = \mathcal{L}_{\text{mse}}(h_{\theta_g}(p), v_g) + \lambda_I \mathcal{L}_{\text{ent}}(\theta_g) \tag{2.2}$$

where $p$ represents patch grid coordinates and $v_g$ means the corresponding ground-truth frame-group pixel values. $\mathcal{L}_{\text{ent}}(\theta_g)$ represents the entropy regularization loss on the model parameters $\theta_g$. It is weighed by the coefficient $\lambda_I$ controlling the rate-distortion trade-off for reconstructing the frame groups.

## 2.4 Experiments

### 2.4.1 Datasets and Implementation Details

The standard benchmark UVG dataset [18] is used to compare our approach NIRVANA with prior video INR works. Following similar setups [3], our approach is evaluated on 7 videos from the dataset at 1080p resolution (UVG-HD) and 120 fps with 6 videos con-

| Dataset | Method | Encoding Time (Hours) ↓ | Decoding Speed (FPS) ↑ | PSNR ↑ | BPP ↓ |
|---|---|---|---|---|---|
| UVG-HD | SIREN | ∼30 | 15.62 | 27.20 | **0.28** |
| | **NIRVANA (Ours)** | **5.44** | **87.91** | **34.71** | 0.32 |
| | NeRV | ∼80 | 11.01 | 37.36 | 0.92 |
| | **NIRVANA (Ours)** | **6.71** | **65.42** | **37.70** | **0.86** |
| UVG-4K | NeRV | ∼134 | 8.27 | **35.24** | 0.28 |
| | **NIRVANA (Ours)** | **20.89** | **50.83** | 35.18 | **0.27** |

Table 2.1: **Comparison with video INR approaches on UVG benchmarks.** We vary patch size of NIRVANA on UVG-HD to match the BPP of SIREN and NeRV respectively. NIRVANA achieved much faster encoding and decoding speed, while maintaining better or on-par quality at comparable BPP.

sisting of 600 frames and 1 with 300 frames. To show our scalability for higher resolution videos, we show results for the 7 videos at 4K resolution (UVG-4K) as well. We additionally include a video from the Youtube-8M (see supplement) [47] dataset at 1080p resolution and 60 fps with 3 separate versions segmented at 2000/3000/4000 frames to demonstrate our model's capability for long videos. We use the standard PSNR (in dB) to measure the reconstruction quality and bits-per-pixel (BPP) to measure the compression rate. We also include encoding times as well as their decoding speed in fps.

The MLP network consists of 5 SIREN layers with a layer size of 512 and sine activation. The network predicts $32 \times 32$ patches for 3 frames ($G = 3$) in all our experiments unless mentioned otherwise. The number of iterations is set to 16000 for the first group in order to obtain a good initialization and 2000 iterations for subsequent groups. We set the learning rate to be $5 \times 10^{-4}$ and optimize the network with the MSE and entropy regularization loss. The entropy loss weight coefficient $\lambda_I$ as defined in Equation 2.2 is set to $1 \times 10^{-4}$. In practice, the coefficient can be varied to control the trade-off between PSNR and BPP. We use the torchac library to perform arithmetic encoding of the quantized weight residuals. Since the convolutional layers typically contain only a fraction of

the total network parameters (∼10%), we do not quantize their weights and use the LZMA compression method for storing their residuals.

We use pixel-wise method SIREN [13] and frame-wise method NeRV [3] as our baselines. For SIREN, we use a 5-layer MLP with hidden dimension of 2048. For NeRV, we use the NeRV-L configuration as specified in the paper. Encoding times reported are equivalent to when fully run on a single NVIDIA RTX 2080 GPU. For NeRV, we fit separate models to each video and remove 40% of the parameters during the pruning stage with the remaining weights quantized to lowest possible bit-width without significant performance drop. Further architectural details can be found in the supplementary material.

### 2.4.2 UVG-HD

Comparisons on the UVG-HD dataset are summarized in Table 2.1. By varying the patch size, we let NIRVANA achieve similar BPP to SIREN and NeRV respectively. NIRVANA outperforms SIREN by a significant margin in terms of PSNR with 6× shorter encoding times. Similarly, our approach obtains speedups of ∼12× compared to NeRV while still achieving marginally higher PSNR (+0.34dB) and lower BPP. Additionally, we obtain a decoding speed of ∼65 FPS which is nearly 65× and 6× speedup in inference time/decoding compared to SIREN and NeRV respectively. This shows the efficacy of our framework to reduce redundant computation in both the spatial and temporal domains. We show our qualitative results in Fig. 2.6.

| Num Frames | Method | Encoding Time (Hours) ↓ | PSNR ↑ | BPP ↓ |
|---|---|---|---|---|
| 2000 | NeRV | 84.44 | 33.38 | 0.22 |
| | NIRVANA (Ours) | 20.85 | 35.43 | 0.62 |
| 3000 | NeRV | 134.58 | 31.6 | 0.16 |
| | NIRVANA (Ours) | 31.37 | 35.21 | 0.64 |
| 4000 | NeRV | 190.30 | 30.53 | 0.12 |
| | NIRVANA (Ours) | 41.84 | 35.15 | 0.65 |

Table 2.2: **Video duration adaptability:** For longer videos, we maintain similar reconstruction quality (∼35 PSNR) and compression rate (∼0.62 BPP). We retain a significantly faster encoding speed than NeRV which suffers from significant degradation with increased number of frames.

### 2.4.3 UVG-4K

To analyze the spatial adaptability of our approach, we test our method on the UVG-4K dataset, with results shown in Table 2.1. Compared to NeRV, we achieve a ∼6× speedup in both encoding and decoding times, while maintaining similar PSNR (35.24 vs 35.18) and slightly better BPP (0.28 vs 0.27). Furthermore, to adapt to such higher resolution data, our method does not require any architectural modifications as opposed to NeRV which requires addition of 2X upsampling blocks to increase predicted resolution. Note that a higher PSNR can be achieved with longer training schedules as we show in Section 2.5.4, but we limit to 2000 iterations to maintain consistency across datasets.

### 2.4.4 Long Videos

We now analyze the effect of increasing video duration for our approach. We utilize a video from the Youtube-8M dataset and evaluate on 3 separate segments consisting of the first 2000/3000/4000 frames. Results are summarized in Table 2.2. Our approach maintains a similar PSNR ($< 0.3$ drop) with increased number of frames while still encoding

at a similar bitrate ($< 0.04$ increase). In contrast, even with higher encoding times ($4\times$ slower), NeRV suffers from significant degradation on longer videos with PSNR dropping from $33.38 \rightarrow 31.6 \rightarrow 30.53$. Since NeRV's model size remains constant, its BPP reduces with increased number of frames. However, the fixed number of network parameters limits its ability to fit to a larger set of frames, leading to performance drops.

Additionally, since our approach is autoregressive, it needs to be trained only once even with increasing number of frames. Networks for future frames are simply initialized with the weights of the previous networks and trained before encoding the weight residuals. Such a modeling makes it suitable for online scenarios as well with a constant stream of frames. In contrast, NeRV requires separate models to be trained for different video durations as each training epoch consists of training on all frames. Note that both approaches scale linearly with increased video duration, but NeRV fits the same model to larger video signals, leading to performance drops. Specific architectural modifications are necessary to improve the PSNR for longer videos which comes at the cost of even higher encoding times.

### 2.4.5 Adaptive Compression

Videos can consist of different levels of inter-frame motion with more static videos containing higher levels of temporal redundancy in comparison to dynamic ones. To illustrate the capability of our approach to exploit such redundancies, we evaluate the compression rate on 6 videos in the UVG-HD dataset which consist of 600 frames. We sort each video according to their average MSE between subsequent frames, which serves as a proxy to the amount of temporal redundancy in the video. More static scenes like Honeybee have a

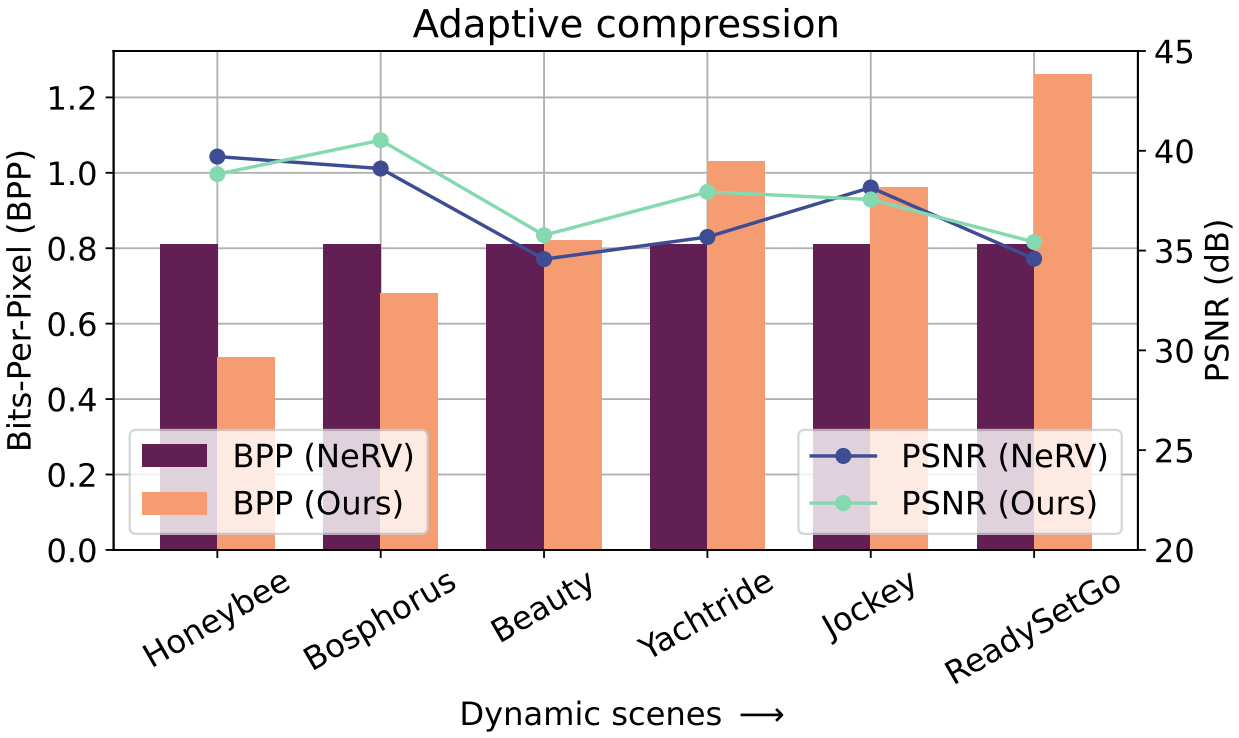



Figure 2.3: **Video content adaptability:** 6 videos are sorted in increasing order of variation between subsequent frames. Our approach shows adaptive bitrate compression, with more static scenes exhibiting lower BPP, while highly dynamic ones being allocated more bits while maintaining a similar PSNR as NeRV (and 12× encoding speed).

lower MSE compared to highly dynamic scenes like Jockey. We plot the PSNR and BPP of NIRVANA and NeRV with increasingly dynamic video content and show the results in Figure 2.3. Note that the average PSNR/BPP over the 6 videos can be increased or decreased by varying other hyperparameters such as patch size, number of groups, entropy loss coefficient etc. (as shown in Section 2.5), but we focus on adaptability to videos for a given hyperparameter configuration. We see that our approach has an adaptive bitrate compression with more static scenes like Honeybee (MSE $2.2 \times 10^{-4}$) that has a lower bitrate (0.51), compared to dynamic scenes such as Jockey (MSE $0.9 \times 10^{-3}$) which are allocated more bits (0.96). We maintain similar PSNR as NeRV which has a constant BPP due to the same model applied to all videos. While NeRV's quantization bit width can be reduced further for lower BPP, it is a post-hoc approach which comes at the cost of PSNR and requires tuning for each video. In contrast, our approach adaptively varies the BPP during training with no change in hyperparameters.

### 2.4.6 GPU Parallelization

We now analyze the scalability of our approach with a larger number of GPUs. In Figure 2.4 we plot the encoding times for "Jockey" (both 1080p and 4K versions) from UVG dataset for NeRV (using distributed training) and our methods. The design in Algorithm 1 allows different *chunks* of the source video to be processed autoregressively on separate GPUs. As the number of GPUs are scaled with a factor of 2×, our approach achieves close to linear scaling with very little overhead for the case of UVG-4K ($1.0\times \rightarrow 2.0\times \rightarrow 3.8\times \rightarrow 7.3\times$) compared to a weaker scaling of NeRV ($1.0\times \rightarrow 1.7\times \rightarrow 2.7\times \rightarrow 4.3\times$).

UVG-HD shows a higher amount of overhead but still scales fairly well with increased GPUs compared to NeRV. Thus, we see the capability of parallelization of our approach with higher number of GPUs. Also note that the time taken by NeRV for HD videos on 8 GPUs is still higher than the time taken by NIRVANA on a single GPU.

## 2.5 Ablation Studies

In this section, we study the impact of various parameters of our approach on the PSNR-BPP tradeoff curves as seen in Figure 2.5. By varying the entropy loss coefficient, we obtain different points on the tradeoff curve with a higher coefficient leading to lower BPP (low rate) but also lower PSNR (high distortion). We additionally show the convergence effects of longer training for each group with increased number of iterations. Results are evaluated on the Jockey video of the UVG-HD dataset. While varying each parameter along with the entropy coefficient, we fix other parameters to their default values of patch size at $32 \times 32$, group size at 3, and number of iterations at 2000. We sample values of the entropy coefficient $\lambda_I$ between $1 \times 10^{-5}$ and $5 \times 10^{-4}$ to obtain various points on the tradeoff curve.

### 2.5.1 Effect of Entropy Regularization

We analyze the effect of varying the entropy coefficient $\lambda_I$ and obtain a PSNR-BPP curve visualized in Figure 2.5(a). In general, we see that increasing $\lambda_I$ decreases the BPP at the cost of PSNR. This is to be expected as increasing the entropy regularization forces the quantized weights of each frame group's networks to lie in fewer quantization bins.

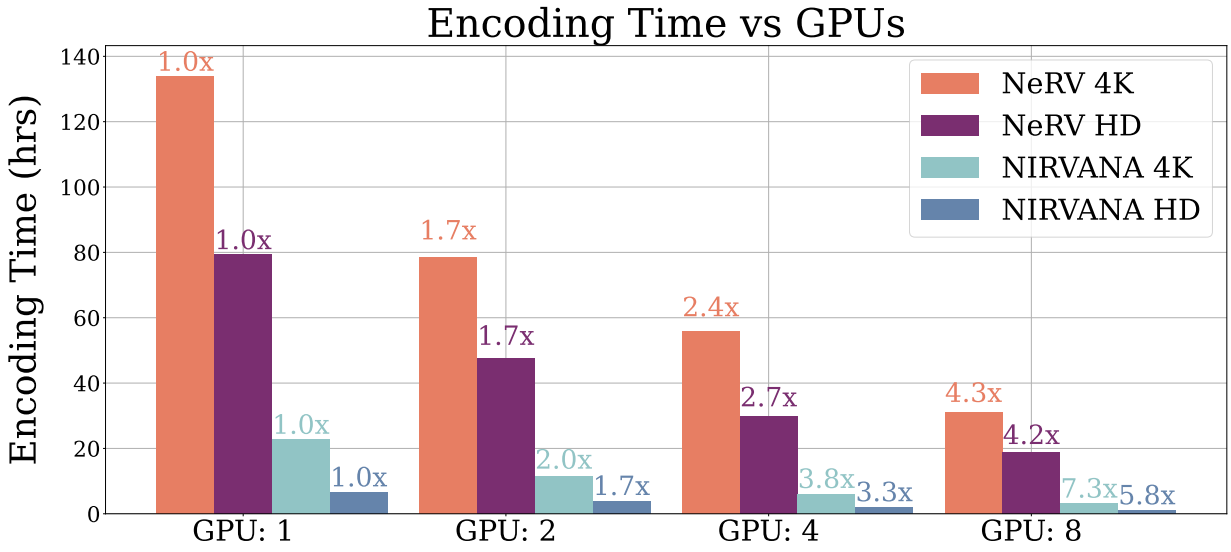
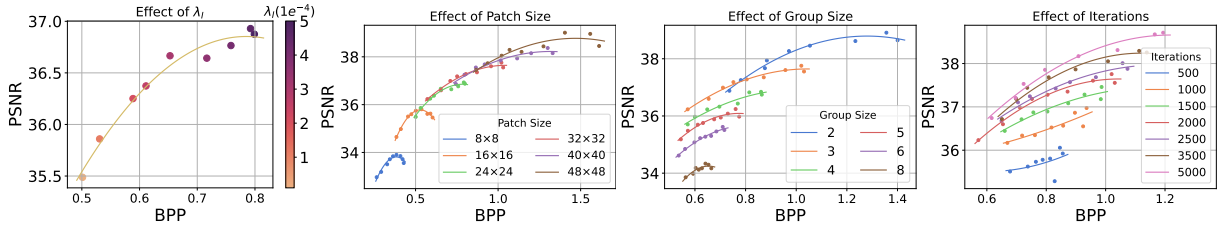



Figure 2.4: **GPU scalability of NIRVANA:** We compare scalability of our approach with NeRV in terms of encoding time with increasing number of GPUs at two video resolutions: 1080p and 4K. We scale close to linear for 4K and have much lower overhead compared to NeRV for both resolutions.




(a) (b) (c) (d)

Figure 2.5: (a) Effect of entropy regularization: the larger $\lambda_I$ is, the lower the entropy and the BPP. (b, c) As the patch size or group size increases, PSNR increases at the cost of higher BPP. (d) The longer the training iteration (encoding time) is, the higher the PNSR gains.

Consequently, more weight residual (difference between quantized weights of subsequent frame groups' networks) values are 0 leading to lower entropy of the weights and subsequently lower BPP of the model. The entropy coefficient thus provides a natural way of controlling the PSNR-BPP tradeoff according to the required application.

### 2.5.2 Effect of Patch Size

We vary the patch prediction size of our network from $8 \times 8$ to $48 \times 48$ in steps of 8. We visualize the results in Figure 2.5(b). In general, increasing patch size shifts the

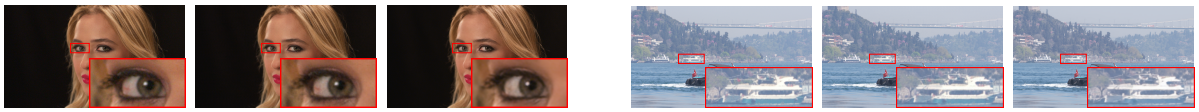


Figure 2.6: (Left) Ground truth video frames. (Center) Reconstruction from NIRVANA. (Right) Reconstruction from NeRV. We show that NIRVANA is able to preserve the image fidelity after reconstruction, capturing important details such as the veins in the eye of Beauty, and the color quality in the Bosphorus video.

curve upwards and to the right corresponding to higher PSNR but also high BPP. A higher patch size results in an increase in number of network parameters (both in convolutional and SIREN layers) and hence its expressivity, leading to higher PSNR. However, as patches are less localized, the outputs of networks between subsequent frame-groups vary more significantly with dynamic scenes (such as Jockey), leading to larger residuals. This increases the entropy of the residuals and as a result, the BPP.

### 2.5.3 Effect of Frame Group Size

We vary the frame group size from 2 to 8 in steps of 1, visualizing the results in Figure 2.5(c). Increasing the group size, in general, reduces BPP at the cost of PSNR. This is expected as a single model shares computation across a larger group of frames effectively leading to fewer parameters per frame and lower BPP. However, group size of 3 obtains the best tradeoff curve in the $< 0.8$ BPP regime with higher group size detrimental to the performance. This is likely because of the fixed MLP representation capability which learns a shared global representation for all frames within a group. For a dynamic scene such as Jockey with significant pixel shift between frames (Section 2.4.5), a larger MLP is necessary for capturing the variations within a group.

### 2.5.4 Effect of Number of Training Iterations

To analyze the effect of longer training schedules, we vary the number of training iterations for the network for each frame group from 500 to 5000. Results are shown in Figure 2.5(d). Increasing the number of iterations improves the PSNR-BPP tradeoff with the curve shifting upwards. This shows that our network can obtain higher quality reconstructions for longer training times at no cost to the compression rate. This can be made feasible with higher number of GPUs as shown in Section 2.4.6. However, increasing iterations provides diminishing gains as we observe the curves approaching closer to each other with higher iterations.

## 2.6 Discussion

In this work, we propose an autoregressive video INR framework, NIRVANA, which segments videos into groups of frames and fits separate neural networks to each group. Each network performs a patch-wise prediction across the group of frames thus exploiting both the spatial and temporal redundancy present in videos, improved from the previous works. Each network is initialized with the weights of the previous frame-group's trained network. We additionally quantize weights during training, requiring no post-hoc pruning or quantization and store weight residuals between subsequent frame group's networks to obtain high levels of compression. NIRVANA achieves 12× speedups on standard datasets compared to previous methods while maintaining similar levels of reconstruction quality and compression rate. NIRVANA adapts to varying video resolution and duration without large performance degradation and no architectural modifications. Additionally, our framework

adaptively compresses videos based on their inter-frame variation and also achieves high levels of decoding speed compared to prior works.

## 2.7 Appendix

### 2.7.1 Additional Implementation Details

#### 2.7.1.1 NeRV

We use the NeRV-L config from the original paper. The model takes in positional embedding of time coordinate as input. We set the number of sine levels to 80 and base value to 1.25. The hidden dimension of the 2-layer MLP in the beginning is set to 1024, with 128 output channels and an 8× channel expansion for the first NeRV Block. We stack 5-NeRV blocks with upscale factors of $\{5, 3, 2, 2, 2\}$ for HD version and an additional block with 2× upsampling for the 4K version. Following the standard training schedule, we use cosine learning rate schedule starting with $5e^{-4}$, with a warmup of 0.2. We use the combination of L1 and SSIM loss and train the model with a batch size of 1, as mentioned in the paper.

### 2.7.2 Entropy Loss and Weight Quantization

We follow the works of [17, 45] for performing model compression with small variations. We represent our MLP layer weights as $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_L$ for L layers with the $l^{th}$ layer weight matrix $\mathbf{w}_l \in \mathbb{R}^{O_l \times I_l}$ having a shape $O_l \times I_l$ consisting of continuous values. For each weight matrix, we maintain a corresponding flattened latent quantized representation

vector $\mathbf{w}_l \in \mathbb{Z}^{O_l I_l}$. $\mathbf{w}_l$ consists of integer values for each corresponding element in $\mathbf{w}_l$. For ease of notation, we drop the subscript $l$.

We then maintain decoders $f_\phi(.)$ parameterized by parameters $\phi$. The weight matrix $\mathbf{w}$ is then obtained from the quantized weights $\mathbf{w}$ as $\mathbf{w} = f_\phi(\mathbf{w})$. Prior works use matrices or vectors to represent the weight decoders while we use a single scalar $\phi$ as a parameter of the decoder. The weight matrix is thus simply, $\mathbf{w} = reshape(\phi\mathbf{w})$, where the scale parameter $\phi$ is multiplied with individual values of $\mathbf{w}$. We make the scale parameter learnable by passing gradients. Thus, it effectively controls the bit width of the weight matrix. We maintain separate decoders for each layer of the MLP.

To make the network fully differentiable, we maintain continuous surrogates $\widehat{\mathbf{w}}$ for the discrete latents $\mathbf{w}$. During training, we simply round the surrogates to their nearest integer to obtain the discrete latents which are then passed to the decoder. We make the rounding operation differentiable using the straight-through estimator [48] to pass the gradients from $\widehat{\mathbf{w}}$ to $\mathbf{w}$.

To reduce the entropy of the quantized latents, we use probability models from [8]. For each continuous surrogate $\widehat{\mathbf{w}}$, we maintain probability models $c_\theta(.)$ parameterized by $\theta$ which output the CDF of the latent distributions. Similar to prior works, we use uniform noise $n \sim \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)$ as a substitute for quantization. The entropy of the model can now be minimized by minimizing the self-information $\boldsymbol{I}$ as follows:

$$I(\mathbf{w}) = -\log_2(c_\theta(\widehat{\mathbf{w}} + n)). \tag{2.3}$$

This serves as the entropy regularization loss which controls the rate-distortion tradeoff. A higher entropy coefficient $\lambda_I$ leads to more compressed latents (lower rate), but usually at the cost of PSNR (higher distortion). The network latents, decoder parameter, and probability model parameters are learnable and jointly optimized. Following prior works, we use a learning rate of $1e{-}4$ for the probability model weights and the same learning rate for the decoder weights. We set the learning rate of the latents to $5e{-}4$. All the parameters are optimized in an end-to-end manner with an Adam optimizer during training, thus requiring no post-hoc approaches. After training, we discard the probability models and use the frequency of each quantized value in the latent vector to obtain the probability tables required for arithmetic entropy coding. Note that the continuous surrogates are discarded and only their rounded discrete latents are stored using entropy coding. These latents can then be decoded using the probability tables. The decoder parameters and probability tables have almost no overhead compared to the overall model latents.

## 2.8 Additional Dataset Details

### 2.8.1 UVG-4K

In addition to the datasets shown in Section 4 of the main paper, we show quantitative and qualitative results on 7 more videos from the UVG dataset at the 4K resolution: Twilight, Sunbath, CityAlley, FlowerFocus, FlowerKids, RiverBank, and RaceNight. We dub this dataset UVG-4K (Set 2).

### 2.8.2 Youtube-8M

We select 5 more videos from the Youtube-8M dataset, with varying video content to further test the ability of our model to encode longer videos. This is an extension of the experiments from Section 4.4 which consists of a single video (Mario Kart). We present the details of each video used in Table 2.3:

| Video | Frames | Link |
|---|---|---|
| Mario Kart | 4000 | `http://bit.ly/3XjIvfR` |
| Dota | 4261 | `http://bit.ly/3Xf6Nru` |
| Ride | 4000 | `http://bit.ly/3TOEgWI` |
| Submarine | 3626 | `http://bit.ly/3EJKzGM` |
| Water Scooter | 4199 | `http://bit.ly/3OhF99h` |
| Mortal Kombat | 3239 | `http://bit.ly/3EdvNGU` |

Table 2.3: **Details of Youtube-8M dataset.**

## 2.9 Video-wise comparison

In addition to the datasets shown in Section 4 of the main paper, we show quantitative results on 7 more videos from the UVG dataset at the 4K resolution: Twilight, Sunbath, CityAlley, FlowerFocus, FlowerKids, RiverBank, and RaceNight, we dub this dataset UVG-4K (Set 2). For videos with longer duration, we pick 5 additional videos from the Youtube-8M dataset and encode approximately 4000 frames for each video and show the quantitative results averaged over the 5 videos.

We show additional quantitative results on UVG-4K (Set 2) and Youtube-8M men-

tioned above.

### 2.9.1 UVG-HD

We provide video-wise results of our approach along with that of NeRV [3]. We evaluate the video on the additional perceptual quality metrics of FLIP [49] and VMAF [50] as well, along with the standard PSNR. In addition to the image quality metrics, we also measure the tOF/tLP metrics proposed by [51] for measuring temporal consistency. tOF measures the L1-error between the optical flows from the predicted frames and the ground-truth frames while tLP uses the LPIPS metric instead. Results are summarized in Table 2.4. We see that we continue to obtain similar performance compared to NeRV in terms of these metrics while being $\sim$12$\times$ faster. Also note the adaptive BPP of our method, which is based on the amount of motion in each video. In contrast, NeRV maintains a fixed BPP due to fixed model size (ShakeNDry shows twice the BPP due to half the number of frames). We observe a small drop in VMAF (92.33 $\rightarrow$ 91.14) while maintaining similar value of FLIP ($\sim$ 0.0632) compared to NeRV. We marginally outperform NeRV based on the temporal metrics of tOF (0.3167 $\rightarrow$ 0.3077) and tLP (0.2125 $\rightarrow$ 0.2032).

### 2.9.2 UVG-4K

We provide video-wise results on the 2 sets of UVG at 4K resolution. For Set 1, we obtain comparable performance to NeRV while obtaining $\sim$6$\times$ faster encoding speed. Similar to UVG-HD, we continue to show the benefits of adaptive compression, with static videos such as Honeybee showing lower levels of BPP (0.14) compared to the most dynamic

| Video Name | NIRVANA (Ours) | | | | | | NeRV | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR ↑ | VMAF ↑ | FLIP ↓ | tOF ↓ | tLP ↓ | BPP ↓ | PSNR ↑ | VMAF ↑ | FLIP ↓ | tOF ↓ | tLP ↓ | BPP ↓ |
| ReadySteadyGo | **35.43** | **98.04** | **0.0862** | **0.3348** | 0.2231 | 1.26 | 34.59 | 96.95 | **0.0862** | 0.3604 | **0.2089** | **0.81** |
| Bosphorus | **40.53** | **90.97** | **0.0549** | **0.1900** | **0.1371** | **0.68** | 39.11 | 88.26 | 0.0621 | 0.1980 | 0.1654 | 0.81 |
| Beauty | **35.77** | 84.46 | **0.0524** | **0.2716** | **0.3123** | 0.96 | 34.57 | **86.35** | 0.0605 | 0.3352 | 0.3646 | **0.81** |
| Honeybee | 38.83 | 91.31 | 0.0505 | 0.0918 | 0.1511 | **0.51** | **39.71** | **95.08** | **0.0419** | **0.0824** | **0.1491** | 0.81 |
| Jockey | 37.56 | 93.07 | 0.0710 | 0.7303 | 0.2777 | 0.96 | **38.16** | **95.76** | **0.0653** | **0.7001** | **0.2684** | **0.81** |
| Yachtride | **37.94** | **91.31** | **0.0668** | **0.3480** | **0.1604** | 1.03 | 35.68 | 88.70 | 0.0786 | 0.4243 | 0.1881 | **0.81** |
| ShakeNDry | 37.82 | 88.81 | 0.0608 | 0.1875 | 0.1607 | **0.76** | **39.68** | **95.20** | **0.0468** | **0.1165** | **0.1429** | 1.61 |
| Average | **37.70** | 91.14 | 0.0632 | **0.3077** | **0.2032** | **0.86** | 37.35 | **92.33** | **0.0631** | 0.3167 | 0.2125 | 0.92 |

Table 2.4: **Video-wise performance on UVG-HD:** We show video-wise results of the 7 videos in UVG-HD and compare the reconstruction quality using the 3 image metrics: PSNR, VMAF, and FLIP, the 2 temporal metrics: tOF/tLP [51], along with compression rate measured by BPP. We see that we maintain similar performance as NeRV in all 5 metrics and BPP while having 12× faster encoding speed (as also shown in Table 1 of the main paper).

video, ReadySteadyGo (0.41 BPP). For Set 2, we outperform NeRV by 1.5 PSNR while still obtaining 25% lower BPP 0.28 → 0.21. The PSNR drop of NeRV on the Twilight video is largely due to quantization at the fixed bit width of 20. Hand-tuning is necessary in order to maintain higher PSNR but at the cost of BPP. In contrast, our approach maintains the reconstruction quality for a variety of videos and adaptively quantizes for each video.

### 2.9.3 Youtube-8M

We now provide video-wise results of 5 videos picked from the Youtube-8M dataset at 1080p resolution. Details of the videos are provided in Table 2.3. Results are summarized in Table 2.6. We see that our reconstruction quality does not degrade with longer videos. This behavior is different from NeRV, which obtains a significant drop in PSNR (about −5.2). This is in line with the observations in Section 4.4 of the main paper, where we see that increasing number of frames results in drop of NeRV's reconstruction quality while we maintain similar levels of performance.

| Video Name | NIRVANA (Ours) | | NeRV | |
|---|---|---|---|---|
| | PSNR | BPP | PSNR | BPP |
| ReadySteadyGo | **33.85** | 0.41 | 33.22 | **0.24** |
| Bosphorus | 38.71 | **0.21** | **39.0** | 0.24 |
| Beauty | **31.96** | 0.28 | 31.05 | **0.24** |
| Honeybee | 35.64 | **0.14** | **36.36** | 0.24 |
| Jockey | 35.05 | 0.30 | **35.9** | **0.24** |
| Yachtride | **36.33** | 0.33 | 35.05 | **0.24** |
| ShakeNDry | 34.78 | **0.24** | **36.09** | 0.49 |
| Average | 35.18 | **0.27** | **35.23** | 0.28 |

(a) UVG-4K (Set 1)

| Video Name | NIRVANA (Ours) | | NeRV | |
|---|---|---|---|---|
| | PSNR | BPP | PSNR | BPP |
| FlowerFocus | 36.50 | **0.12** | **37.08** | 0.24 |
| CityAlley | 37.43 | **0.17** | **38.39** | 0.24 |
| Twilight | **38.02** | **0.13** | 20.99 | 0.24 |
| FlowerKids | **34.62** | 0.26 | 33.77 | **0.24** |
| RiverBank | **33.83** | 0.26 | 32.35 | **0.24** |
| RaceNight | 32.72 | 0.27 | **32.92** | **0.24** |
| Sunbath | 37.75 | **0.24** | **44.17** | 0.49 |
| Average | **35.84** | **0.21** | 34.23 | 0.28 |

(b) UVG-4K (Set 2)

Table 2.5: **Video-wise comparison on different sets of UVG-4K:** We show video-wise results on 2 different sets of 7 UVG videos at 4K resolution. Set-1 consists of videos from UVG-HD at 4K resolution while Set 2 consists of additional 7 videos from the dataset. We maintain similar performance in terms of PSNR and BPP as NeRV while also being $\sim$6$\times$ faster for both sets and being 6$\times$ faster in terms of encoding time.

## 2.10 Additional Ablations

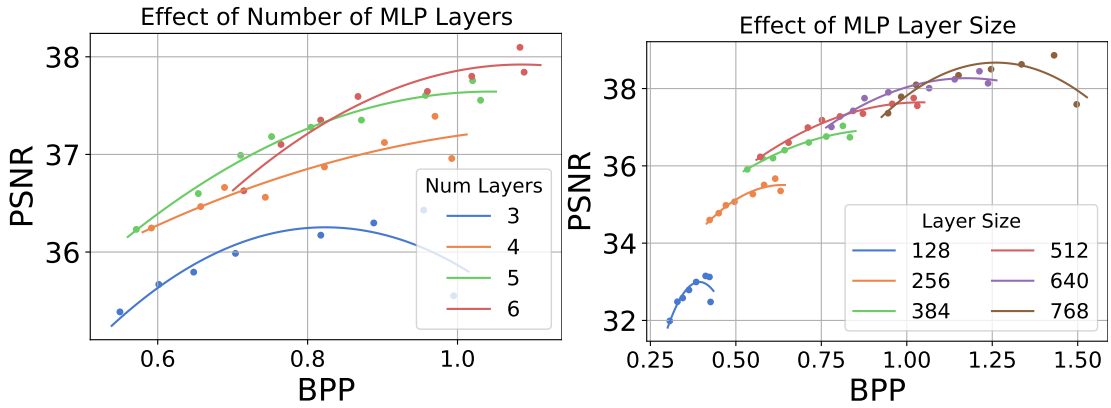
In addition to the ablations shown in Fig. 5, we analyze the effect of layer size and the number of layers of the MLP in our network when evaluating on the Jockey video of the UVG-HD dataset. Note that the default values of layer size is 512 and the number of layers is 5.

### 2.10.1 Effect of Layer Number

We increase the number of layers from 3 to 6, while keeping other parameters at their default values and varying the entropy coefficient $\lambda_I$ for each curve as in Section 5. Results are summarized in Figure 2.7(a). We see that increasing the number of layers from 3 to 5 improves the tradeoff curve (shifts upwards) in the low BPP regime (<0.8). However, increasing it further shifts the curve upwards and to the right. This might be because the

| Video Name | NIRVANA (Ours) | | NeRV | |
|---|---|---|---|---|
| | PSNR | BPP | PSNR | BPP |
| Dota | **38.03** | 0.62 | 35.53 | **0.34** |
| Ride | **36.65** | 1.09 | 29.74 | **0.36** |
| Submarine | **38.48** | 0.69 | 33.64 | **0.40** |
| Water Scooter | **37.79** | 0.84 | 30.46 | **0.34** |
| Mortal Kombat | **36.02** | 1.03 | 31.36 | **0.45** |
| Average | **37.39** | 0.85 | 32.14 | **0.38** |

Table 2.6: **Results on Youtube-8M videos with long duration.** We provide video-wise results on 5 videos picked from the Youtube-8M datasets with approximately 4000 frames compared to the typical 600 from UVG. Still we maintain PSNR/BPP with no change in hyperparameters, whereas NeRV shows large degradation in performance for the same network and similar encoding times.




(a) (b)

Figure 2.7: Increasing number of layers improves the PSNR/BPP curve upto 5 layers in the lower BPP regime (<0.8). Increasing layer size shifts the PSNR/BPP curve upwards and to the right as representation capacity increases along with more parameters.

MLP network requires higher levels of non-linearity to learn a global representation for a group of 3 frames which typically contain significant motion in the case of Jockey. However, for 6 layers the network shifts the curve upwards and to the right, and we no longer obtain increase in PSNR at no cost of BPP.

### 2.10.2 Effect of Layer Size

We vary the layer size from 128 to 768 progressively, in steps of 128 for each of the 5 layers. Results are summarized in Figure 2.7(b). We see that increasing the layer size simply shifts the curve upwards and to the right, which is expected as a higher number of parameters leads to more representation capability of the network at the cost of more parameters. While increasing the number of layers increases number of parameters as well, a similar tradeoff is not present in that case up to a certain level, suggesting that a minimum number of non-linearities/activation functions are important to achieve the optimal tradeoff.

### 2.10.3 Effect of video content

Section 4.5 in the paper shows the capability of our approach to adapt to video content based on varying stability. To further illustrate this, we visualize the correlation between BPP and L1-error (average L1 norm between pixel values of two frame groups) in Fig. 2.8. We see that for the Jockey video, lower L1-error between 2 subsequent frame groups shows direct correlation with the BPP required for storing that frame-group. This is to be expected as lower frame residuals reduces the entropy of the quantized residual weights as well and subsequently, lower BPP.

In addition to BPP, we also analyze the effect of convergence speed for various types of videos. We vary number of iterations for training networks for each frame group for the Honeybee and Jockey videos and visualize the results in Table 2.7. Stable videos such as Honeybee converge faster with only a 0.9 dB PSNR drop for 4× encoding speedup from

| Iterations | Honeybee | | Jockey | |
|---|---|---|---|---|
| | PSNR | BPP | PSNR | BPP |
| 500 | 37.93 | **0.35** | 35.97 | **0.85** |
| 1000 | 38.25 | 0.37 | 36.82 | 0.90 |
| 1500 | 38.72 | 0.50 | 37.25 | 0.93 |
| 2000 | **38.83** | 0.51 | **37.56** | 0.96 |

Table 2.7: **Convergence.** We vary number of training iterations for each frame group in the Honeybee (static) and Jockey (dynamic) videos from UVG-HD. Honeybee achieves faster convergence showing that stable videos can be encoded faster.

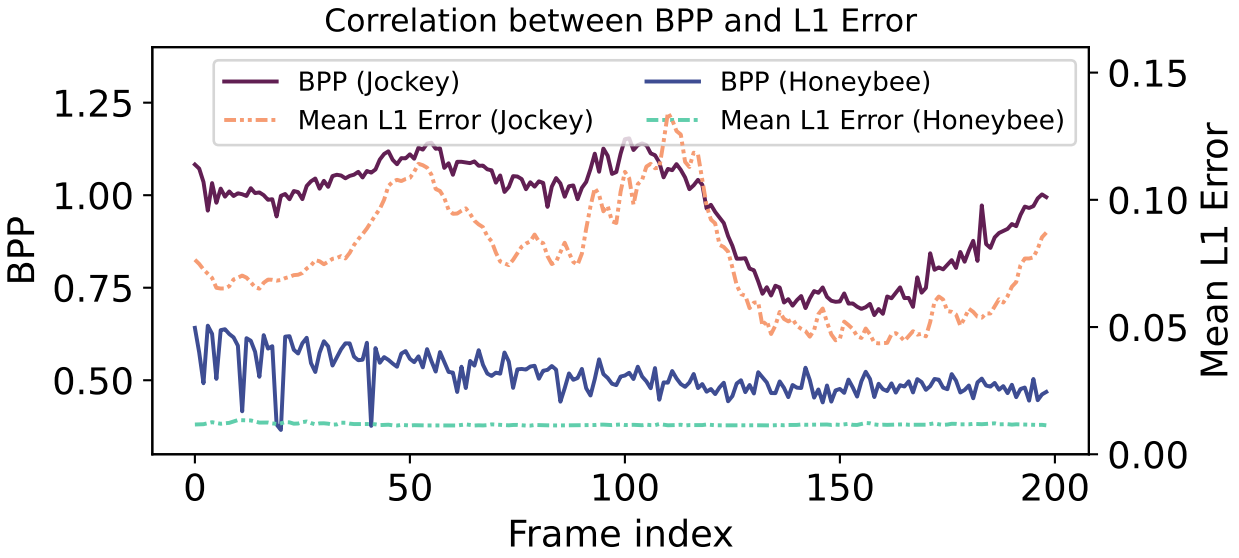



Figure 2.8: BPP correlates with L1 error: lower L1 error gives lower BPP.

2000 to 500 iterations while dynamic videos such as Jockey obtain a larger $1.5dB$ PSNR drop at similar encoding speedups. Due to our autoregressive modeling, the initialization of the network weights from the previous frame group provides a good solution for stable videos with little inter frame shift in comparison to dynamic ones.

| Denoising Method | Black | White | Salt& Pepper | Random | Average |
|---|---|---|---|---|---|
| Baseline | 27.5 | 28.29 | 27.95 | 30.95 | 28.74 |
| Mean Filter | 29.11 | 29.06 | 29.10 | 29.63 | 29.22 |
| Median Filter | 33.89 | 33.84 | 33.87 | 33.89 | 33.87 |
| Gaussian Filter | 30.27 | 30.14 | 30.23 | 30.99 | 30.41 |
| NIRVANA | **37.18** | **37.19** | **37.21** | **37.22** | **37.20** |

Table 2.8: **Results for video denoising.** We outperform classical denoising filters by a large margin for different types of noises.

## 2.11 Denoising

To test our method on downstream applications, we choose the task of denoising. Given a noisy video, our method is capable of removing the noisy patterns without any explicit supervision. We train and test our method on videos with various noise patterns and observe that out method outperforms all classical filter baselines. Results are shown in table 2.8. We outperform classically filters such as Mean/Median/Gaussian filter for a variety of noises such as Black/White/Salt and Pepper showing the efficacy of our representations to be applied to other tasks as well.

## 2.12 Qualitative Results

In Figure 2.10 we qualitatively visualize the reconstruction results for 3 videos from Set 2 of UVG-4K and 2 videos from Set 1. We obtain higher-quality and more faithful reconstructions while preserving more details at similar or even lower BPP compared to NeRV; *e.g.*, Twilight (0.24 $\rightarrow$ 0.13), RiverBank (0.24 $\rightarrow$ 0.26), CityAlley (0.24 $\rightarrow$ 0.17). Notice the bird which is reconstructed by our approach in Twilight (top), or finer details

Figure 2.9: **Qualitative results from UVG-4K Set-2:** (Left) Ground truth video frames. (Center) Reconstruction from NIRVANA. (Right) Reconstruction from NeRV. Top to bottom: We show further examples where NIRVANA is able to preserve the image fidelity after reconstruction, such as the bird in Twilight, the tree in RiverBank, and the human faces in CityAlley.

of the branches in RiverBank (second), or maintaining the right color information of the door and the people's shirts in CityAlley (third) or the red dot in Yachtride (fourth). We continue to maintain important information in the images such as the number on the signboard of ReadySetGo (bottom) while NeRV fails to capture these fine details.

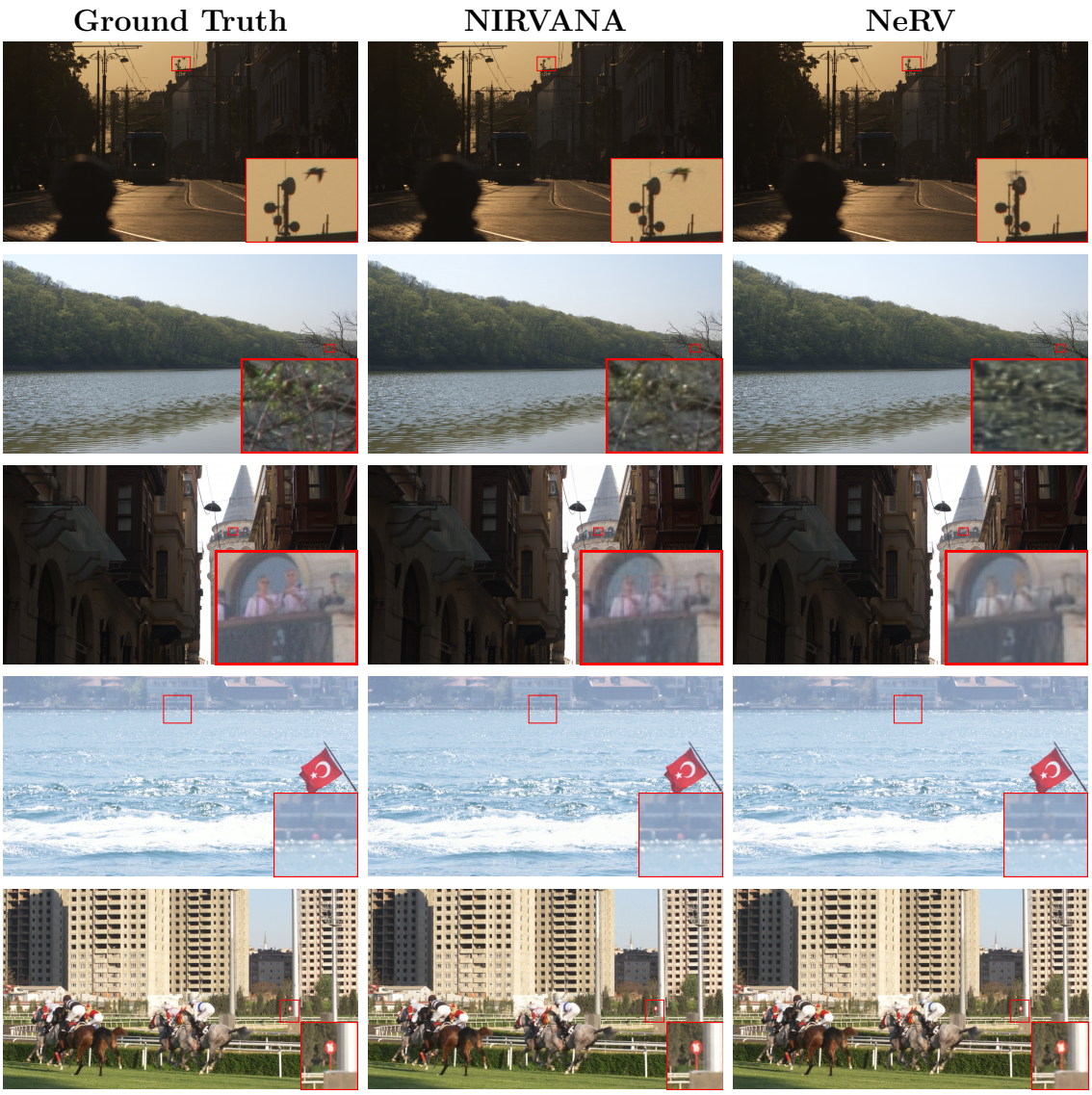



Figure 2.10: **Qualitative results from UVG-4K:** (Left) Ground truth video frames. (Center) Reconstruction from NIRVANA. (Right) Reconstruction from NeRV. Top to bottom: We show additional examples where NIRVANA is able to preserve the image fidelity after reconstruction, such as the bird in Twilight (top), the tree in RiverBank (second), humans in CityAlley (third) and signboards in ReadySetGo (bottom).

# Chapter 3: SIEDD: Fast Implicit Neural Video Coding

In this chapter we dive deep into the most obvious drawback of video INR systems - encoding speed. In real world settings, long encoding times translate to higher costs, which negates any potential benefits from the compression. The science might "just work", but the economics don't. Hence, it is crucial to develop INR-based video codecs that have fast encoding speeds along with good compression. Since the problem is overfitting, we need to find a way where we can have a good enough prior and at the same time ensure that the optimization process is fast.

## 3.1 Introduction

Video data forms the majority of internet traffic and it is projected to grow exponentially over the next decade. Traditional video codecs [52, 53, 54] have hit a wall and the field is increasingly looking towards neural-based methods [cite] to deliver efficient rate-distortion trade-offs. Implicit Neural Representations (INRs) for videos offer an alternative functional representation of videos. Video-INRs have good compression and great decoding speeds, but suffer from slow encoding speeds, which makes them impractical.

Unlike autoencoder-based video coding methods [55, 56, 57], Video-INRs are optimized per video, making them more truthful to the source, without any hallucinations [11]

a necessary property. But this presents a huge problem - encoding a video is no longer a simple forward pass, but an extensive gradient optimization process which can take hours to encode a single clip. We stress on the fact that encoding time is crucial for widespread adoption of INR-based video codecs. For example, the price of encoding a single minute of 1080p video at 30fps costs around $0.04 on AWS Mediaconvert, while training a Video-INR like [58] on the same clip with an RTXA5000 would cost upwards of $3, a whopping 75x increase.

Existing works [29, 59, 60] in the field point towards having a good prior/initialization to be the key factor in imporving optimization times. However, these methods require huge memory [29] or do not scale beyond small video resolutions [60], limiting their impact. To overcome these limitations, we introduce SIEDD, a shared-encoder architecture designed for scalable and efficient encoding. We first rapidly train a shared encoder on a small set of keyframes—without requiring full convergence—to capture low-frequency, video-specific features. Inspired by findings in [61, 62], we leverage the insight that early INR layers encode generalizable representations that converge quickly and transfer well across frames. In contrast to frame-wise video INRs, which require per-frame encoding and lack spatial flexibility, our method takes normalized 2D coordinates as input. This enables continuous-resolution decoding from a single encoding pass—eliminating the need for resolution-specific transcoding and significantly reducing overhead. Once the encoder is trained, we freeze it and train lightweight, frame-group-specific decoders independently. This design enables scaling to long videos and allows parallelized decoder training, as demonstrated in [63]. Additionally, by exploiting spatial sparsity, we subsample the coordinate space during training—yielding large gains in encoding speed without compromising

reconstruction fidelity. Finally, by using simple MLP layers throughout, our architecture remains compatible with recent advances in LLM quantization [64, 65], enabling further compression without any architectural changes. SIEDD achieves an impressive 20× encoding speed-up on UVG-HD [18] and over 30× on UVG-4K [18], while preserving high reconstruction quality. This speedup is a step towards making INR based video codecs more practical for deployment.

To summarize, our contributions are as follows:

- A novel architecture with shared encoder and discrete decoders that greatly speeds improves video encoding times of Video-INRs. Our model can scale both spatially (to 4K) and temporally (for longer videos) without any modifications.
- A two-stage training process that uses the fact that early INR layers require fewer iterations to converge, combined with sparse sampling.
- Extensive experiments on UVG [18], UVG-4K and DAVIS datasets along with architectural ablations.

### 3.1.1 Video Compression

Legacy video codecs like H264 [52], HEVC [53] and the more recent VVC [54] operate on similar first principles. They compress videos by exploiting redundancy and motion between frames. However, such hand-engineered techniques and heuristics have hit a limit in terms of performance gains [66]. Neural video codecs [55, 56, 57] build on existing autoencoder based hyper-prior architectures [8] to improve compression.

### 3.1.2 Implicit Neural Representations

Implicit Neural Representations (INRs). have emerged as a compact and differentiable paradigm for modeling continuous signals such as images[13], videos [3], audio [67] and 3D scenes [14]. Rather than storing discrete data, INRs encode a signal as the weights of a neural network that maps input coordinates to output values, offering high fidelity and resolution-agnostic reconstructions. Early works like SIREN [13] demonstrated the expressivity of periodic activation functions in fitting detailed signals from scratch. Extending to video, NeRV [3] introduced a frame-wise INR architecture mapping timestamps to RGB frames, enabling fast inference but at the cost of limited spatial control. Subsequent efforts[58, 63, 68, 69] addressed these limitations: HNeRV[58] introduced content-adaptive embeddings to improve convergence and generalization, while NIRVANA [63] adopted an autoregressive, patch-wise approach to exploit spatio-temporal redundancy and support scalable encoding of high-resolution, long-duration videos. Works like Tree-Nerv [70], DS-Nerv [71]incorporated ideas of efficient sampling and dynamic codes to further improve these systems.

## 3.2 Model Compression

With Video-INRs, the task of Video compression is essentially transformed into a model compression problem. In this functional paradigm, the challenge then becomes to effectively quantize [72, 73, 74] and store the weights of the resulting neural network with minimal loss. We employ HQQ Quantization [64] - a post training quantization technique combined with lossless entropy coding [75] to provide efficient bitstream.

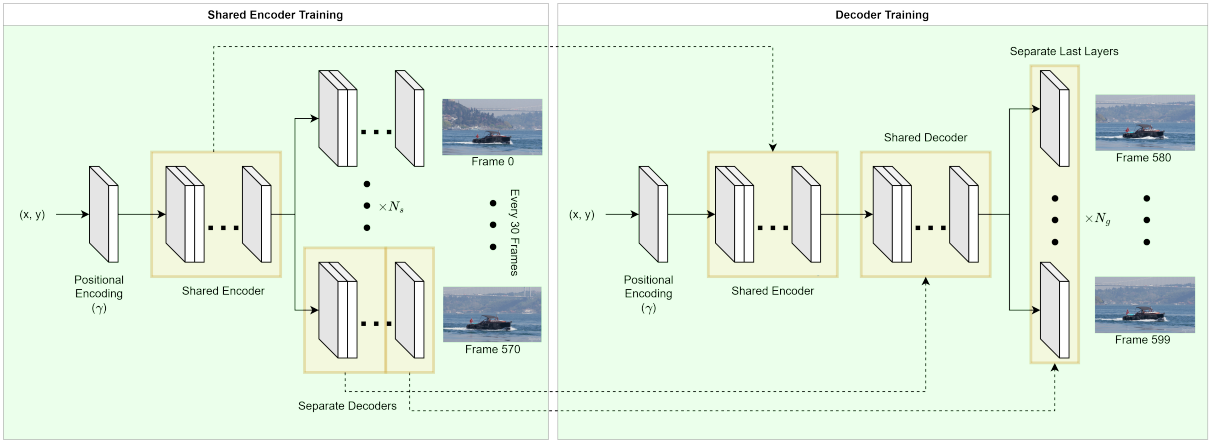



Figure 3.1: Overview of the SIEDD architecture. During the shared encoder training phase (left), a positional encoding of 2D coordinates is passed through a shared encoder and used to train a small number of frame-specific decoders on anchor frames sampled every $N_g$ frames. In the decoder training phase (right), the encoder is frozen, and separate lightweight decoders (or last layers) are trained independently for each frame group, enabling parallelization and efficient scaling to longer videos.

## 3.3 Method

### 3.3.1 Overview

Here, we will introduce our video encoding pipeline. SIEDD consists of a two-stage training process. First we train a shared encoder model using a small subset $N_s$ out of $N$ video frames ($N_s \ll N$). In the next stage, we freeze the trained encoder and only train separate decoder networks for each frame group $N_g$. In all our experiments, we held $N_s = N_g$.

### 3.3.2 Shared Encoder Training

We define the shared encoder as an MLP $f_\theta : \mathbb{R}^{\text{in}} \to \mathbb{R}^d$, which maps input coordinates to a latent representation. Each *decoder* $g_{\phi,i} : \mathbb{R}^d \to \mathbb{R}^3$, where $0 < i < N_s$, is also an MLP that maps the shared latent vector to an RGB output for frame $i$. Prior to the encoder, we

apply a positional embedding $\gamma : \mathbb{R}^2 \to \mathbb{R}^{\text{in}}$ to the 2D input coordinates. Both the encoder and decoder use the sine activation function [13] with frequency parameter $\omega = 30$ in all layers except the final one, which is left linear to produce the output. A SIEDD network is composed of a frozen positional embedding, followed by a shared encoder, and finally the $N_s$ decoders. This is defined as

$$h_{\phi,\theta}(x) = \text{concat}_i \left( g_{\phi,i} \circ f_\theta \circ \gamma(x) \right)$$

$$h : \mathbb{R}^2 \to \mathbb{R}^{N_s \times 3}$$

We initially fit a shared encoder by overfitting it to $N_s$ separate decoders on uniformly sampled keyframes from the whole video. To achieve this, we optimize

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \; L(h_{\phi,\theta}(x), y)$$

These trained decoders are used to initialize the weights of frame specific decoders in the next stage. We use the standard $L2$ loss for all of our experiments unless specified otherwise.

### 3.3.3 Discrete Decoder Training

We chunk our videos into groups of $N_g$ frames each. We take the trained shared encoder from stage-1 and freeze it, while proceeding to train individual decoders for each frame group. To improve the speed, decoder weights are initialized from the closest key frame's decoder weights from the shared encoder model. This method deviates from [61] where the shared encoder is also trained for unseen images. Note that since our encoder

is frozen, these decoders are not dependent on each other, allowing us to train them in parallel, across devices.

#### 3.3.3.1 Sharing Decoder Weights

Using the same architecture as the shared encoder model for the video frame fitting is highly parameter inefficient due to independent weights between similar frames. Therefore, in the second stage of the pipeline, we combine the $N_g$ separate decoder MLPs into a shared decoder for the frame group. However, the last layer of the decoder must remain separate to allow for precise prediction of pixel colors. This approaches vastly improves video compression. We employ BatchLinear layers to speed up matmuls in the decoder, allowing us to decode entire frame groups at once.

#### 3.3.3.2 Coordinate Sampling

Efficient coordinate sampling was critical to achieve low encoding time for SIEDD. A forward pass using a 1080p image's (x, y) coordinates of shape $(1920 \cdot 1080) \times 2$ consumes excessive GPU VRAM and is extremely computationally heavy. While this is necessary to reconstruct the image, we find sampling can greatly speed up training. The $N$ coordinates we use while training are effectively different data samples and it is not necessary to train on each one in every iteration. We use uniform random sampling to sample $C$ points where $C \ll H \cdot W$. To reduce the overhead of random sampling, we shuffle all coordinates once per epoch and iterate through them sequentially, with each minibatch containing $C$ frame coordinates. We also found an approximate lower limit for C, which was $\approx \frac{H \cdot W}{1024}$ which

accelerates training while causing minimal loss to reconstruction quality. For 1080p, this decreases our batch size from 2e6 to 2e3, a 1000× reduction.

### 3.3.4 Compression Pipeline

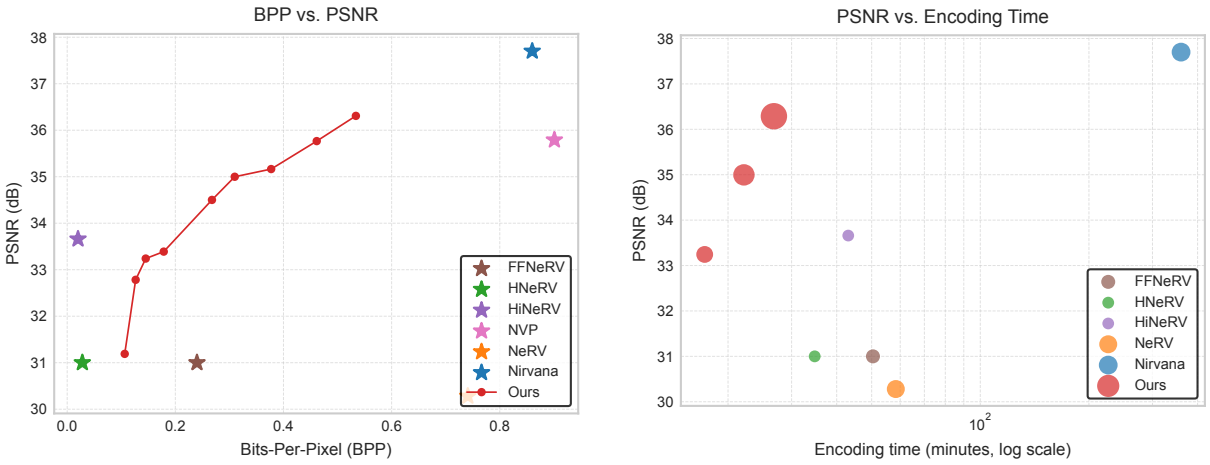
The shared encoder is not quantized due to its insignificant contribution to the overall parameters. The decoders for all video frames undergo post-training quantization. In particular, Half Quadratic Quantization (HQQ) [64] was the optimal method for compressing model weights while retaining reconstruction quality. An important note is that the last layers of the decoders, which produce the output pixels, are kept unquantized to preserve reconstruction quality. After the model weights are quantized, they are compressed further using huffman encoding. Finally, the resulting bitstream is saved to the disk using lzma-based compression.[1]

## 3.4 Experiments

### 3.4.1 Datasets and Implementation

We perform experiments using multiple datasets including UVG [18], DAVIS, and Big Buck Bunny. We experimented on 7 UVG-HD videos (Beauty, Bosphorus, HoneyBee, Jockey, ReadySteadyGo, ShakeNDry, and YachtRide) containing a total of 3900 1920×1080 video frames. For 4k experiments, we used the same 7 UVG-4k videos with image sizes of 3840 × 2160. For the DAVIS dataset, we use 10 1080p videos from the validation set (blackswan, bmx-trees, boat, breakdance, camel, car-roundabout, car-shadow, cows, dance-

[1] `https://github.com/lucianopaz/compress_pickle`




(a) BPP vs. PSNR on UVG-HD

(b) PSNR vs. encoding time (log-minutes).

Figure 3.2: Comparison of our method and baselines on UVG. Left: rate–distortion; Right: speed–quality trade-off.

twirl, and dog). In total, this subset of DAVIS contains 748 frames. Finally, to study long video performance, we use one video from Youtube-8M [47] about Mario Kart. The video was downloaded at $1280 \times 720$ resolution and the first 4000 frames were used. We use ffmpeg to convert the raw YUV files into PNG frames. Further details are included in the Appendix.

We use the standard metrics of PSNR (Peak signal to noise ratio) and SSIM (Structural similarity) as our primary measures of video quality. We use BPP (bits per pixel) to measure the compression efficiency. All encoding time measures for all models are for a single NVIDIA RTXA5000 GPU. Additional quality and speed metrics are included in the supplementary.

### 3.4.2 Setup

Our models were implemented using PyTorch and experiments were run using NVIDIA RTX A5000 GPUs. We used the schedule-free AdamW optimizer [76] which provided stable

training when compared to commonly used optimizers such as those of the Adam family. We define 3 SIEDD models: SIEDD-S with a model dimension of 512, SIEDD-M with a model dimension of 768, and SIEDD-L with a model dimension of 1024. For all 3 models, the shared encoder has 1 hidden layer while the decoders contain 3. The number of iterations for the shared encoder training and the video frame training were kept equal at 20000.

### 3.4.3 HD Video Reconstruction

We showcase SIEDD's ability to efficiently encode 1080p videos from the UVG-HD dataset. In Figure 3.2a, we compare the rate-distortion trade-off of our method against several frame-based Video-INR baselines [3, 58, 63, 68, 69], all given a maximum encoding time budget of 1 hour for a single 600-frame, 1080p clip. SIEDD achieves high-quality reconstructions at competitive compression levels, outperforming several methods that either sacrifice quality (e.g., HiNeRV[68]) or require substantially more bandwidth (e.g., Nirvana [63], FFNeRV[69]). Figure 3.2b visualizes the trade-off between PSNR and encoding time (log-scale), with marker size proportional to BPP. SIEDD consistently achieves the best quality-per-time ratio. Notably, it is approximately 20× faster than the closest high-quality baseline while operating at lower or comparable bitrates. Unlike FFNeRV [69] or NIRVANA [63] which cluster in the high-time, high-rate regime, SIEDD pushes the pareto front forward—offering both efficiency and fidelity.

A key observation is that some methods like NeRV[3] and HNeRV[58] attain decent reconstruction quality but at significantly higher encoding costs, making them less

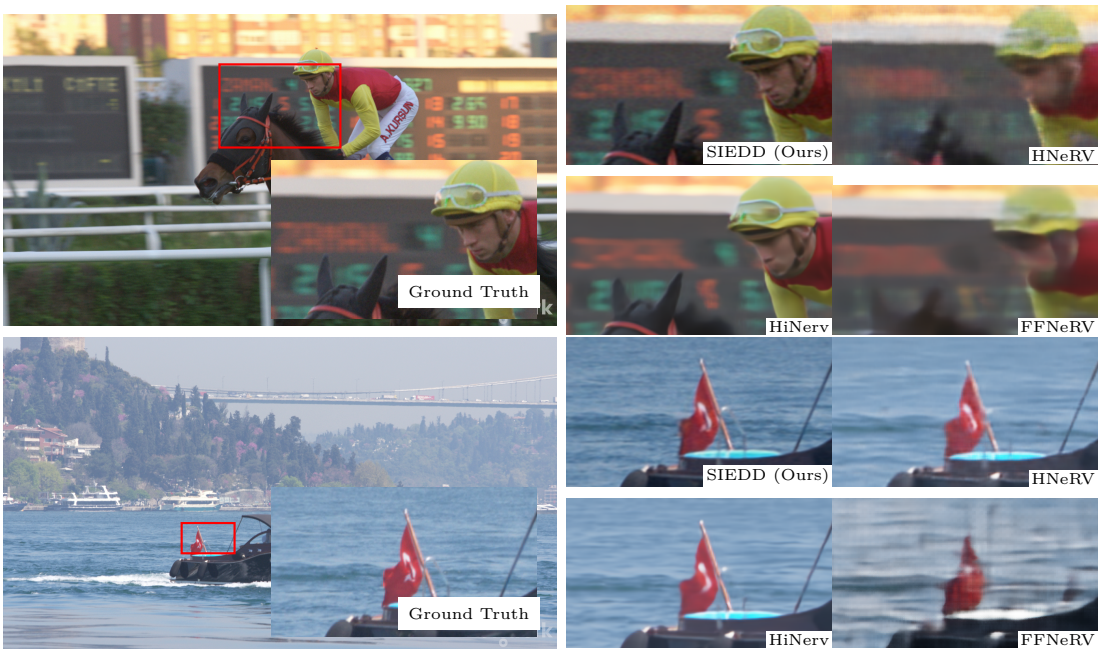



Figure 3.3: Video Reconstruction Visualization. We compare the reconstructions of SEIDD with other baselines for 2 UVG-HD Videos: Jockey (top) and Bosphorus (Bottom). We can clearly see that SEIDD produces much sharper reconstructions, staying true to the ground truth.

practical. SIEDD's performance illustrates the advantage of its shared encoder design, efficient sampling strategy, and decoder parallelizability. This positions SIEDD not only as a competitive Video-INR in terms of compression, but as a viable candidate for real-world, time-sensitive deployment scenarios. In Fig 3.3 we visualize the reconstructed frames from SEIDD and other baselines for "jockey" and the "bosphorous" sequence from UVG-HD [18] dataset. We can clearly see that our method preserves high frequency details and does not have the *smudging/smoothing* effect that is visible in the baselines.

### 3.4.4 4K Video Reconstruction

We additionally show SIEDD's ability to encode 4k video and present the results for 3 different model configurations in 3.3. We can see that SEIDD clearly outperforms all the baselines in terms of reconstruction quality with excellent encoding speeds. In fact, this is

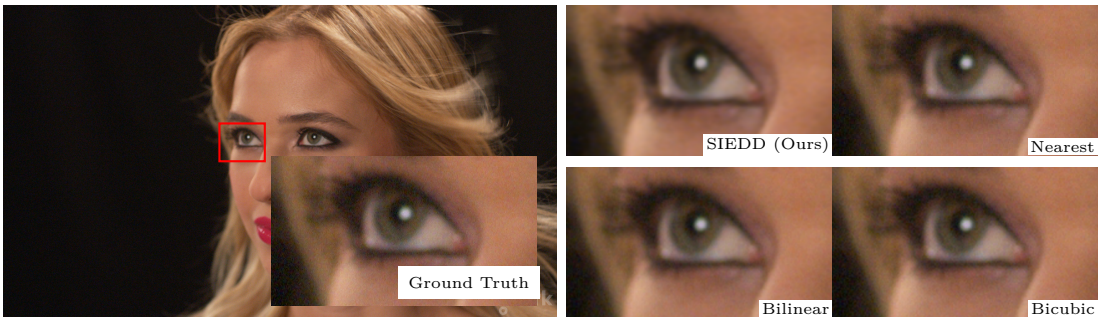



Figure 3.4: Super resolution visualization between baseline methods (nearest, bilinear, bicubic) and SIEDD-L. Upon close inspection, it is visible that the noise present in the ground truth image is not represented by SIEDD compared to other methods.

Table 3.1: Shared Encoder Weight Transfer from UVG-HD to DAVIS

| Method | PSNR | SSIM | Time (s) |
|---|---|---|---|
| SIEDD-M | 30.71 | 0.82 | 442 |
| From UVG-HD | 30.63 | 0.83 | 158 |

the first Video-INR method that is able to encode a 600-frame 4K video in under an hour, achieving upto 30X faster times compared to baselines.

We additionally test two SIEDD-L models on UVG-4k with $3 \times 3$ and $6 \times 6$ patches. We use square patches of size $p \times p$, meaning that for every image coordinate, SIEDD will output the colors of $p^2$ pixels. This reduces the total number of image coordinates by a factor of $p^2$ while increasing the number of parameters in the last layer of the decoder by $p^2$ as a tradeoff. As shown in 3.3, decoding speed drastically improves with associated trade-offs in encoding time, and compression.

#### 3.4.4.1 Shared Encoder Transfer

We also show the possibility of shared encoder transfer across datasets in 3.1. To show this, we use the weights of the shared encoder trained on UVG-HD and use it to train DAVIS in place of the shared encoder training process. We compare this to training

Table 3.2: Long Video Results

| Method | PSNR | BPP | Time (s) |
|---|---|---|---|
| SIEDD-M | 31.10 | 0.325 | 6408 |
| HNeRV | 24.78 | 0.035 | 7800 |
| NeRV | 23.94 | 0.12 | 11000 |
| HiNeRV | 26.68 | 0.04 | 6860 |

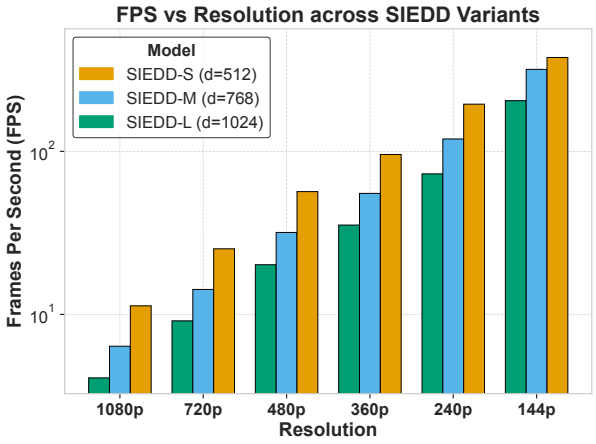
Table 3.3: 4K reconstruction results on UVG-4K. SIEDD variants compared with NeRV, HiNeRV, and Nirvana. Encoding Time reported in seconds.

| Method | PSNR | SSIM | BPP | FPS | Encoding Time (s) |
|---|---|---|---|---|---|
| SIEDD-M | 33.41 | 0.83 | 0.078 | 1.16 | 2574 |
| SIEDD-M $N_g = 10$ | 34.42 | 0.84 | 0.147 | 1.14 | 2223 |
| SIEDD-L, $N_g = 10$ | 35.41 | 0.85 | 0.256 | 0.76 | 3033 |
| 3×3 Patches | 34.52 | 0.84 | 0.275 | 7.49 | 3160 |
| 6×6 Patches | 33.70 | 0.82 | 0.331 | 21.70 | 4211 |
| NeRV | 28.67 | 0.83 | 0.486 | 6.3 | 3660 |
| HiNeRV | 30.47 | 0.86 | 0.0051 | 21.0 | 10200 |
| Nirvana | 35.18 | 0.93 | 0.270 | 28.4 | 75600 |

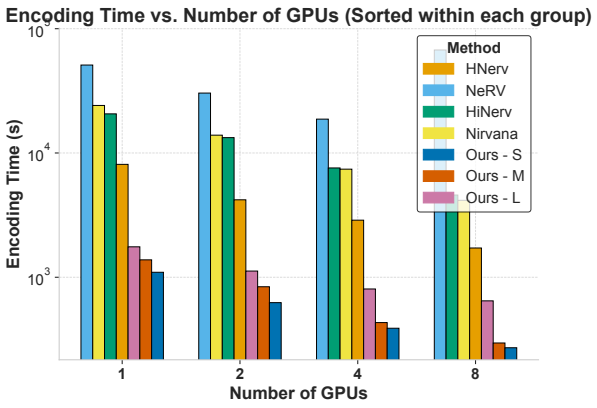
SEIDD on DAVIS from scratch and find that using a shared encoder from UVG-HD provides similar reconstruction quality while reducing the encoding time significantly due to the lack of shared encoder training. This points us towards a broader idea - that the shared encoder is like an ever growing "prior" of videos which can be updated when required, else used to perform zero-shot transfer to unseen videos.

### 3.4.5 SuperResolution

We demonstrate SIEDD's capability for continuous-resolution decoding by comparing its super-resolved outputs against traditional interpolation methods. As shown in Figure 4, SIEDD-L reconstructs fine details such as eyelashes and iris contours with greater fidelity compared to nearest, bilinear, and bicubic upsampling. Interestingly, the noise texture




(a) FPS vs. Resolution on UVG-HD.




(b) Encoding time and GPU Parallelization for SIEDD and baselines.

Figure 3.5: Comparison of our method and baselines on UVG. Left: rate–distortion; Right: speed–quality trade-off.

seen in the ground truth is absent in SIEDD's reconstruction, suggesting that the model implicitly denoises while super-resolving. Unlike baselines that rely on fixed grid interpolation, SIEDD leverages learned implicit mappings to reconstruct semantically meaningful detail, making it well-suited for applications requiring resolution-adaptive decoding.

### 3.4.6 Any Resolution Decoding

Unlike Frame-based Video-INRs our method takes 2D positional grid as input which allows us to control the spatial resolution of the decoded output. In Figure 3.5a we measure how the decoding speed is impacted at different resolutions and observe a consistent pattern of faster decoding at lower resolutions.

### 3.4.7 GPU parallelization

We evaluate how encoding time scales with the number of GPUs for various Video-INR baselines and our SIEDD variants. As shown in Figure 3.5b, SIEDD exhibits near-

Table 3.4: Ablation studies: (a) Effect of coordinate sampling rate on reconstruction quality and encoding time. (b) Effect of shared encoder training iterations on reconstruction quality.

**(a)** Sampling rate vs. quality and encoding time.

| Sampling Rate | PSNR (dB) | SSIM | Time (s) |
|---|---|---|---|
| 1/128 | 33.30 | 0.766 | 5318.51 |
| 1/256 | 33.28 | 0.765 | 2790.15 |
| 1/512 | 33.27 | 0.765 | 1556.00 |
| 1/1024 | 33.24 | 0.765 | 887.49 |
| 1/2048 | 32.72 | 0.758 | 618.18 |

**(b)** Shared encoder iterations vs. quality.

| Shared Iters | PSNR (dB) | SSIM |
|---|---|---|
| 500 | 35.15 | 0.886 |
| 2000 | 35.23 | 0.886 |
| 5000 | 35.27 | 0.886 |

linear scaling with increasing GPU count. Specifically, our largest model (SIEDD-L) shows a consistent drop in encoding time from 1 GPU to 8 GPUs, highlighting the effectiveness of parallel decoder training across frame groups.

Compared to baselines like NeRV [3] and HiNeRV [68], which show limited gains with more GPUs due to their sequential or monolithic training structure, SIEDD benefits directly from architectural parallelism. For example, at 8 GPUs, SIEDD-L achieves a $8\times$ speedup relative to its single-GPU decoder training time, while NeRV improves by only $4\times$. Even smaller SIEDD variants outperform stronger baselines like Nirvana, despite using fewer parameters and lower computational overhead.

This scalability makes SIEDD a compelling choice for high-resolution or long video scenarios where encoding throughput is critical.

### 3.4.8 Long Video Training

To test the scaling of our model on the temporal axis, we train on a sequence of 4000 frames from "mario-kart" sequence from Youtube-8M dataset. The results are presented in Table 3.2 and we see that SEIDD outperforms other baselines with great encoding speeds.

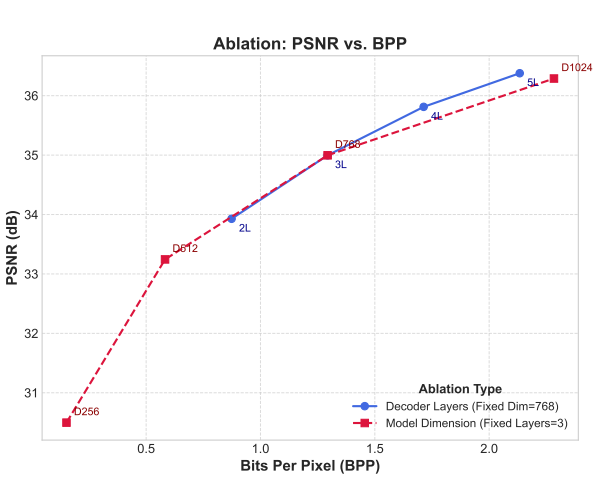
### 3.4.9 Ablation Analysis

#### 3.4.9.1 Sampling Rate

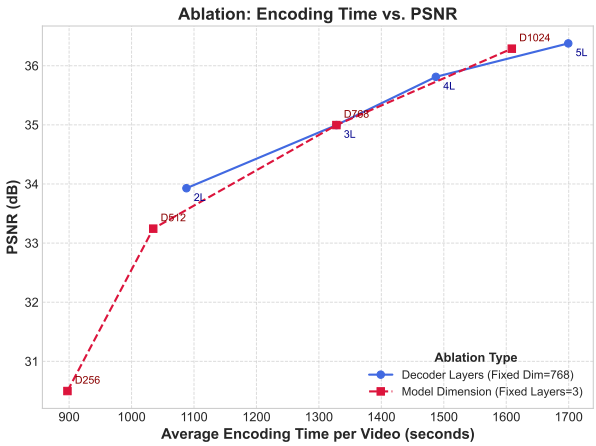
We perform an ablation study on the coordinate sampling rate to understand its impact on encoding time and reconstruction quality. As shown in Table 3.4(a), reducing the sampling rate from 1/128 to 1/2048 leads to a significant drop in encoding time—from 5318s to just 618s—demonstrating a nearly 9× speedup. Notably, the reconstruction quality (PSNR and SSIM) remains largely stable for moderate reductions (up to 1/1024), with only a minor degradation ( 0.05 dB PSNR). Beyond this, quality drops more noticeably, indicating diminishing returns. This trade-off highlights the effectiveness of our sparse coordinate sampling strategy, allowing users to balance compute budget and fidelity depending on application needs.

#### 3.4.9.2 Shared Encoder iterations

We study the effect of shared encoder training iterations on reconstruction quality in Table 3.4(b). As expected, increasing the number of training steps improves PSNR marginally—from 35.15 at 500 iterations to 35.27 at 5000—while SSIM remains largely unchanged. This suggests that the encoder converges quickly to useful low-frequency features, validating our design choice to limit encoder training for faster overall encoding.




(a) BPP vs. PSNR varying model dim and layers




(b) Varying model dim and layers with encoding time

Figure 3.6: Ablation study on decoder architecture. (a) PSNR vs. BPP when varying decoder layer count (with fixed dim = 768) and model dimension (with fixed 3 decoder layers). (b) PSNR vs. encoding time, illustrating trade-offs in reconstruction quality with increased decoder depth or width. More layers improve quality marginally with negligible cost, while increasing model dimension significantly boosts quality at the expense of higher encoding time.

#### 3.4.9.3 Model Layers and Layer dimension

We investigate how architectural capacity—specifically the number of decoder layers and model dimensionality—affects the rate-distortion performance and encoding time of SIEDD. The results are summarized in Figure 3.6. In Figure 3.6a, we vary the decoder MLP dimension (256, 512, 768, 1024) and also vary the number of decoder layers (2, 3, 4, 5). We observe a consistent improvement in PSNR as dimensionality increases, along with a mild rise in BPP. Notably, the 1024d variant achieves the highest reconstruction quality while maintaining competitive compression, demonstrating that increased latent capacity allows the decoders to model finer visual details more effectively. In contrast, Figure 3.6b explores the impact of model hyperparameters on encoding speed. While larger decoders yield marginal gains in PSNR, the returns diminish beyond 3 layers and d=768. More importantly, larger networks incur significant increases in encoding time

due to slower convergence and additional compute. Overall, we find that increasing width (dimensionality) provides better quality–bitrate trade-offs, while deeper networks primarily affect training time. These trends guided our default configuration (768d, 3-layer decoder), striking a balance between efficiency and quality.

## 3.5 Discussion

We present SIEDD, a fast and scalable Video-INR architecture that leverages shared representations and per-group decoders to dramatically reduce encoding time. By training the encoder on sparse anchor frames and freezing it for the rest of the video, SIEDD enables parallelized decoding and efficient representation learning—achieving up to 30× faster encoding compared to existing INR methods. Our coordinate-based formulation allows continuous-resolution decoding, and our use of simple MLPs makes the architecture amenable to post-training quantization using state-of-the-art compression techniques like HQQ and BNB.

While SIEDD significantly advances the practicality of INR-based codecs, a few areas remain open. Inference-time decoding, especially for high-resolution and high-framerate video, could be further accelerated with fused matmul kernels and specialized hardware-aware optimizations. Our current quantization is post-training; future work could explore quantization-aware training (QAT) to further improve compression without sacrificing fidelity. Finally, while our shared encoder shows strong transfer potential, systematic studies on cross-video generalization and zero-shot inference remain a rich direction for exploration.

## 3.6 Appendix

### 3.6.1 Experimental Baseline Settings

Note that all the following models were trained with a hard limit of 60 minutes on encoding time, on an RTXA5000. We chose to train from scratch as the learning rate schedule has a significant effect on final quality.

#### 3.6.1.1 NeRV

We use the NeRV-L setting from the original paper [3]. This comes with 5-NeRV blocks with upscale factors of $[5, 3, 2, 2, 2]$ for UVG-HD and $[5, 3, 2, 2, 2]$ for UVG-4K. We use the standard hyperparameter settings from the paper.

#### 3.6.1.2 HiNeRV

Since HiNeRV [68] training can be quite slow, we choose to use the HiNeRV-S configuration from the original paper and add an additional block to make it work for 4K.

#### 3.6.1.3 FFNeRV

To balance quality and encoding time, we choose the FFNeRV configuration with $C_1, C_2, S$ as $(112, 896, 54)$


Figure 3.7: Visual comparison between the ground truth frame, patching output, and the default SIEDD-L model output on UVG-4k ShakeNDry. Pixels within a patch are always very similar, causing a pixelated effect.

#### 3.6.1.4 HNeRV

We use the default model configuration for UVG-HD as reported in the paper and added an additional block for UVG-4K. Due to architectural constraints, HNeRV [58] output is restricted to $960 \times 1920$ ( 12% less) for UVG-HD and $3600 \times 2160$ for UVG-4K ( 7% less).

### 3.6.2 Additional Qualitative Results

We also provide qualitative results from the UVG-4k experiments. We compare a ground truth frame from ShakeNDry with the result from SIEDD-L, SIEDD-L with a $3 \times 3$ patch, and SIEDD-L with a $6 \times 6$ patch in 3.7. While patching improved the decoding speed enormously, it comes at a cost of higher encoding time and a loss in reconstruction quality. The pixels within a patch are relatively uniform, causing an effect similar to nearest-neighbor upsampling.

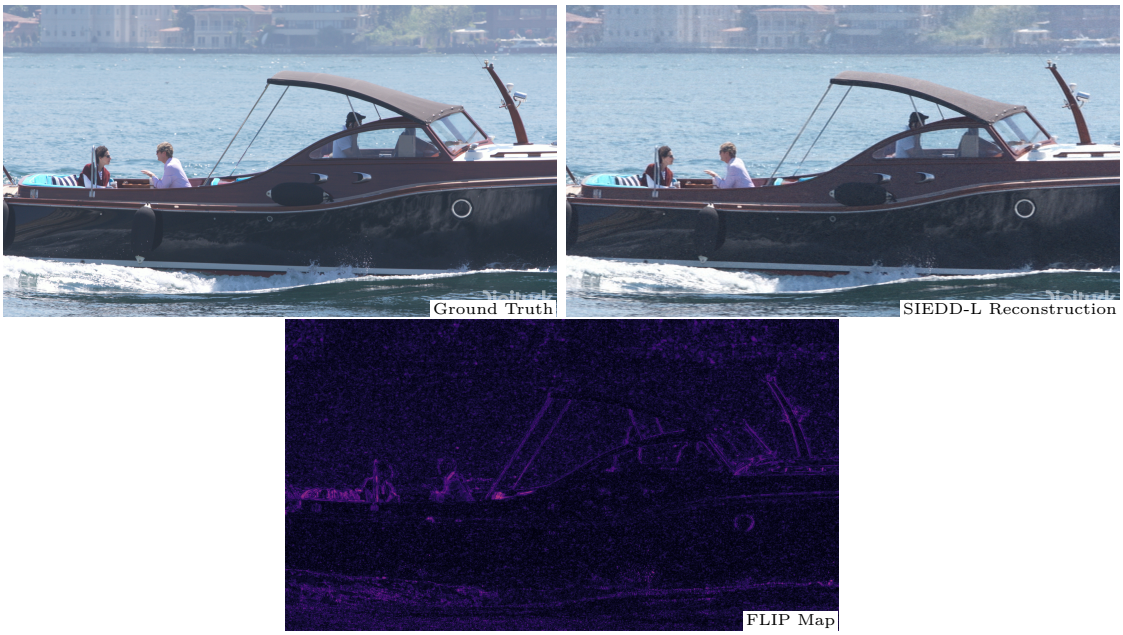



Figure 3.8: FLIP Visualization on UVG-HD YachtRide using SIEDD-L

Table 3.5: VMAF and FLIP metrics on UVG-HD using SIEDD-S, SIEDD-M, and SIEDD-L

| Model | FLIP | VMAF |
|---|---|---|
| SIEDD-S | 0.100 | 66.69 |
| SIEDD-M | 0.080 | 80.85 |
| SIEDD-L | 0.068 | 87.75 |

### 3.6.3 Additional Reconstruction Metrics

To provide additional metrics to ensure high quality video reconstruction, we use FLIP [77] and VMAF [78]. We evaluate these metrics on SIEDD-S, SIEDD-M, and SIEDD-L on the UVG-HD dataset. We visualize the FLIP error map in 3.8 on a frame from UVG-HD YachtRide and list the FLIP and VMAF metrics in 3.5.

### 3.6.4 Effect of Group Size

To study the effect of the group size parameters $(N_g, N_s)$ on SIEDD, we ran an ablation experiment on SIEDD-M in 3.6. SIEDD-M uses $N_g = N_s = 20$ by default, so

Table 3.6: Group Size Ablation

| Group Size | PSNR | SSIM | bpp | Time (s) |
|---|---|---|---|---|
| 10 | 36.54 | 0.907 | 0.587 | 14915 |
| 15 | 35.67 | 0.892 | 0.402 | 11171 |
| 20 | 34.84 | 0.878 | 0.297 | 9296 |
| 25 | 34.52 | 0.871 | 0.255 | 8720 |
| 30 | 34.14 | 0.864 | 0.218 | 8225 |

in this experiment, we test the values of 10, 15, 25, and 30. We encode and evaluate the metrics of SIEDD-M with each group size on the UVG-HD dataset.

The group size hyperparameter is negatively correlated with the reconstruction quality (PSNR, SSIM) and compression (bpp) and is positively correlated with the encoding time. This is because with a larger group size, more frames share a single decoder, decreasing the number of parameters (and the bpp) for the video. A larger group size also means fewer decoder training loops, speeding up the encoding time. However, these come at a heavy cost to the reconstruction quality, and we chose 20 as the balanced default for SIEDD.

### 3.6.5 Decoders with Low Rank Adaptation (LoRA)

Table 3.7: LoRA Decoder Results

| LoRA Type | LoRA Rank | PSNR | SSIM | bpp | fps | Time (s) |
|---|---|---|---|---|---|---|
| None (SIEDD-M) | - | 34.84 | 0.878 | 0.297 | 6.27 | 9296 |
| Sin LoRA | 1 | 35.65 | 0.891 | 0.385 | 6.30 | 42802 |
| Sin LoRA | 2 | 35.78 | 0.893 | 0.432 | 6.31 | 42781 |
| Sin LoRA | 4 | 35.96 | 0.896 | 0.523 | 6.36 | 42614 |
| Sin LoRA | 8 | 36.22 | 0.900 | 0.703 | 6.34 | 43385 |
| LoRA | 1 | 35.58 | 0.890 | 0.385 | 6.30 | 41701 |
| LoRA | 2 | 35.71 | 0.892 | 0.431 | 6.30 | 42286 |
| LoRA | 4 | 35.87 | 0.894 | 0.523 | 6.33 | 42389 |
| LoRA | 8 | 36.10 | 0.898 | 0.700 | 6.34 | 42787 |

In SIEDD, the shared decoder has no independent parameters for each output frame with the exception of the last layer which produces the output coordinates. To introduce

| Method | Bits | PSNR | SSIM | bpp | fps |
|---|---|---|---|---|---|
| None | 32 | 35.00 | 0.880 | 1.294 | 6.38 |
| BNB | 4 | 30.19 | 0.789 | 0.184 | 6.25 |
| BNB | 8 | 34.96 | 0.879 | 0.335 | 3.09 |
| HQQ | 4 | 32.54 | 0.838 | 0.222 | 6.26 |
| HQQ | 6 | 34.97 | 0.879 | 0.310 | 6.26 |
| HQQ | 8 | 35.16 | 0.882 | 0.378 | 6.26 |
| Post | 4 | 15.55 | 0.418 | 0.178 | 6.39 |
| Post | 5 | 27.34 | 0.758 | 0.220 | 6.38 |
| Post | 6 | 33.13 | 0.852 | 0.263 | 6.39 |
| Post | 7 | 34.51 | 0.873 | 0.306 | 6.38 |
| Post | 8 | 34.87 | 0.878 | 0.348 | 6.37 |

Table 3.8: Quantization Sweep on SIEDD-M with Different Methods

additional parameters that are unique to each video frame to the decoders, we use low rank adapters on each decoder layer.

In particular, we test traditional LoRA [79] and Sine LoRA [80]. Sin LoRA introduces a sine nonlinearity when multiplying the low rank matrices, allowing for more representative abilities. We find that the Sin LoRA provides marginal boosts in reconstruction quality (PSNR) compared to traditional LoRA.

Due to the materialization of a unique activation for each video frame in the forward pass, using LoRA adapters on the decoder layers increase encoding time by a significant value, as seen in 3.7.

### 3.6.6 Quantization Results

Due to the importance of quantization in our compression pipeline, we experimented with different quantization methods. Traditional post-training quantization (PTQ) involves scaling the tensor from 0 to $2^b$ where b is the number of bits to quantize to, casting this tensor to an integer, and storing it. While this method works for larger $b$ values, performance degrades rapidly at lower precision integers as shown in 3.8. We also show the

| Bit | bpp | PSNR | SSIM |
|---|---|---|---|
| 4 | 0.106 | 31.19 | 0.812 |
| 5 | 0.127 | 32.78 | 0.841 |
| 6 | 0.145 | 33.24 | 0.849 |
| 8 | 0.179 | 33.39 | 0.851 |

(a) SIEDD-S

| Bit | bpp | PSNR | SSIM |
|---|---|---|---|
| 4 | 0.222 | 32.82 | 0.843 |
| 5 | 0.268 | 34.50 | 0.872 |
| 6 | 0.310 | 35.00 | 0.880 |
| 8 | 0.377 | 35.16 | 0.882 |

(b) SIEDD-M

| Bit | bpp | PSNR | SSIM |
|---|---|---|---|
| 4 | 0.380 | 34.00 | 0.865 |
| 5 | 0.461 | 35.77 | 0.895 |
| 6 | 0.534 | 36.31 | 0.903 |
| 8 | 0.644 | 36.49 | 0.905 |

(c) SIEDD-L

Table 3.9: Quantization Sweep for HQQ on SIEDD-S, SIEDD-M, and SIEDD-L

performance of BNB [65] which has 4 bit and 8 bit implementations. However, the best performing method at low precision is hqq [64] which shows minimal degregation in PSNR and SSIM compared to other methods at low bpp.

To further show the limits of hqq in the SIEDD architecture, we tested the reconstruction quality against bpp for each of the SIEDD-S, SIEDD-M, and SIEDD-L models in 3.9. As 6 bit gave a negligible (<0.2 PSNR) performance drop compared to 8 bit, this became the baseline for the SIEDD models.

### 3.6.7 Decoding

Decoding a coordinate based model such as SIEDD is an extremely computationally intensive task. A naive implementation will result in out-of-memory errors and a low FPS. To make 1080p and 4k decoding possible, the forward pass must be batched in terms of coordinates and the separate decoders. In our implementation, we chunk the coordinates into 8 separate forward passes, and within each forward pass, the decoders are run one at a time. For 4k decoding, we instead chunk the coordinates into 32 forward passes of

the model, ensuring that 4k videos can be encoded using an A5000 GPU. In addition, the SIEDD model parameters are converted to bfloat16 to accelerate decoding speed. To calculate the decoding speed (fps), we record the time required to run the forward pass on all coordinates on the GPU and send the frames to the CPU.

For our LoRA experiments, the FPS would normally be extremely low due to the materialization of activations for each individual frame, as opposed to one activation for the frame group. As a result, LoRA decoding performance would have been even worse. However, we split the shared decoder and LoRA adapters into separate decoder layers, effectively splitting the decoder into $N_g$ parallel MLPs. This allows the FPS to be on par with SIEDD without LoRA adapters.

### 3.6.8 Video Denoising

Table 3.10: Video Denoising PSNR Results

| | All White | All Black | Salt & Pepper | Random |
|---|---|---|---|---|
| Baseline | 28.08 | 29.69 | 28.07 | 30.95 |
| Gaussian Blur | 35.25 | 36.03 | 35.56 | 36.71 |
| Median Blur | 36.47 | 36.46 | 36.47 | 36.47 |
| SIEDD-L | 35.66 | 35.66 | 35.66 | 35.66 |

We also showcase SIEDD's ability to carry out image denoising in 3.10. To achieve this, we add different types of image noise to UVG-HD, encode the video using traditional denoising methods and SIEDD, and compare the output quality to the original frames. To add noise, we tested setting certain pixels to all white, all black, adding salt and pepper noise, and random noise. We added noise to $10^4$ pixels of each frame using these different methods. The traditional denoising methods include gaussian and median blurs. We also record the baseline which is the PSNR of the noisy image and the ground truth. The results

show a comparable performance to traditional denoising techniques such as gaussian and median blurring with the added benefits of SIEDD.

# Chapter 4: Latent INR: A framework for semantically meaningful video compression

In the previous chapter [1] , we discussed in detail about building efficient and compressed functional video representations which were fast to encode (compared to current methods) and offer real time decoding. However, one wonders whether these functional mappings are really "representations"? Maybe yes, if we use a narrow definition of the word, where we are only concerned about efficient compression. But a codec of the future entails much more. We expect not just faithful reconstructions, but some form of built-in understanding, a proxy for the data that enables many semantic tasks. In this chapter, we showcase Latent-INR, a new INR based architecture for videos that not only achieves compression, but offers us a learnt latent that abstracts the underlying weights, allowing us to perform tasks like video retrieval, frame interpolation and even interfacing with video-LLMs to enable chat.

[1] Work done jointly with Anubhav Gupta. I was responsible for the idea, philosophy, and implementations of video compression, Video-LLM and interpolation. Anubhav Gupta was responsible for video retrieval task.

## 4.1 Introduction

In today's age of content explosion, large quantities of data are created every second, and storing them reliably and efficiently is of utmost importance for many applications. A scalable compression technique enables companies to provide better services at reduced cost and helps the end consumer by improving their access to high-fidelity data in addition to decongesting the network. Since the early 90s, several compression techniques have been created and widely deployed for this exact purpose. Out of these, JPEG [81] for images, HEVC [53], AV1[6], and H.264 [82] for videos have emerged as the most popular choices, owing to their simple design and scalable performance.

In the past decade, the rise of deep learning led to a renaissance in computer vision, eventually impacting the visual data compression landscape [83, 84, 85]. Despite their success, these ML-based codecs have not seen widespread adoption like traditional codecs. This is in part due to failure to generalize, since ML codecs trained on large datasets can give sub-optimal compression for data points that differ significantly from their training set [86, 87]. Implicit Neural Representations (INR) attempt to avoid the generalization issue by operating internally. Instead of training large models that learn to identify *general* patterns in training data and apply them to specific out-of-distribution data, implicit techniques involve training a small model to exploit the *specific* patterns for the given data point. That is, for video compression, this approach would train one network per video, and for image compression, it would train one network per image. The resulting model is essentially a function that represents the underlying signal in spatial/temporal space.

Despite these advances, neural video compression remains unsolved. Various methods

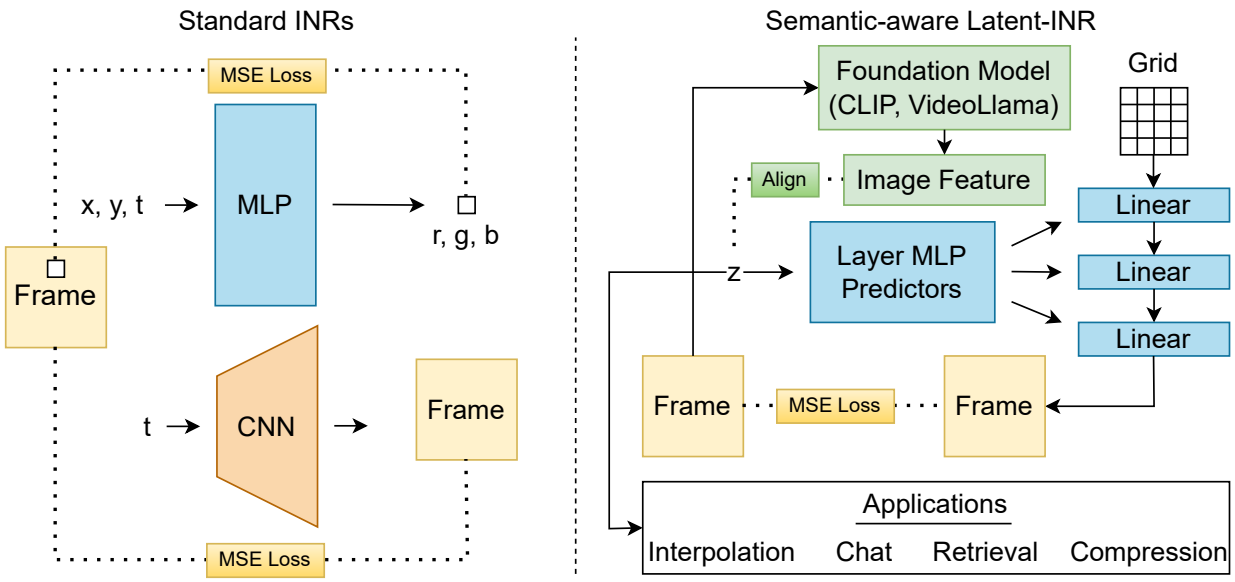



Figure 4.1: Existing INRs for video (left) typically take some time-coordinate, or time and positional coordinates and train a single network to reconstruct a video. In contrast to these, we propose an INR system where a dictionary of implicit latent codes is learned for a video, one latent per frame. The latents are aligned to the image features of a large vision model, while simultaneously an INR system is learned which, given these latent codes, generates a positional INR which can reconstruct the frame. With this framework, we successfully develop an INR which performs both reconstructive tasks like compression, and semantic downstream tasks like retrieval and interactive chat.

address issues of compression quality [3, 88], but two crucial questions remain unanswered – (i) how to scale for longer videos given architectural rigidity and (ii) how to reduce long encoding time due to training a network for every video. Although recent works make some progress for these [63], the training time is still quite long, and INR behavior for lossy compression is not well-understood [89], limiting potential for practical adoption.

Furthermore, these approaches for INR tackle only one axis of the problem, i.e., how to formulate video INRs with the primary goal of compression. These aim to solve problems like long encoding time directly, by reducing it. In contrast to these works, we instead aim to justify the compute and time needed to train implicit representations. So, as a step towards ML-based codecs with compelling real-world potential, we present Latent-INR – a

new flexible framework for formulating video INRs, where in addition to compression, the INR enables downstream tasks like retrieval and video question answering, without the need to decode the video. Our framework consists of two parts: (i) a dictionary of learnable latents, one for each frame, and (ii) a set of hypernetworks learned on the entire video which, given a latent as input, predict frame-specific weight modulations on the shared base network. This shared base takes a spatial coordinate grid as input and outputs the specific frame

This design allows us to separate the spatial and temporal aspects of the video by modeling them separately. We can view the set of hypernetworks as a base model that learns the general structure and style of the video, while each learned latent conditions it to output a specific frame. The latent here acts as a proxy for the weights of the frame-specific INR. This property is apparent from the video interpolation ability of our model - a task that other video INR representations struggle to perform. Like other video INRs, our method is competitive for compression, but uniquely retains the properties of original coordinate-based INR. That is, our continuous representations of frames allows for spatial interpolation, which can be leveraged for superresolution and a decoding paradigm we refer to as "any-resolution inference." That is, at inference/decoding time, our same model, with no changes to latents or architecture, can decode a video at any resolution - a key feature missing from traditional codecs. This latent is also quite flexible, and according to the procedure shown in Figure 4.1, we can align it with the features from a large vision model, such as CLIP [90] to encode the visual semantics of the frame while retaining nice properties such as alignment with CLIP text embeddings. This allows for a whole spectrum of applications, including frame, concept, and whole video retrieval with text queries.

In summary, our framework gives that extra edge apart from compression to ML-based codecs, paving the way for their widespread adoption. Concretely,

- We propose an auto-decoder latent-based framework with spatio-temporal decoupling for implicit video representations. Compared to other video INR methods, this is a new way of formulating the problem.
- Our system has good compression performance, competing well with other ML-based codecs for PSNR, BPP, and decoding speed while also enabling any-resolution inference.
- The learnt latent embeddings from our framework demonstrate internal generalization from the encoded dataset, achieving video interpolation, a task that other INR based methods struggle to achieve.
- We align our latents with large foundational models like CLIP [90], thus making our representations useful for retrieval tasks.
- We align our entire dictionary with video features for VideoLlama [91] to enable chat-style applications, including video question answering and captioning.

## 4.2 Related Work

**Implicit Neural Representations (INR's)** are a class of neural networks designed with the intention of representing a given data point or dataset perfectly rather than exploiting general patterns and generalizing for unseen data. SIREN [13] pioneered the use of periodic activations to train simple MLP's that worked well across images, SDF and

audio. This was followed by a host of works that improved the training process of INR's by making them faster [19, 92, 93] work across multiple scales [94] and encode multiple data points [95]. Models that used meta learning [26, 96] started gaining ground as they offered the advantages of compression along with generalization. [97, 98] further made improvements to this line of work by directly learning sparse-INR's leading to improved compression and improved optimization by dataset selection respectively.

**Hypernetworks** are a class of networks optimized for predicting parameters of another network, with the aim of generalizing across unseen tasks[99]. Some utilized these for scenes [100, 101, 102]. Trans-INR [28] introduced the paradigm of using a transformer based hypernetwork to convert data directly from image-space to INR's. [62] improved upon this idea and made the important observation that it is sufficient to modulate only the first hidden layer of an INR to represent a dataset of points. Unfortunately, these hypernetworks act on input data points which require test-time optimizations, making them unsuitable for compression tasks. [103] try to overcome this with an "auto-decoder" framework, where learnable latents represent a dataset of videos, with each latent corresponding to a single video, such that no encoder is needed. Others have investigated this paradigm for a variety of modalities[104, 105, 106]. Still, the lack of decoupling space from time prohibits the method from scaling to real-world videos.

**Video INRs** have recently gained popularity for compression. [3] was the first implicit representation which modelled a video as a function mapping the temporal coordinates to the corresponding frames. Later works [33, 58, 107, 108] iterated on this method, providing improvements in performance. [88] enhanced this concept by incorporating hashgrid [93] representations to speed up encoding times. NIRVANA [63] represented a video

using a series of smaller INR models trained in an autoregressive manner to scale for longer videos.

**Video Interpolation** has been a fundamental task in computer vision, helping in creating smoother visual experiences. Over the past few years, deep learning based methods have vastly improved the quality of these interpolations [109, 110].However, current INR-based video encoders lack this feature (see discussion in [58, 111], for example), hindering their widespread usage.

**Video Retrieval** is an essential process in the digital media landscape, where the objective is to efficiently search and extract specific video content from expansive datasets. The complexity of understanding and indexing diverse video content has traditionally posed significant challenges. However, with the advent of machine learning-based methods, there has been a remarkable improvement in both the accuracy and efficiency of video retrieval systems [112, 113, 114]. These advances are limited to systems requiring an additional model, which can act as a burden on the system as they do not compress the data.

## 4.3 Approach

### 4.3.1 Background

Implicit Neural Representations parameterize a function,

$$f_\theta : X \to Y \quad \text{where} \quad X = \{(x_i, y_i) | 0 \leq x_i \leq W, 0 \leq y_i \leq H\}$$

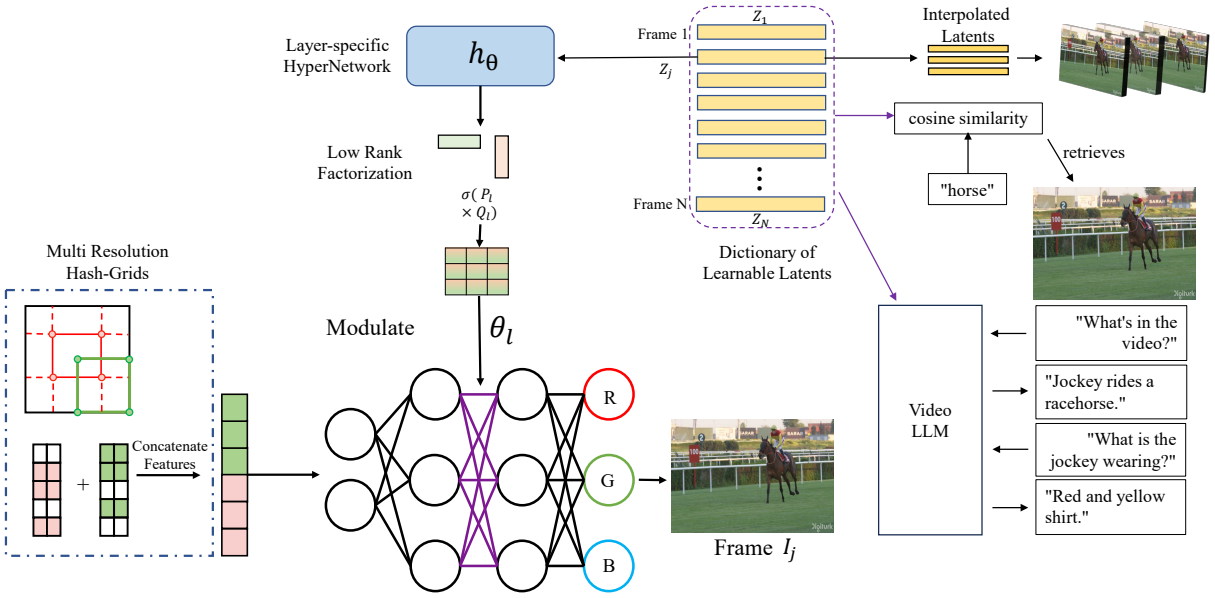



Figure 4.2: We propose a new framework for video INR models by decoupling the spatial and temporal aspects of modeling. Our framework consists of auto-decoder based learnable latents that modulate the base network using a hypernetwork, via low-rank modulation. Once encoded, the resulting latents act as a proxy for the underlying weights of the representation. On the right, we show the use of these latents for additional tasks like video interpolation. By aligning these latents to the embedding space of foundational models like CLIP and VideoLlama, we also perform retrieval and chat.

which represents a mapping between the coordinate space, with *height* H and *width* W, and the underlying signal $Y$. This formulation is usually trained with a standard MSE-loss: $||f_\theta(X) - Y||_2$. For a given video $V \in R^{N \times H \times W \times 3}$ containing $N$ frames, [13] represents them as pixels moving across time, i.e.,

$$f_\theta(x, y, t) = Y_t$$

Other formulations exist which learn frame-based [3] or patch-based [63] representation, yet in each of these formulations, the focus is on representing the underlying data, with the added motivation of compressing it. However, none of these systems are designed with the goal of making these representations, $f_\theta$, useful for downstream tasks [89, 115].

Instead, we utilize a learnable latent, $z$, as a part of an auto-decoder framework, along with a hypernet $h$ to not only compress but to create useful representations.

$$f_\theta((x,y)|\theta_t) = Y_t \quad \theta_t = h(z_t) \tag{4.1}$$

The resulting latent $z$ can be used for various downstream tasks like interpolation and retrieval, as we show in our work.

### 4.3.2 Latent-INR

Directly predicting the weights $\theta$ of the base network $f$, using the hypernet $h$, is expensive, parameter-heavy, and unsuitable for compression. Hence, we follow [116] [105] and instead predict low-rank matrices, which are then applied to the base network weights. This type of modulation acts as a form of subnetwork selection, analogous to systems proposed in [36] [117]. For a base network $f$ with $L$ layers, our formulation now looks like

$$\begin{gathered} f_\theta((x,y)|\theta_t^{l_1}, \theta_t^{l_2} ... \theta_t^{l_L}) = Y_t \\ \theta_t^l = \sigma(P^l \times Q^l) \cdot \theta^l \quad h_l(z_t) = [P^l, Q^l] \end{gathered} \tag{4.2}$$

where $\theta^l$ represents the weights of the $l$-th layer and $\theta_t^l$ denotes the modulated weights for frame $t$. Here, $\sigma$ signifies an activation function on the matrix-product of low rank matrices $P^l$ , $Q^l$, which are of dimensions $R^{N\times r}$ and $R^{M\times r}$, where $N \times M$ is the width of the base network $f_\theta$ and rank $r \ll (N, M)$. These matrices are responsible for adjusting the weights $\theta_l$ as dictated by the corresponding hypernetwork $h_l$. Note that all hypernetworks use the same latent $z_t \in R^D$ as input. The rank $r$ and the number of modulated layers essentially

act a hyperparameters that control the compression-performance trade-off.

### 4.3.3 Model architecture

In our experiments, both the base network $f_\theta$ and hypernetworks $h_l$ are feedforward MLP's that take in a coordinate input. Following [63], we also propose a variation to the base network with an additional convolutional up-sample block, which accepts coordinates of centroids as input and gives frame patches as output. We use the standard ReLU for base network and tanh for the hypernetwork as the respective non-linearities. The latents $Z$ are initialized to be a standard normal with small variance, as we found empirically that this made the convergence faster. The complete model architecture is presented in Figure 4.2. For more details, see Appendix.

### 4.3.4 Model Compression

We train this entire system end-to-end with MSE-loss as the objective function. Once trained, we apply a standard quantization to all network parameters, further reducing the required storage. Given $\phi$, a flattened parameter tensor, we transform it according to the following equations

$$\phi_i = \left\lceil \frac{\phi_i - \phi_{\min}}{2^b} \right\rfloor \quad \text{scale} = \frac{\phi_{\max} - \phi_{\min}}{2^b} \tag{4.3}$$

where the $\lceil \cdot \rfloor$ (round) operation converts its argument to the nearest integer as dictated by bit width $b$ of the quantization process. We also store the scale, $\phi_{\max}$, $\phi_{\min}$ and the parameter shapes. These quantized values for all parameters are concatenated and further

compressed using Huffman encoding.

### 4.3.5 Interpolation

Given a video of $N$ frames and a scale $\alpha$, the task of interpolation involves creating $\alpha \cdot N$ coherent frames. Once we encode a video using our framework, we perform linear interpolation on the frame latents $\{z_t\}$ and pass the resulting latent through the hyper-network. This gives us the weight modulation required in the INR, and the updated base network is used to obtain the interpolated frames:

$$z_{\text{inter}} = \beta_i \cdot z_t + (1 - \beta_i) \cdot z_{t-1} \quad Y_{\text{inter}} \quad = f_\theta(X; h(z_{\text{inter}})) \tag{4.4}$$

where,

$$\beta_i \in \left[\frac{1}{\alpha}, \frac{2}{\alpha}, ..., \frac{\alpha - 1}{\alpha}\right]$$

essentially generating $\alpha - 1$ frames between any two given frames. We train with held out frames and show results for $\alpha \in \{2, 4, 8\}$.

### 4.3.6 Downstream Tasks

**Retrieval.** Video retrieval involves searching and retrieving videos or clips from a large database based on similarity to given user search queries that are usually in the form of text. This can be viewed as a function $R$ mapping query $q$ to a set of corresponding videos $V$.

$$R : q \rightarrow V \tag{4.5}$$

The function $R$ can use any similarity measure like cosine, euclidean, or nearest neighbors to retrieve matches. We encode a dataset of videos using our Latent-INR framework and use the resulting trained latents as our frame level representations. To ensure these latents share the same space as the text queries, we add a cosine similarity loss between the latents and the CLIP image embeddings of the corresponding frames. Our encoding loss function is modified to be:

$$L = L_{\text{MSE}} + \lambda \cdot L_{\text{clip}}(Z_t, Z_t^{\text{clip}}) \tag{4.6}$$

where $Z_t^{\text{clip}}$ is the CLIP Image embedding of the input frame and $\lambda$ controls the strength of this loss. In all our experiments, $\lambda$ is set to 0.01.

**Chat.** We modify the formulation from retrieval slightly, aligning our dictionary of features to VideoLlama [91] instead of CLIP. Since the shapes are not compatible, we treat our latents as tokens and project the dimension to match the VideoLlama space. With this, we are able to integrate our latents with a powerful LLM, substituting our latents for the raw video input tokens. We can then perform any task that VideoLlama can, in particular question answering and captioning. We wish to emphasize that our latents are flexible – we can align well with any large off the shelf model, for any downstream task.

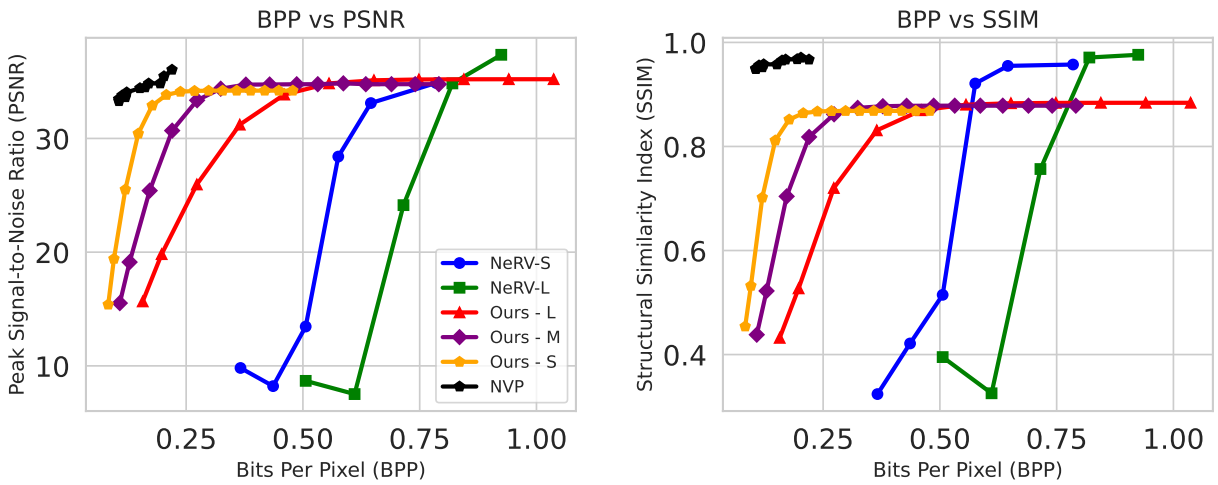



Figure 4.3: We plot the rate distortion curves on PSNR and SSIM to compare compression with other methods. We observe that our large model achieves comparable PSNR to the current SOTA [88]. Note that, while not plotted here, our decoding FPS is superior. Additional per-video results are available in the Supplementary.

## 4.4 Experiments

### 4.4.1 Video Compression

We perform comparative analysis for video compression on the standard Ultra Video Group (UVG) dataset [18]. This dataset comprises seven high-quality videos, each featuring diverse scenes shot at 120fps over a duration of five seconds. While most videos contain 600 frames, the 'shakendry' video is an exception with 300 frames, all at a resolution of

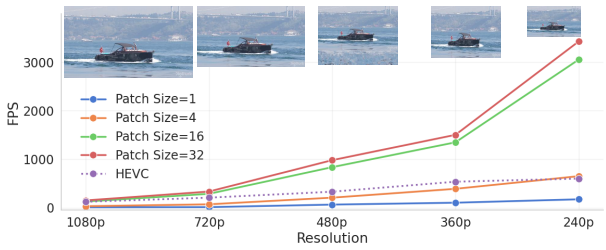



Figure 4.4: With the same model, we can perform inference at any resolution, with speeds competitive or beating HEVC. We show sample frames for each resolution.


Figure 4.5: We achieve high quality reconstruction and are able to reproduce even the finer details like water fountains and the hair on the horse.

1080x1920. To assess the visual quality, we use standard metrics such as Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity index (SSIM). We measure the storage efficiency of these methods using bits per pixel (BPP). As mentioned earlier, we use feedforward MLPs for both the base network $f_\theta$ and hypernetworks $h_l$. The base network consists of 6 layers with layer size of 512 and each hypernetwork that modulates a selected layer has one hidden layer of size 128 with tanh non-linearity, followed by the output layer. In the case where we use patch centroids as inputs, we add a convolutional layer followed by a pixel-shuffle [118] for upsampling.

We use hash-grids [93] for positional encoding due to their high quality reconstruction, although it should be noted we can use other schemes, such as Fourier features [19] to exchange some quality for faster training (see Appendix). We compare our method against NeRV [3] and NVP [88], with each of them encoding a video per model, and the results are presented in Figure 4.3. We observe that compression from our framework is comparable to baselines at similar bpp ranges, in addition to the other downstream benefits it offers.

Due to our architecture, we are also able to operate in a novel paradigm,"**any-resolution inference**." Without changing the network architecture at all, we can decode the video at arbitrary smaller resolutions, as well as at higher resolutions (super-resolution) by leveraging the continuous resolution property of our hash grids and MLPs. We show our FPS decoding at various resolutions in Figure 4.4, although it should be noted that HEVC, the standard codec we compare to, must encode separately for every resolution while we can store all in the same model. Figure 4.5 provides samples that showcase our method's fidelity.

Table 4.1: Interpolation Performance (PSNR), for different scale strides ($\alpha$).

| Dataset | $\alpha$ | NeRV | NIRVANA | NVP | Ours |
|---|---|---|---|---|---|
| Bunny | 2 | 15.92 | 19.14 | 20.10 | **33.17** |
| | 4 | 15.43 | 18.90 | 19.11 | **28.08** |
| | 8 | 13.68 | 18.67 | 18.08 | **25.88** |
| TaiChi | 2 | 16.91 | 18.19 | 19.33 | **35.13** |
| | 4 | 17.14 | 17.71 | 18.52 | **31.84** |
| | 8 | 15.72 | 16.21 | 17.7 | **27.72** |

Table 4.2: Reconstruction and retrieval ablations of CLIP on MSR-VTT.

| | Reconstruction | Retrieval (T2V) | | |
|---|---|---|---|---|
| CLIP $\lambda$ | PSNR | R@1 | R@5 | R@10 |
| 0.0 | 30.03 | 0.1 | 0.3 | 0.8 |
| 1e-3 | 29.83 | 28.4 | 50.8 | 60.6 |
| 1e-2 | 29.46 | 30.2 | 52.4 | 61.0 |
| 1e-1 | 28.93 | 29.7 | 51.5 | 61.8 |
| 1.0 | 28.61 | 30.2 | 51.4 | 61.3 |

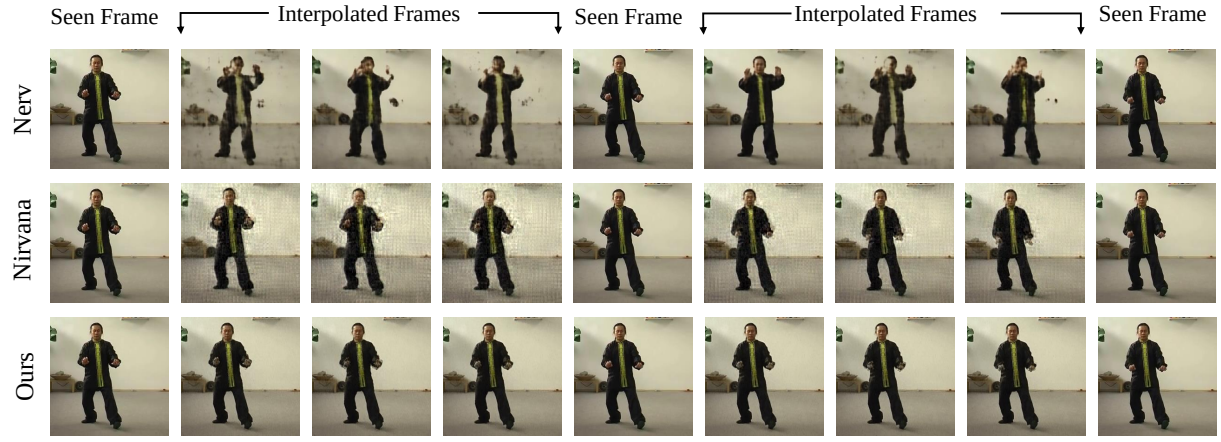



Figure 4.6: We compare interpolation with Latent-INR to NVP and NIRVANA. We find that our method has less artifacts and smoother motion in the interpolated frames.

### 4.4.2 Video Interpolation

In our framework, we can interpolate in the latent space to generate valid interpolated frame outputs. We conduct experiments on two datasets: the "big buck bunny sequence" and a selection of ten videos from the Taichi test set. Frames are held out at a scale stride $\alpha$ during encoding. During testing, we interpolate the resulting latents on the held out frames and evaluate their performance.

We use the same INR models utilized for compression as our baselines, with a re-

Table 4.3: Class and segment retrieval. Our method often exceeds CLIP performance.

| | | Class Level | | | Segment Level | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **Method** | R@1 | R@5 | R@10 | R@1 | R@5 | R@10 |
| COIN | CLIP | 31.60 | 44.70 | **50.70** | **6.60** | 13.10 | 16.50 |
| | Ours | **34.40** | **45.10** | 50.50 | 6.40 | **13.30** | **17.00** |
| HowTo100m* | CLIP | **31.58** | 36.84 | 47.37 | 21.13 | 37.32 | 40.85 |
| | Ours | 31.58 | **42.11** | **47.36** | **23.24** | **43.67** | **48.60** |

Table 4.4: Whole video retrieval. Our method matches CLIP performance.

| | | Text to Video | | | Video to Text | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **Method** | R@1 | R@5 | R@10 | R@1 | R@5 | R@10 |
| MSR-VTT | CLIP | 30.10 | 51.50 | **61.50** | 24.70 | 49.30 | **61.90** |
| | Ours | **30.20** | **52.40** | 61.10 | **25.40** | **49.90** | 61.70 |
| ActivityNet* | CLIP | 38.4 | **74.8** | **86.6** | **36.2** | **73.6** | **84.8** |
| | Ours | **38.5** | 73.9 | 86.4 | 36.1 | 73.5 | 84.7 |

duction in network layer size and modulating mask rank. While NeRV [3] and NVP [88] interpolate time positions used as input, NIRVANA interpolates the weights. In Table 4.1, we observe that while other INR methods fail to produce perceptual frames at scale of 2, our model can give reasonable interpolations even at a scale of 8. We confirm this qualitatively also, by inspecting interpolated frames such as those shown in Figure 4.6. Our outputs have noticeably fewer artifacts, and while imperfect, handle the motion better. Compared to other video INR methods, our approach of using learnt latents facilitates the model to have an internal representation of the video content.

### 4.4.3 Downstream Tasks

#### Retrieval

To showcase the flexibility of our latents, we align them with CLIP and evaluate their performance on standard retrieval tasks. We utilize the validation set of COIN dataset [119] and a subset of Howto100m dataset to evaluate performance. We first encode each video in our split using our Latent-INR framework with a loss that encourages the latents to be closer to the CLIP-Image embeddings of the frames, in addition to the standard reconstruction loss. We consider two distinct problems – retrieval of the correct class across all videos


Figure 4.7: Nearest Neighbours for segment-level matching of sample queries from COIN validation set. The green boxes denote the true positives and the red ones are false positives. We show the inner product similarity between the image and the corresponding query inside the green boxes at the bottom of each image.

and retrieval of the correct segment within a video. These two use cases cover both ends of the spectrum, from localizing an event in a given video to searching for similar events across videos. We utilize the standard recall at K, where we have selected $k \in [1, 5, 10]$ to evaluate the efficacy of our method. The results are presented in Table 4.3. We can see that our method matches CLIP in its retrieval performance and even exceeds it in some cases. The qualitative results are presented in Figure 4.7, where we visualize the top 5 nearest neighbours of the text query that map to trained latents across all videos. Further results can be found in the supplementary. We even find that our method can perform whole-video retrieval on MSR-VTT [120] and a custom 1,000 video sample from the ActivityNet Captions [121] 'val-1' split. We average-pool both our features and CLIP features (similar to [122]) and use CLIP features computed on video captions. In Table 4.4 we find that our retrieval is quite competitive to retrieval using the CLIP features themselves, showing


Figure 4.8: Latent-INR LLM. We show results for aligning our learned latents to a VideoLlama model, which allows for interactive chat. We show successes (left) and failures (right) for summarization (top) and question answering (bottom).

that the learnt latents have similarly good averaging and summarizing properties even over longer (180 seconds) videos, as well as alignment even to the paragraph-length captions used in ActivityNet.

**Video-based Chat**

We evaluate the performance of our trained latents, when aligned to intermediate VideoLlama features. This alignment enables access to the full scope of text chat with video understanding. We show a sample of such results, in the form of text and video prompts with text response, in Figure 4.8. These results show the LLM is able to understand video inputs when provided in the form of INR latents rather than raw video tokens. While not perfect, we infer the majority of the shortcomings of this system are primarily the fault of the LLM we align to. Furthermore, on the basis of our success in aligning with CLIP and now VideoLlama, we believe our latents can be aligned to any representation. So, for more powerful chat, one simply needs to align to a more powerful chatbot. We thus provide these results two purposes. First, we show our model's capability to power efficient open-ended

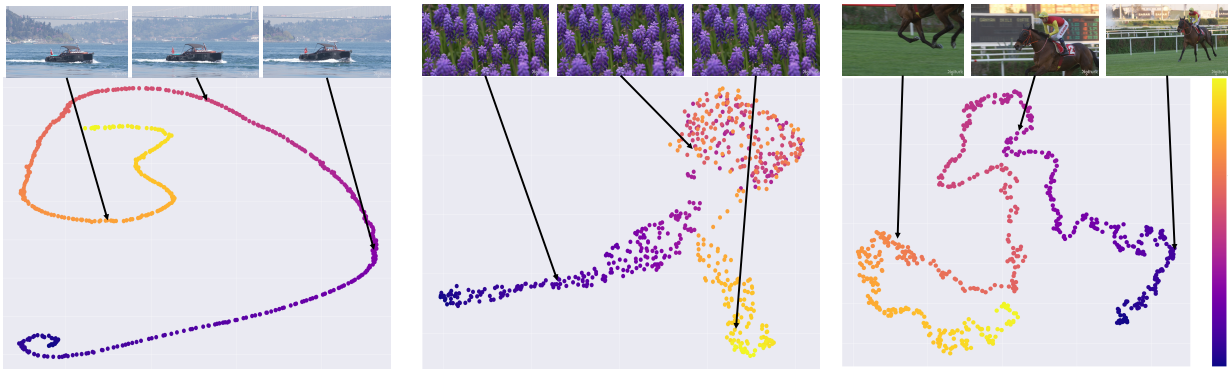


Figure 4.9: We visualize the trained latents $Z_t$ projected to 2D using UMAP. We show that the trained latents from our framework capture meaningful semantics of the underlying data. Latents for Bosphore (left), Honeybee (middle) and Jockey (right) from UVG dataset. Dark to Light color indicates frame numbers ranging from 0 to 600.

captioning and question answering, while still retaining reconstruction capabilities. Second, we point to the immense potential of our model (or a similar paradigm) to continue to be leveraged with such models as they expand in their size and performance.

### 4.4.4 Visualizing Trained Latents

The trained latents, representing the modulated frames, offer intriguing insights when visualized in a reduced dimensional space. Utilizing Uniform Manifold Approximation and Projection (UMAP)[123] we project the embeddings $Z_t$ into a 2D space, allowing for an intuitive interpretation of their relationships. In Figure 4.9, we plot the UMAP for three distinct videos from the UVG dataset: 'Bosphore,' 'Honeybee,' and 'Jockey,' each offering unique characteristics for examination.

'Bosphore', characterized by its slow-moving object and relatively static foreground, exhibits a smooth latent trajectory in the 2D space. This smoothness reflects the minimal variance in frame content, suggesting that our method effectively captures the subtle dy-

namics of the scene. In contrast, the 'Honeybee' video, with its repetitive frames, results in latents that cluster tightly together, signifying our model's ability to recognize and encode repetitive patterns efficiently. The most dynamic of the three, 'Jockey', presents a more complex scenario with rapid changes in both the foreground and background. Here, the latents form clusters around similar scenes, yet maintain a discernible trajectory through the 2D space. These visualizations illustrate the semantic richness embedded within the latents obtained from our framework even when trained only for compression.

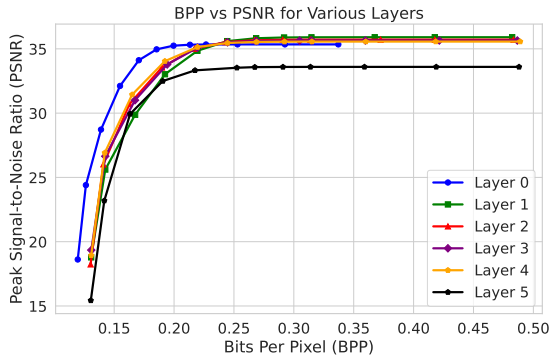
## 4.5 Ablation Studies

**CLIP $\lambda$.** We investigate the impact of the large model alignment weighting term on both reconstruction and retrieval for MSR-VTT. In Table 4.2, we find that PSNR decreases slightly as $\lambda$ increases. However, the retrieval performance seems to saturate at $\lambda = 0.01$. So, we suggest not tuning the $\lambda$ too high for any application, given the diminishing returns.

**Layer Modulations.** In our approach, we have separate hypernetworks that modulated the selected layers. To evalute the importance of each, we design an experiment where they are modulated in isolation. We use the same setup as the compression experiments with the modulating mask rank fixed at 20 for all models. In Figure 4.10, we can clearly see that the first few layers have a significant impact on the encoding performance. This matches the observations from [62] about the out sized impact of first few layers while modulating INRs.

**Patch Size.** Scaling to higher-resolution videos can be memory-intensive. This is particularly true when employing memory-demanding positional encoding schemes such as

Effect of layer modulations in the hypernetwork




Effect of patch size on reconstruction quality (PSNR)

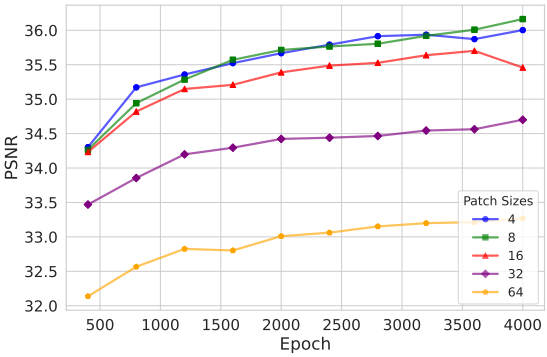



Figure 4.10: Ablations to study the effect of layer modulations in the hypernetwork (top) and the effect of patch size on reconstruction quality (PSNR) (bottom). The layer modulations study shows how different types of modulations affect the model's performance. The patch size study demonstrates the relationship between patch size and reconstruction quality, measured in PSNR (Peak Signal-to-Noise Ratio).

hash-grids [93]. To investigate this aspect further, we experiment with models that process centroids of fixed-size patches, directly predicting the corresponding frame patches, to save memory. From Figure 4.10, performance is consistent for smaller patch sizes, but drops off sharply for higher patch sizes.

## 4.6 Discussion

**Limitations.** Our latents are somewhat restricted by the quality of the embeddings they are aligned to. Additionally, more work is still required to match standard codecs in terms

of storage and encoding time, in spite of impressive gains in terms of quality and decoding speed. Future work could both improve the compression, and leverage more powerful vision models.

**Broader Impacts.** Our method for simultaneously compressing and learning useful features for recognition could reduce the need to decode videos for these tasks and thus save computational resources, cutting costs and helping the environment. However, work that advances performance for compression and recogntion also has applications in surveillance and warfare.

In this work, we propose a new framework, Latent-INR, where we decouple the temporal aspect from the spatial into a dictionary of learnable latents. These auto-decoder based learnable latents modulate the layers of the base INR network via low-rank modulation using hypernetworks. Latent-INR is not only well-suited to video compression, but the resulting latents learn an internal representation of the data they encode that lends itself to SOTA interpolation for video INRs. Additionally, we also augment these latents by training them to be aligned with CLIP and VideoLlama, which allows us to bring the power of foundational models to compressed representations, and perform retrieval and chat-based applications like captioning and question answering. Our work thus opens up new possibilities of research in the implicit neural space where downstream tasks can be performed by these model without the need for decoding.

## 4.7 Appendix

### 4.7.1 Network Architecture

**Base Network:** We use an MLP with 10 layers, width of 512 and ReLU non-linearity as our base network $f_\theta$.

**Hypernetwork:** All hypernetworks $h^l$ used to modulate a layer $l$ of the base network have 3 layers with a hidden dimension of 512 and tanh as non-linearity. Unless specified, we only modulate the first hidden layer of the base network.

**Latents:** Each latent $Z_t$ corresponding to a frame has a dimension of 512 and is initialized to be standard Gaussian before training. We set our learning rate as 5e-4 and used the standard Adam optimizer without any weight decay.

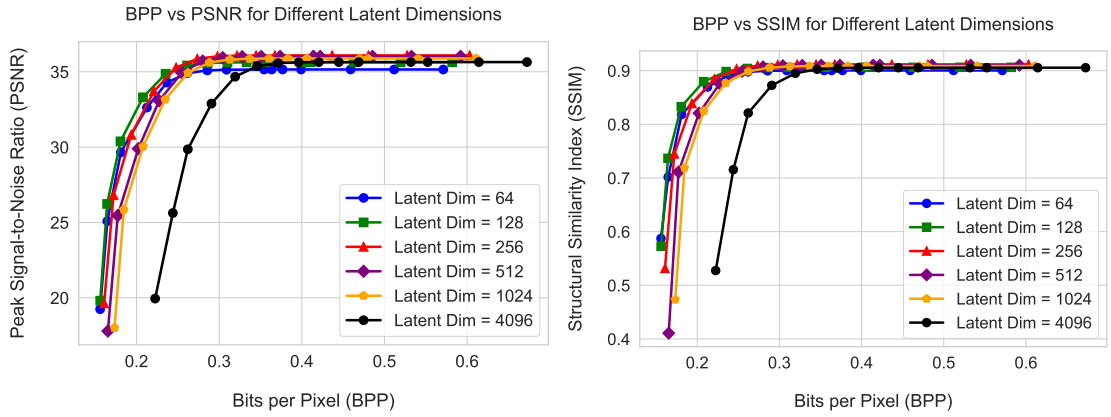
### 4.7.2 Compression

#### 4.7.2.1 Fourier Features

We use the multiresolution hash grid for positional encoding in all our models. In table 4.5 we show results for full coordinate resolution using fourier features for positional encoding. Due to lack of a hash grid, the resulting models train upto 30% faster, but at the cost of inferior reconstruction.

#### 4.7.2.2 Quantization

Instead of quantizing all components equally, we notice that retaining the latents and the base network at full precision provides better reconstruction at negligible additional




(a) (b)

Figure 4.11: Effect of varying latent dimension across different bitrates. (a) PSNR vs. bitrate. (b) SSIM vs. bitrate.

storage.

#### 4.7.2.3 Effect of latent dimension

To study the effect of latent dimension on compression, we train models by varying it and encode the "bosphore" video from UVG dataset. The results are presented in Figure 4.11. We notice that there is positive gains till dimension 512 and diminishing returns thereafter. Hence we choose that as our default latent size in all our experiments.

### 4.7.3 Video Retrieval

We perform two retrieval tasks on the COIN dataset[119] - *class-level*, and *segment-level*. In both settings, we use the standard val set as the database. For *class-level*, we use the distinct video-level task names in COIN as our query set. For *segment-level*, we use the set of distinct clip-level captions in COIN as our query set. We get the CLIP ViT-B/32 text embeddings of each of these captions, and these become our query vectors. For database vectors, we use the per-frame learned latents for each video in the database.


Figure 4.12: Nearest Neighbours for segment-level matching of sample queries from COIN validation set. The green boxes denote the true positives and the red ones are false pos- itives. We show the inner product similarity between the image and the corresponding query inside the green boxes at the bottom of each image

For comparison with CLIP, we replace these database vectors with the CLIP ViT-B/32 image embeddings for each frame. For *class-level* retrieval, we consider a result frame a positive match if it belongs to a video with the same class label as the queried caption. On the other hand, for *segment-level* retrieval, we consider a result frame a positive match only if it belongs to a segment with the same caption as the query. Further, this search is done over all videos. We use FAISS[124] as our retrieval implementation and use *cosine similarity* as the distance metric.

We perform whole-level video retrieval as described in the main paper. For text, we use CLIP to compute a feature for the paragraph caption. For the video, we compute a per-frame feature for CLIP, or use the learnt latents from Latent-INR. For a single video feature, we then average these per-frame features. We normalize all features, and perform retrieval by finding the closest embeddings using dot product similarity. Both text-to-video and video-to-text are performed in the same manner, the only difference being which features are used as query and key.

Table 4.5: Fourier Feature Models

| Method | PSNR | BPP |
|---|---|---|
| Ours- Fourier - $S$ | 31.99 | 0.31 |
| Ours- Fourier - $M$ | 33.69 | 0.62 |
| Ours- Fourier - $L$ | 33.19 | 0.84 |

Fig.4.12, shows the retrieval results on the COIN data in the *segment-level* setting. It can be seen that a majority of failure cases could be attributed to visual similarity across different tasks when seen at an individual frame level.

### 4.7.4 Video Chat

We interface our latents with learned features from Video-Llama [91] to enable interactive chat with the compressed videos. In [91], the $N$ video frames are passed through a ViT based visual encoder to extract features of size $k \times d$ per frame. These are then passed through a Query Former [125] to obtain a unified video representation of size $k_v \times d_v$. This tensor is then passed to a trainable MLP layer before aligning with an LLM of our choice (LLama-2 [126] in our models).

We align our latents $Z$ with these per-video features of size $k_v \times d_v$ using a linear projection layer which is trained end to end. The loss function is slightly modified to incorporate a cosine similarity loss between the terms.

$$L = L_{\text{MSE}} + \lambda \cdot L_{cos}(F_t, F_t^{\text{V-LLM}}) \tag{4.7}$$

where $F_t$ is the predicted feature and $F_t^{\text{V-LLM}}$ is the corresponding Video-LLama extracted features. We show additional results of the interactive chat in Figure 4.13.


Figure 4.13: Additional results for Latent-INR interface with Video-LLM.

# Chapter 5: LTH for object recognition

In this chapter [1] we explore the role of sparsity in vision systems. The Lottery ticket hypothesis (LTH) states that we can find sparse sub-networks within a fully trained network that can either match or sometimes even exceed the performance of the original network. We explore the effects of LTH on recognition systems such as object detection, instance segmentation, and keypoint detection. Since sparse networks also lead to model compression and faster inference, this chapter lays the foundation for a series of works which then go on to explore the transformation from data compression to model compression.

## 5.1 Introduction

Recognition tasks, such as object detection, instance segmentation, and keypoint estimation, have emerged as canonical tasks in visual recognition because of their intuitive appeal and pertinence in a wide variety of real-world problems. The modus operandi followed in nearly all state-of-the-art visual recognition methods is the following: (i) Pre-train a large neural network on a very large and diverse image classification dataset, (ii) Append a small task-specific network to the pre-trained model and fine-tune the weights jointly on a much smaller dataset for the task. The introduction of ResNets by He *et al.* [127]

[1] Work done jointly with Sharath Girish. I was responsible for implementing LTH on detection, segmentation and ablations. Sharath was responsible for the keypoint detection task and ablations.

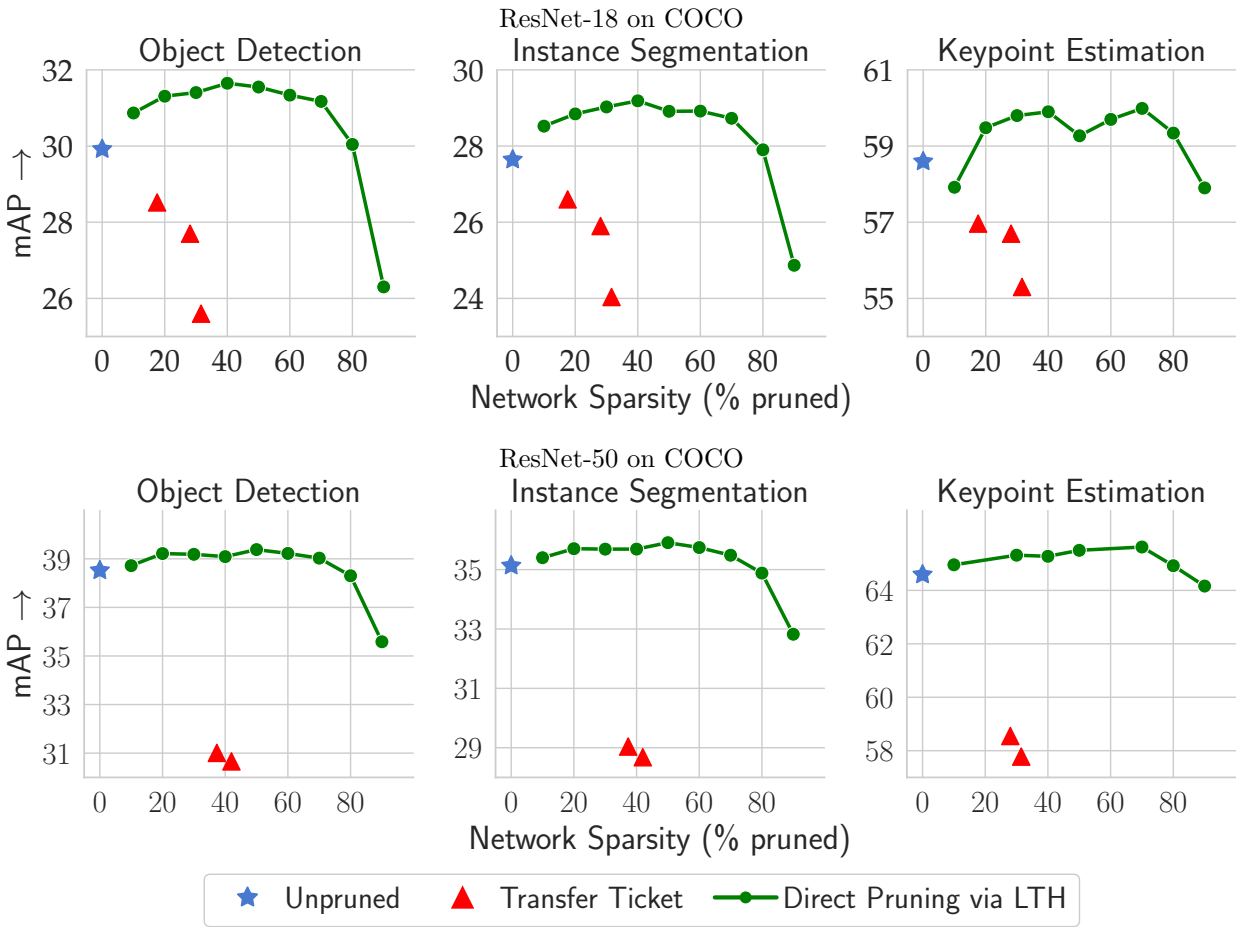



Figure 5.1: Performance of lottery tickets discovered using direct pruning for various object recognition tasks. Here we have used a Mask R-CNN model with ResNet-18 backbone (top) and ResNet-50 backbone (bottom) to train models for object detection, segmentation and human keypoint estimation on the COCO dataset. We show the performance of the baseline dense network, the sparse subnetwork obtained by transferring ImageNet pre-trained "universal" lottery tickets, as well as the subnetwork obtained by task-specific pruning. Task-specific pruning outperforms the universal tickets by a wide margin. For each of the tasks, we can obtain the same performance as the original dense networks with only 20% of the weights.

made the training of very deep networks possible, helping in scaling up model capacity, both in terms of depth and width, and became a well-established instrument for improving the performance of deep learning models even with smaller datasets [128]. As a result, the past few years have seen increasingly large neural network architectures [129, 130, 131, 132], with sizes often exceeding the memory limits of a single hardware accelerator. In recent years, efforts towards reducing the memory and computation footprint of deep networks

have followed three seemingly parallel tracks with common objectives: weight quantization, sparsity via regularization, and network pruning; Weight Quantization [133, 134, 135, 136, 137] methods either replace weights of a trained neural network with lower precision or arithmetic operations with bit-wise operations to reduce the memory up to an order of magnitude. Regularization approaches, such as dropout [138, 139] or LASSO [140], attempt to discourage an over-parameterized network from relying on a large number of features and encourage learning a sparse and robust predictor. Both quantization and regularization approaches are effective in reducing the number of weights in a network or the memory footprint, but usually at the cost of increased error rates [137, 141]. In comparison, pruning approaches [142, 143] disentangle the learning task from pruning by alternating between weight optimization and weight deletion. The recently proposed Lottery Ticket Hypothesis · (LTH) [144] falls in this category.

According to LTH, an over-parameterized network contains sparse sub-networks which not only match but sometimes even exceed the performance of the original network, all by virtue of a "lucky" random initialization before training. The original paper was followed up with tips and tricks to train large-scale models under the same paradigm [73]. Since then, there has been a large, growing body of literature exploring its nuances. Although some of these recent works have tried to answer the question – how well do the tickets transfer across domains [145, 146], when it comes to vision tasks – the buck stops at image classification.

In this work, we aim to extend and explore the analysis of lottery tickets to fundamental visual recognition tasks of object detection, instance segmentation, and keypoint detection. Popular methods for such recognition tasks use a two-stage detection pipeline,

with a supervised pre-trained convolutional neural network (ConvNet) backbone, a region proposal network (RPN), and one or more region-wise task-specific neural network branches. Loosely speaking, a ConvNet backbone is the most computationally intensive part of the architecture, and pre-training is the most time-consuming part. Therefore, as part of this study, we explore the following questions: (a) Are there *universal* sub-networks within the ConvNet backbone that can be transferred to the downstream object recognition tasks? (b) Can we train sparser and more accurate sub-networks for each of the downstream tasks? And, (c) How does the behavior or properties of these sub-networks change with respect to the corresponding dense network? We investigate these questions under the dominant settings used in object recognition frameworks. Specifically, we use ImageNet [147] pre-trained ResNet-18 and ResNet-50 [127] backbones, Faster R-CNN [148] and Mask R-CNN [149] modules for object recognition on Pascal VOC [150] and COCO [151] datasets. Our contributions are as follows:

- We show that tickets obtained from ImageNet training don't transfer to object recognition in case of COCO, *i.e.*, there are no *universal* tickets in pre-trained ImageNet models that can be used for downstream recognition tasks without a drop in performance. This is in contrast with previous works related to ticket transfer in vision models [145, 146]. In case of smaller datasets such as Pascal VOC, we are able to find winning tickets from ImageNet pre-training with upto 40% sparsity.

- With direct pruning, we can find "task-specific" tickets with up to 80% sparsity for each of the datasets and backbones. We also investigate the efficacy of methods introduced by [144, 145, 152, 153] such as iterative magnitude pruning, late resetting,

early bird training, and layerwise pruning in the context of object recognition.

- Finally we analyse the behavior of tickets obtained for object recognition tasks, with respect to various task attributes such as object size, frequency, and difficulty of detection, to make some expected (and some surprising) observations.

## 5.2 Related Work

**Model Compression:** Ever since deep neural networks started gaining traction in real-world applications, there have been serious attempts made to reduce their parameters, intending to attain lower memory footprints [133, 134, 135, 136, 137, 154], higher inference speeds [155, 156, 157] and potentially better generalization [158]. Amongst the various proposed techniques, model pruning approaches are predominant mainly due to their simplicity and effectiveness. One line of methods follow an unstructured process where insignificant weights are set to zero and are frozen for the rest of the training. The significance of weights are quantified either by magnitude [143] or gradients during training time [159]. In structured pruning methods, relationships between pruned weights are taken into consideration, leading to pruning them in groups. Methods like [160] utilize Group Lasso regularization to prune redundant filter weights to enable structural sparsity, [161] uses explicit $L0$ regularization to make weights within structures have exact zero values, and network slimming [162] learns an efficient network by modelling the scaling factor of batch normalization layer.

**The Lottery Ticket Hypothesis:** The introduction of Lottery Ticket Hypothesis by [73] opened a pandora's box of immense possibilities in the field of pruning and sparse models.

The original paper was followed by [37] where the authors introduce the concept of "late resetting" which enabled the application of the hypothesis to larger and deeper models. [163] followed up by proposing an extensive, in-depth analysis where they show that the resetting of the weights need not be to the exact initialization, but just need to the initial signs. [152] probes the aspect of resetting further to show that the reason why LTH works is because of its ability to make the subnetwork stable to SGD noise. [164] probed an orthogonal question about the number of possible tickets from a network. They showed that a single initialization had multiple winning tickets with low overlap and empirically conclude that there exists an entire "distribution" of winning tickets. Complementary to LTH[73], [159] and [165] offer algorithms that can pick the winning ticket without the need for training. But they do not match the performance of the original procedure. The problem of longer training using LTH was effectively tackled by [166] which introduced the concept of "early bird tickets" where the authors show that the winning tickets and their masks are obtained in the first few epochs of training, foregoing the need to train the original initialization till convergence. The intriguing properties of LTH led to a glut of works which investigated its eclectic aspects. [145] scrutinize the generalization properties of winning tickets and offer empirical evidence that winning tickets can be transferred across datasets and optimizers, in the realm of image classification. [146] then proposed a variation of the theory titled "transfer ticket hypothesis" where they investigate the effectiveness of transferring a mask generated from source dataset to a target dataset. The work of [167] briefly analyzes LTH on single stage detectors such as YOLOv3 [168] and achieves 90% winning tickets, while maintaining the mAP on the Pascal VOC 2007 dataset. However, as they evaluate on light-weight and fast detectors, their mAP ($\sim 56$) is much lower compared to networks like Faster

R-CNN [148] which reach mAP of $\sim 69$ with just a ResNet-18 backbone. Their work is also limited to object detection and does not provide a detailed analysis of LTH for the task. The idea for transferring subnetworks obtained from ImageNet to object detection tasks was concurrently discussed by [169]. For small datasets such as Pascal VOC, [169] observes that ImageNet tickets transfer for detection and segmentation tasks. However, we extend the analysis to the larger COCO dataset and show that this observation doesn't hold. We further build upon these results, to test out the generalization and transfer capabilities of winning lottery tickets across different object recognition datasets and tasks in computer vision.

## 5.3 Background: Lottery Ticket Hypothesis

LTH states that dense randomly-initialized neural networks contain sparse sub-networks which can be trained in isolation and can match the test accuracy of the original network. These sub-networks are called winning tickets and can be identified using an algorithm called Iterative Magnitude Pruning (IMP). Suppose the number of iterations for pruning is $T$ and we wish to prune $p\%$ of the network weights. The weights/parameters are represented by $w \in \mathbb{R}^n$ and the pruning mask by $m \in \{0,1\}^n$ where $n$ is the total number of weights in the network. The complete algorithm is presented in 2. This pruning method can be one-shot when it proceeds for only a single iteration or it can proceed for multiple iterations, $k$, pruning $p^{\frac{1}{k}}\%$ each round. The authors also use other techniques such as learning rate warmup and show that finding winning tickets is sensitive to the learning rate.

**Algorithm 2:** Iterative Pruning for LTH

Randomly initialize network $f$ with initial weights $w_0$, mask $m_0 = \mathbb{1}$, prune target percentage $p$, and $T$ pruning rounds to achieve it.
**while** $i < T$ **do**
    Train network for N iterations $f(x; m_i \odot w_0) \rightarrow f(x; m_i \odot w_i)$
    Prune bottom $p^{\frac{1}{k}}\%$ of $m_i \odot w_i$ and update $m_i$
    Reset to initial weights $w_0$
    $i \leftarrow i + 1$ `// next round`
**end**

While this method obtains winning tickets for smaller datasets, like MNIST [170], CIFAR10 [171], they fail to generalize to deeper networks, such as ResNets, and larger vision benchmarks, such as ImageNet [147]. [37] shows that IMP fails when resetting to the original initialization. They claim that resetting instead to the network weights after a few iterations of training provides greater stability and enables them to find winning tickets in these larger networks. They show that rewinding/late resetting to $3 - 7\%$ into training yields subnetworks which are 70% smaller in the case of ResNet-50, without any drop in accuracy.

## 5.4 Experiments

In this section, we extend the Lottery Ticket Hypothesis to several object recognition tasks, such as Object Detection, Instance Segmentation, and Keypoint Detection and study their effects. Then we analyze the transfer of winning tickets from ImageNet to these tasks, and the effect of direct pruning on these tasks. Finally, we analyze the properties of these winning tickets, ablate the effect of various hyperparameters, and study the effect of task-specific properties on the winning tickets.

### 5.4.1 Setup

We evaluate LTH primarily on the 2 datasets - Pascal VOC 2007 and COCO. We deal with the 3 tasks of object detection, instance segmentation, and keypoint detection for COCO and only object detection for VOC. We use Mask-RCNN for object detection and segmentation for COCO, Keypoint-RCNN for keypoint detection, and Faster-RCNN for object detection on VOC. We compare the results for ResNet-18 and ResNet-50 backbones. Note that while we use the term mean Average Precision (mAP) as a performance metric for all the tasks and datasets, the actual calculation of mAP is done using code provided in the respective datasets (and cannot be compared across the tasks).

### 5.4.2 Transfer of ImageNet Tickets

Many object recognition tasks utilize pre-trained networks whose backbones are trained on the ImageNet dataset. This is because ImageNet features and weights have shown the ability [149] to generalize well to several downstream vision tasks. A plethora of works exists which perform LTH for the ImageNet classification task and obtain winning tickets. Therefore, tickets for standard convolutional backbones, such as ResNets, are readily available. This raises the pertinent question of whether pruned ImageNet trained models transfer directly to object recognition tasks. In order to answer this question, we transfer the pruned model to the backbone of the RCNN-based network and fine-tune the full network while ensuring the pruned weights in the backbone remain as zeros.

We perform experiments for the two architectures: ResNet-18 and ResNet-50, where we obtain $10\%, 20\%$, and $50\%$ tickets on ImageNet by following the approach of [37], and

Table 5.1: Performance on the COCO dataset for ImageNet transferred tickets for ResNet-18 backbone at varying sparsity. The results for VOC are averaged over 5 runs with the standard deviation in parentheses.

| Prune (%) | COCO Detection | | | COCO segmentation | | | COCO Keypoint | | | VOC Detection | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | sparsity | mAP | AP50 | sparsity | mAP | AP50 | sparsity | mAP | AP50 | sparsity | mAP |
| 90 | 31.61% | 25.59 | 43.69 | 31.61% | 24.03 | 40.89 | 21.47% | 55.30 | 79.30 | 79.49% | 63.91 (±0.41) |
| 80 | 28.10% | 27.70 | 46.50 | 28.10% | 25.90 | 43.70 | 19.09% | 56.70 | 81.10 | 70.66% | 65.82 (±0.23) |
| 50 | 17.57% | 28.52 | 47.54 | 17.57% | 26.60 | 44.66 | 11.94% | 56.96 | 80.83 | 44.16% | 68.06 (±0.11) |
| 0 | 0% | 29.91 | 49.05 | 0% | 27.64 | 46.00 | 0% | 58.59 | 82.04 | 0% | 68.53 (±0.29) |

Table 5.2: Performance on the COCO dataset for ImageNet transferred tickets for ResNet-50 backbone at various levels of pruning. The results for VOC are averaged over 5 runs with the standard deviation in parentheses. We obtain higher levels of sparsity compared to ResNet-18 transferred tickets which can be expected as it has fewer redundant parameters. Additionally, tickets for VOC have much higher sparsity with no drop in mAP compared to unpruned model.

| Prune (%) | COCO Detection | | | COCO segmentation | | | COCO Keypoint | | | VOC Detection | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | sparsity | mAP | AP50 | sparsity | mAP | AP50 | sparsity | mAP | AP50 | sparsity | mAP |
| 90 | 41.99% | 30.66 | 50.75 | 41.99% | 28.68 | 47.76 | 31.49% | 57.78 | 82.25 | 65.37% | 71.20 (±0.21) |
| 80 | 37.33% | 31.01 | 50.98 | 37.33% | 29.04 | 47.90 | 28% | 58.55 | 83.06 | 58.11% | 71.08 (±0.20) |
| 0 | 0% | 38.5 | 59.29 | 0% | 35.13 | 56.39 | 0% | 64.59 | 86.48 | 0% | 71.21 (±0.32) |

then transfer the model to the backbones of the three R-CNN based networks. All models were trained on COCO, encompassing the three recognition tasks and the results are summarized in Tables 5.1, 5.2. Additionally, we also perform similar experiments on the smaller Pascal VOC 2007 dataset for object detection to verify whether ImageNet tickets transfer without a significant drop in mAP. The results are shown in the last column of Tables 5.1, 5.2. Note that pruning percentage of the ImageNet ticket is not equal to the actual network sparsity of the various networks as only the backbone of the networks are transferred and they make up a fraction of the total weights. We see that ImageNet tickets transferred to COCO show a noticeable drop in mAP even with low levels of sparsity. We also note that another drawback of training transferred tickets on COCO is that they require careful tuning of learning rate and batch size for different tasks. On the other hand, for smaller datasets such as Pascal VOC, winning tickets are easily obtained at higher levels of sparsity which is $\sim 45\%$ for ResNet-18 and $\sim 65\%$ for ResNet-50. The larger networks can be pruned to a greater extent for both datasets. ImageNet transferred tickets offer very little sparsity as the backbone of the networks usually do not make up most of the weights. For example, the ResNet-18 based Mask-RCNN for COCO Detection and Segmentation has only $44\%$ of the parameters in the backbone, and hence, the overall network sparsity reaches $31\%$ when pruning $90\%$ of the backbone weights, as shown in Table 5.1. The rest of the weights are usually from the fully-connected layers of the network. It is therefore imperative to prune layers in addition to the backbone to increase network sparsity without decrease in mAP. As a consequence, we look into directly pruning the full network using LTH.

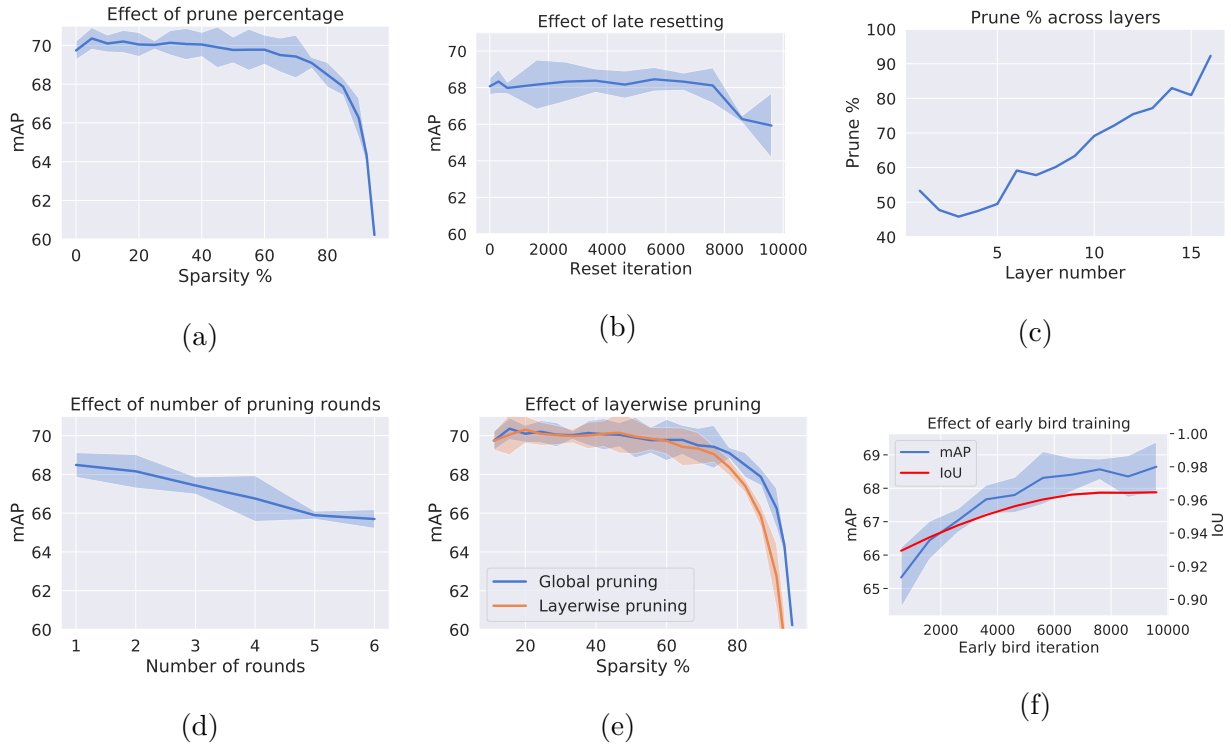



Figure 5.2: Effect of varying different hyperparameters for pruning Faster RCNN with ResNet-18 backbone on the Pascal VOC 2007 [150] dataset. All solid lines reported are the values averaged over 5 runs and the error bands are within 3 times the standard deviation.

### 5.4.3 Direct Pruning for Downstream Task

In this section, we analyze the effect of various hyperparameters and pruning strategies for detection networks in order to obtain winning tickets. We primarily use the ResNet-18 backbone for Faster-RCNN trained on VOC for all our experiments in this section, unless mentioned otherwise. Even though the ResNet-18 backbone is smaller than other backbone networks such as ResNet-50, we find that similar conclusions hold for the larger networks as well. The Faster RCNN network consists of parameters which we group into 4 main modules: Base Convolutions, Classification Network Convolutions (Top), Region Proposal Network (RPN), Classification network box and classification fully connected heads (Box and Cls Head). We provide a detailed analysis of pruning these 4 groups and their effect

on winning tickets. Additionally, we also analyze different pruning strategies and the role played by hyperparameters.

**Varying Pruning Percentage:** We evaluate the network performance at varying levels of sparsity. We prune different percentages of parameters in the Base and Top modules which include 88% of the total network parameters. The results are plotted in Fig.5.2**??**. We achieve performance within one standard deviation of the baseline, with 70% sparsity. Our models outperform the baseline mean, thereby proving that we can indeed obtain high performance winning tickets at much higher levels of sparsity for detection. Additionally, we see that at any given sparsity, direct pruning yields much better results compared to ImageNet transferred tickets. We also show that these observations pan other vision tasks by obtaining winning tickets for Mask-RCNN and Keypoint-RCNN with both ResNet-18 and ResNet-50 backbones on the COCO dataset. We additionally prune FC layers in these set of experiments in order to achieve desired sparsity levels as they take up $\sim 50\%$ of the total weights. The results for ResNet-18 are shown in Fig. 5.1. We obtain winning tickets with 80% sparsity on all the three tasks while outperforming the unpruned network for lower levels of sparsity. Additionally, we consistently outperform the different ImageNet transferred tickets (50%, 80%, 90%) by a large margin supporting our claim that direct training of tickets on downstream tasks yield better results than ImageNet tickets.

**Effect of Early/Late Resetting:** [37] states that resetting the network to a few iterations through training instead of the initialization stabilizes the winning ticket training. We evaluate whether this holds true for detection tasks as well. We show the performance of winning tickets as a function of resetting at various stages of training in Fig. 5.2**??** and

observe that resetting during the earlier or even mid stages of training does not have a very strong effect on the final mAP. This is likely because the backbones of detection networks are initialized with ImageNet weights and are not random as is the case with other papers dealing with LTH in the classification setting. Therefore, the weights are more stable and late resetting is not necessary. We also additionally analyze effects of resetting towards the end of training and notice that there is a sharp drop in the performance after $8k$ iterations. This is because the learning rate is decayed at this stage of training and the parameters change significantly right after. A similar case holds when we perform learning rate warmup but do late resetting before the learning rate is fully warmed up. The performance drops significantly as the learning rate keeps fluctuating showing that late resetting is quite sensitive to learning rate.

**Pruning different Faster-RCNN modules:** We prune 20% of the parameters of the various modules within the Faster-RCNN network and analyze their effects on the mAP. We also try different combinations of pruning with the modules and report the results in Table 5.3. Pruning the Box and Classification head (which takes up only 65% weights) outperforms the baseline case of no pruning, but does not always improve performance when other modules are being pruned. Additionally, pruning the $RPN$ module increases the performance slightly even though it comprises of only 10% of the network weights. Next, pruning the Base module and/or the Top module of the backbone leads to a drop in performance, which is expected as they consist of 22% and 66% of the weights respectively. Pruning the Base alone, excluding the Top, performs nearly as well as the baseline, while including the Top yields a lower mAP.

Table 5.3: Performance on Pascal VOC by pruning different modules of a ResNet-18 Faster-RCNN network. The results are averaged over 5 runs with the standard deviation in parantheses. ✓ represents the module being pruned, while Param % represents the percentage of parameters occupied by the modules being pruned.

| Base | Top | RPN | Box, Cls Head | Param % | Network Sparsity | mAP |
|---|---|---|---|---|---|---|
| - | - | - | - | 0 | 0% | 69.74 (±0.16) |
| - | - | - | ✓ | 0.65 | 0.52% | 70.30 (±0.14) |
| - | - | ✓ | - | 9.71 | 7.77% | 70.02 (±0.19) |
| - | - | ✓ | ✓ | 10.36 | 8.29% | 70.08 (±0.10) |
| ✓ | - | - | - | 21.93 | 17.55% | 69.32 (±0.07) |
| ✓ | - | - | ✓ | 22.59 | 18.07% | 69.60 (±0.19) |
| ✓ | - | ✓ | - | 31.64 | 25.31% | 69.39 (±0.25) |
| ✓ | - | ✓ | ✓ | 32.29 | 25.83% | 69.47 (±0.15) |
| - | ✓ | - | - | 66.39 | 53.11% | 69.02 (±0.19) |
| - | ✓ | - | ✓ | 67.04 | 53.63% | 68.74 (±0.21) |
| - | ✓ | ✓ | - | 76.09 | 60.88% | 68.88 (±0.25) |
| - | ✓ | ✓ | ✓ | 76.75 | 61.40% | 68.93 (±0.26) |
| ✓ | ✓ | - | - | 88.32 | 70.66% | 68.45 (±0.21) |
| ✓ | ✓ | - | ✓ | 88.97 | 71.18% | 68.54 (±0.23) |
| ✓ | ✓ | ✓ | - | 98.03 | 78.42% | 68.51 (±0.23) |
| ✓ | ✓ | ✓ | ✓ | 98.68 | 78.94% | 68.47 (±0.10) |

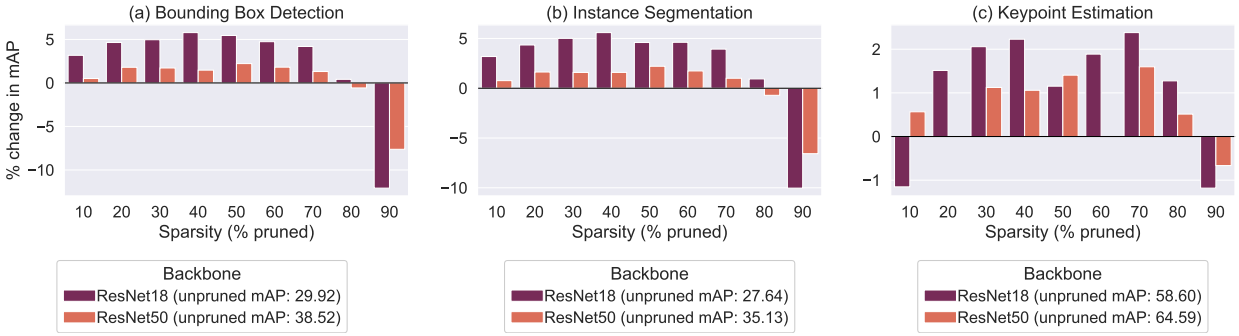



Figure 5.3: ResNet-18 *vs.* ResNet-50. We analyse change in mAP by using LTH on Mask R-CNN with different backbones.

**Performance of Early-bird tickets:** [166] showed that tickets can be found at early stages of training. We visualize this by obtaining masks at various stages in training and evaluating their performance. We also plot each masks' Intersection over Union (IoU) with the default mask obtained at the end of training. This IoU shows the overlap in the parameters being pruned. The results are visualized in Fig. 5.2**??**. We see that within 50% of network training we find tickets whose performance is within a standard deviation of the performance of the default ticket (obtained at the end of training). This is because the IoU becomes more or less stable at around 0.96 during the middle stages of training and the mask is unchanged as training advances. This allows us to cut down on the number of training iterations significantly with very little cost to the network performance.

**Effect of number of rounds of pruning:** [144] states that iterative pruning performs better than one-shot pruning on the classification task with small datasets and networks. We show that this does not necessarily hold true for detection and larger backbones. We plot the network's performance against various rounds of pruning and observe that one-shot pruning outperforms iterative methods in Fig. **??**.

**Layer-wise *vs.* global pruning:** [37] performs global pruning for larger datasets and

networks and claims that pruning at the same rate in lower layers as compared to higher layers, is detrimental to the network performance. We evaluate the two methods of pruning on the detection task and show the results in Fig. 5.2??. Additionally, for global pruning, we plot the percentage of parameters pruned in each layer of the backbone network in Fig. ??. Layer-wise pruning does as good as global pruning for lower levels of sparsity. However, there is a noticeable performance gap for sparsity levels above 60%. This is because layer-wise pruning forces lower layers with very few parameters to have high sparsity percentages. But as per Fig. ??, for global pruning, we see that lower layers are pruned less as they are crucial to both the RPN and Classification stages of the network.

### 5.4.4 Properties of Winning Tickets

In Section 5.4.3, we showed that we can discover sparser networks within our two-stage Mask-RCNN detector if we directly prune on the task itself. We build upon those results to further probe the properties of winning tickets.

**Effect of backbone architecture:** In Fig. 5.3, we show how winning tickets behave for 2 different backbones, ResNet-18 and ResNet-50, at different sparsity levels (50%, 80%, 90%). We make two observations: First, the breaking point for both networks is $\sim$ 80% sparsity. However, performance of ResNet-18 drops more sharply than ResNet-50 afterwards. This is intuitive since ResNet-18 has fewer redundant parameters and over-pruning leads to drop in the performance. Second, as we gradually increase the sparsity of the networks, mAP increases for all tasks in case of both networks. However, gains for ResNet-18 models are consistently more than ResNet-50.

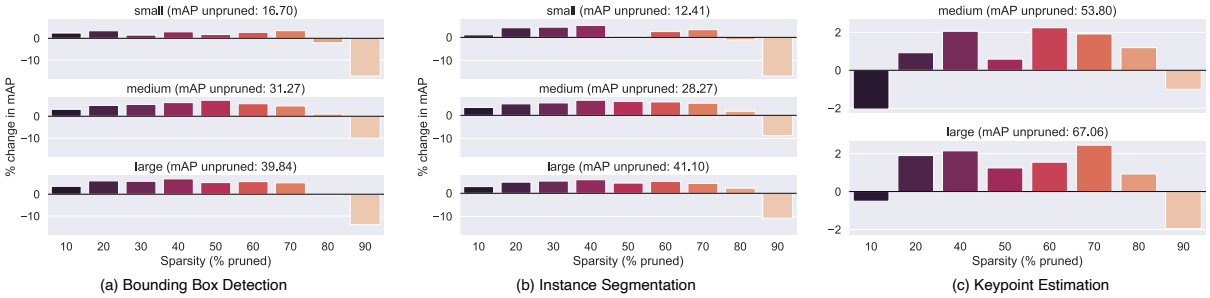



Figure 5.4: Comparison of Mean Average Precision (mAP) of pruned model for different object sizes in case of Object Detection, Instance Segmentation, Keypoint Estimation. x-axis shows the sparsity of the subnetwork (or the percentage of weights removed). y-axis shows the percentage drop in mAP as compared to the unpruned network. For all tasks, and object sizes, performance doesn't drop till about 80% sparsity. After which, small objects are hit slightly harder as compared to medium and large objects.

**Do winning tickets behave differently for varying object sizes?** Using the definition from [151], we categorize bounding boxes into small (area $< 32^2$), medium ($32^2 <$ area $< 96^2$), and large (area $> 32^2$). To understand how sparse networks behave for different sized objects, we plot the percentage gain or drop from the mAP of a dense network. Figure 5.4 shows the percentage change in mAP for different levels of sparsity in the Mask R-CNN model. We can observe that in each case, the model performance increases with sparsity, until sparsity reaches 80%, after which, mAP sharply declines. We note that the percentage drop for small boxes is more, with winning tickets (10% of weights) showing a drop of over 17% in case of detection and segmentation tasks while medium sized objects show smaller drops than large objects for all tasks.

**How does the performance of the pruned network vary for rare *vs.* frequent categories?** We sort the 80 object categories in COCO by their frequency of occurrence in training data. We consider networks with 80% and 90% of their weights pruned and observe the percentage change in the bounding box mAP of the model with respect to the unpruned network for each of the categories. Figure 5.5(a) depicts the behavior with a bar graph.

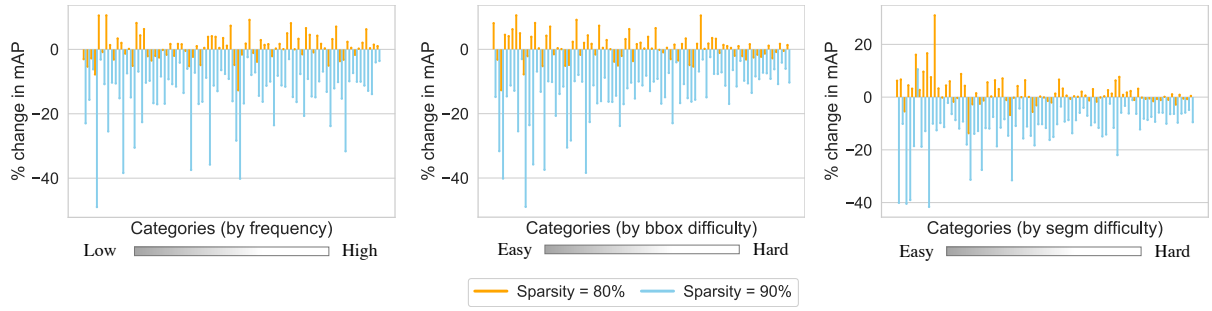



Figure 5.5: Comparison of Mean Average Precision (mAP) of pruned model for 80 COCO object categories. x-axis in each of the plot is a list of categories (sorted using different criteria). y-axis shows the percentage drop in mAP as compared to the unpruned network.

While for most categories, winning tickets are obtained at 80% sparsity, performance drops sharply with more pruning in case of rare categories (such as toaster, parking meter, and bear) as compared to common categories (such as person, car, and chair).

**Do the winning tickets behave differently on easy vs hard categories?** For a machine learning model, an object can be easy or hard to recognize because of a variety of reasons. We have already discussed two reasons that influence the performance — number of instances available in the training data, and size of the object. There can also be other causes that can render an object unrecognizable in given surroundings. Camouflage or occlusion, poor camera quality, light conditions, distance from the camera, or just variations within different instances or views of the object are few of them. Since exhaustive analyses of these causes is intractable, we rank object categories based on performance of an unpruned Mask R-CNN model. We do this categorization for detection and segmentation models as shown in Figure 5.5(b) and (c). Note that 'easy' and 'hard' categories from these two definitions have an overlap but they are not the same. For example, knife, handbag, and spoon are the categories with lowest bounding box mAP, and giraffe, zebra, and stop signs are one with the highest (excluding 'hair drier' which has 0 mAP). On the other hand, skis, knife, and spoon have the lowest segmentation mAP, while stop sign, bear, and fire hydrant

Table 5.4: Effect of ticket transfer across tasks. Transferred tickets do worse than direct training as expected, but still do not result in drastic drops in the mAP or AP50. Here we do task transfer using the 80% pruned model.

| Target task | Source task | Network sparsity | mAP | AP50 |
|---|---|---|---|---|
| Detection | Detection/Segmentation | 78.4% | 30.04 | 49.40 |
| | Keypoint | 50.11% | 23.94 | 41.08 |
| Segmentation | Detection/Segmentation | 78.4% | 27.90 | 46.68 |
| | Keypoint | 50.11% | 23.02 | 39.01 |
| Keypoint | Detection/Segmentation | 76.98% | 58.31 | 81.53 |
| | Keypoint | 79.4% | 59.34 | 82.36 |

have the highest. From the Figure 5.5(b) and (c), we make the following observations — (i) tickets with 80% sparsity can actually increase mAP for certain categories like snowboard by as much as 38%, (ii) Going from 80% to 90% sparsity, mAP drops significantly for easy categories compared to hard categories, (iii) categories that are hit the hardest such as skis, hot dog, spoon, fork, handbags usually have long, thin appearance in images.

**Do winning tickets transfer across tasks?** We showed that ImageNet tickets transfer to a limited extent to downstream tasks. We further study whether the tickets obtained from the downstream task of detection/segmentation transfer to keypoint estimation and vice-versa. We train Mask-RCNN and Keypoint-RCNN respectively for the two tasks on the COCO dataset while maintaining a sparsity level of 80%. For both the tasks we transfer all values till box head modules, after which the model structures differ. The results are shown in Table 5.4. We can observe that the drop is marginal for the transfer of tickets between detection-segmentation to keypoint task, as compared with the reverse case which registers a significant drop. This might be because the ticket is obtained on the keypoint task which is trained only on 'human' class and it fails to transfer well for the detection task which uses the entire COCO dataset.

## 5.5 Discussion

[145, 146] show that winning tickets transfer well across datasets. However, the study in [146] was limited to smaller datasets, like CIFAR-10 and FashionMNIST, and both [145, 146] are limited to classification tasks. We obtain contrasting results when transferring tickets across tasks as shown in Sec. 5.4.2. ImageNet tickets transfer with approximately 40% sparsity to fall within one standard deviation of the baseline network. This is likely due to the fact that winning tickets retain inductive biases from the source dataset which are less likely to transfer to a new domain and task. Additionally, we show that unlike prior LTH works, iterative pruning degrades the performance of subnetworks on detection and one-shot pruning provides the best networks. We also observe that due to the use of pre-trained weights from ImageNet for the backbone of detection networks, late resetting is not necessary for finding winning tickets. This is in contrast to the [37], which is restricted to the classification task involving random initialization for the networks. Like previous works, in our experiments as well, we find that sparse lottery tickets often outperform the dense networks themselves. However, we make another interesting observation — in each of object recognition tasks, tickets with fewer parameters such as ResNet-18 show more gains in performance as compared to tickets with more parameters (ResNet-50). We also find that small and infrequent objects face higher performance drop as the sparsity increases.

## 5.6 Remarks

We investigate the Lottery Ticket Hypothesis in the context of various object recognition tasks. Our study reveals that the main points of original LTH hold for different recognition tasks, *i.e.*, we can find subnetworks or winning tickets in object recognition pipelines with up to 80% sparsity, without any drop in performance on the task. These tickets are task-specific, and pre-trained ImageNet model tickets don't perform as well on the downstream recognition tasks. We also analyse claims made in recent literature regarding training and transfer of winning tickets from an object recognition perspective. Finally, we analyse how the behavior of sparse tickets differ from their dense counterparts. In the future, we would like to investigate how much speed up can be achieved using these sparse models with various hardware [172] and software modifications [173]. Extending this analyses for even bigger datasets such as JFT-300M [174] or IG-1B [129] and for self-supervised learning techniques is another direction to pursue.

## 5.7 Appendix

We provide additional details for some of the experiments presented in the paper. In particular, we provide comparison with a simpler ImageNet ticket transfer alternative in Section 5.7.1, compare the different errors made by dense and pruned models in Section 5.7.2, and finally verify the faster convergence of sparser models in Section 5.7.3.

Table 5.5: Performance on the COCO dataset for ImageNet backbones with mask transfer tickets for ResNet-50 at various levels of pruning. The results for VOC are averaged over 5 runs with the standard deviation in parentheses.

| | COCO Detection | | | COCO segmentation | | | COCO Keypoint | | | VOC Detection | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Prune (%) | Network sparsity | mAP | AP50 | Network sparsity | mAP | AP50 | Network sparsity | mAP | AP50 | Network sparsity | mAP |
| 90 | 41.99 | 35.46 | 56.51 | 41.99 | 32.40 | 52.89 | 31.49 | 62.27 | 84.93 | 65.37 | 61.75 |
| 80 | 37.33 | 36.52 | 57.28 | 37.33 | 33.53 | 54.15 | 28.00 | 63.48 | 85.72 | 58.11 | 67.30 |
| 50 | 24.55 | 37.99 | 58.83 | 24.55 | 34.76 | 55.91 | 19.71 | 64.21 | 86.32 | 36.32 | 70.32 |
| 0 | 0.00 | 38.50 | 59.29 | 0.00 | 35.13 | 56.39 | 0.00 | 64.59 | 86.48 | 0.00 | 71.21 |

### 5.7.1 Mask Transfer Without Retraining

In Section 4.2, we analyzed the effects of transferring tickets only for the ImageNet trained backbones. While this deals with transferring the ticket mask as well as values, we further analyze whether transferring only the mask provides winning tickets for these tasks using the methodology from [146]. We use the default ImageNet weights in the ResNet-18 and ResNet-50 backbone and keep the top $p\%$ of the weights in convolutional layers while setting the rest to zeros and maintaining it throughout the training of the entire network. We refer to this method as 'Mask Transfer'. Since training the backbone on much larger ImageNet data is performed only once, 'Mask Transfer' is a much cheaper or computationally efficient way of obtaining tickets from parent task. We observe that behavior of 'Mask Transfer' is similar to the 'Transfer Ticket' obtained by method discussed in Section 4.2 where the sparse subnetwork weights are fully retrained on ImageNet. Either cases are outperformed by direct pruning on the downstream tasks. The results are summarized in Figure 1 (ResNet-18) and Table 1 (ResNet-50).

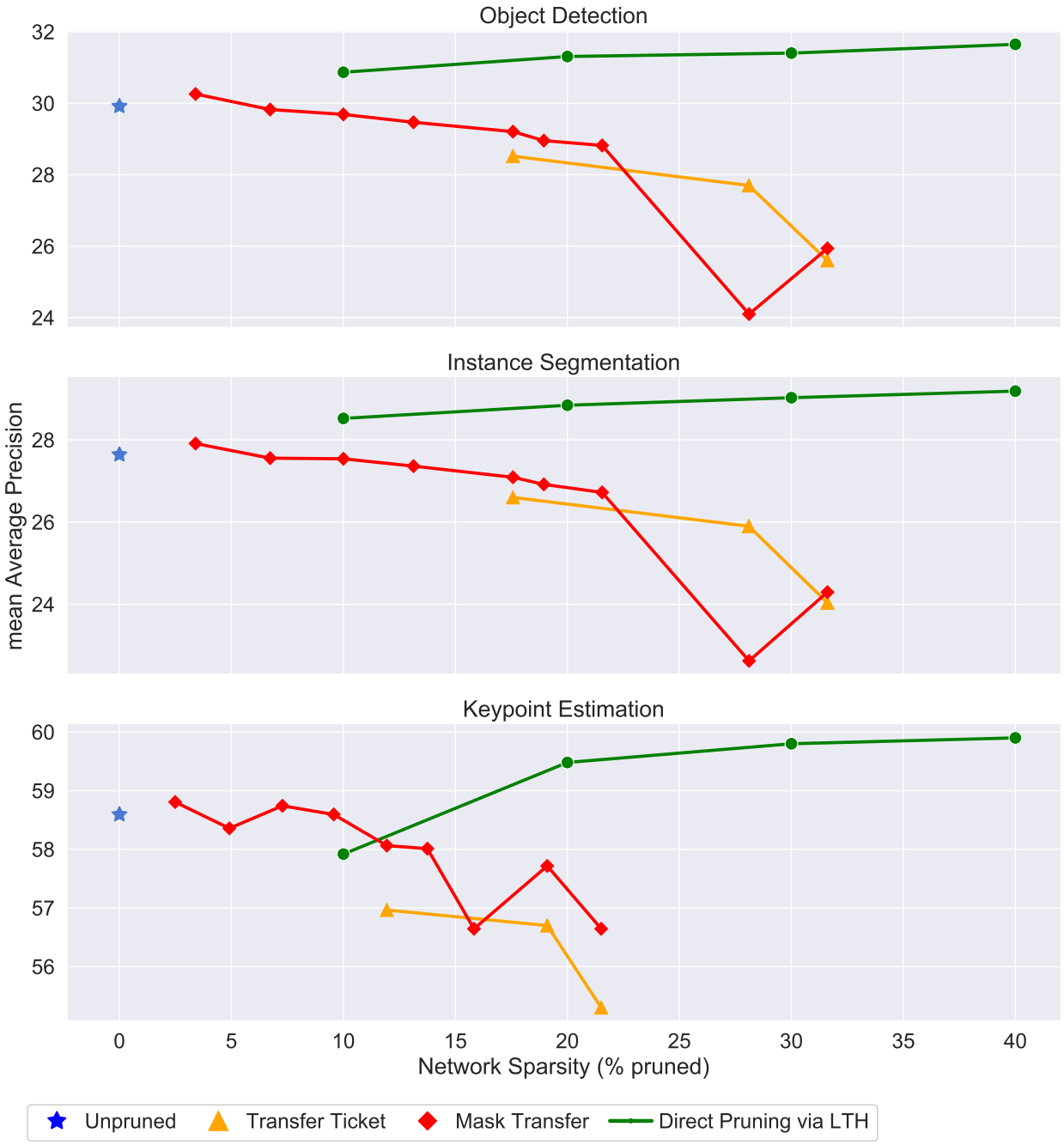



Figure 5.6: Transferring ImageNet backbone tickets to object recognition tasks *vs.* Direct pruning via LTH on the object recognition tasks. We experiment with two variations of transferring ImageNet backbone tickets to object recognition tasks. 'Transfer ticket' refers to the case when we transfer the lottery ticket backbone trained on ImageNet data to downstream task (also discussed in the Section 4 of the paper). 'Mask Transfer' refers to the case when ticket is transferred without retraining on ImageNet, *i.e.*, only the relevant mask from backbone is transferred keeping ImageNet weights the same. Best viewed in color.

### 5.7.2 Error analysis on downstream tasks

The mAP score provides us a good way to summarize the performance of an object recognition model with a single number. But it hides a lot of information regarding what kind of mistakes the model is making. Do the sparse subnetworks obtained by LTH make same mistakes as the dense models? In order to answer this question, we consider a dense Mask R-CNN model with ResNet-50 backbone and a sparse Mask R-CNN model with 20% of the parameters obtained via LTH. Both the models achieve same performance on downstream tasks as also discussed in Section 4.2 of the paper.

#### 5.7.2.1 Object Detection and Instance Segmentation

We resort to a toolbox from [175] to analyze object detection and instance segmentation errors. We consider 5 main sources of errors in object detection. (i) 'Cls' refers to an error corresponding to miss-classification of a bounding box by a model, (ii) 'Loc' refers to the case when bounding box is classified properly but not localized properly, (iii) 'Dupe' corresponds to the errors when model makes multiple predictions at the same location, (iv) 'Bkgd' are the cases when background portion of the image (with no objects) are tagged as an object, and finally (v) 'Missed' cases when the objects are not detected by the model. Figures 5.9 and 5.10 summarize the analysis of detection and segmentation errors obtained for dense model as compared to a sparse model (with only 20% of the weights). While in the case of object detection, the performance of both the models is identical, subtle differences emerge in case of segmentation where sparse model makes fewer localization errors but higher background errors.

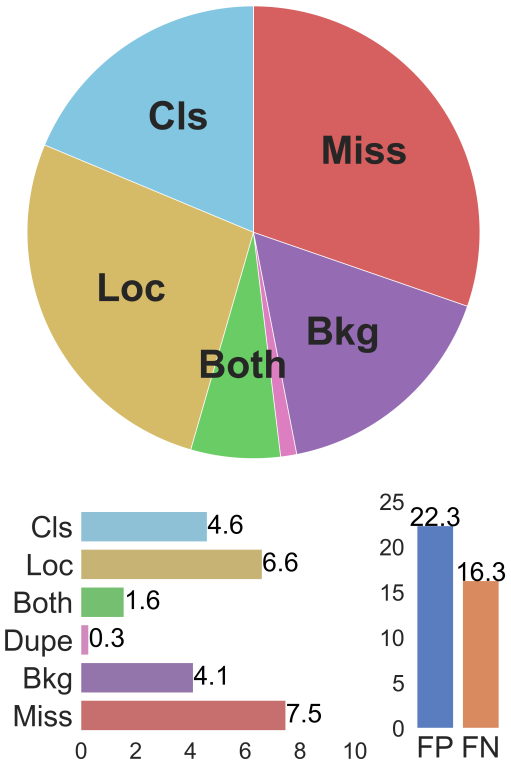



Figure 5.7: Unpruned model

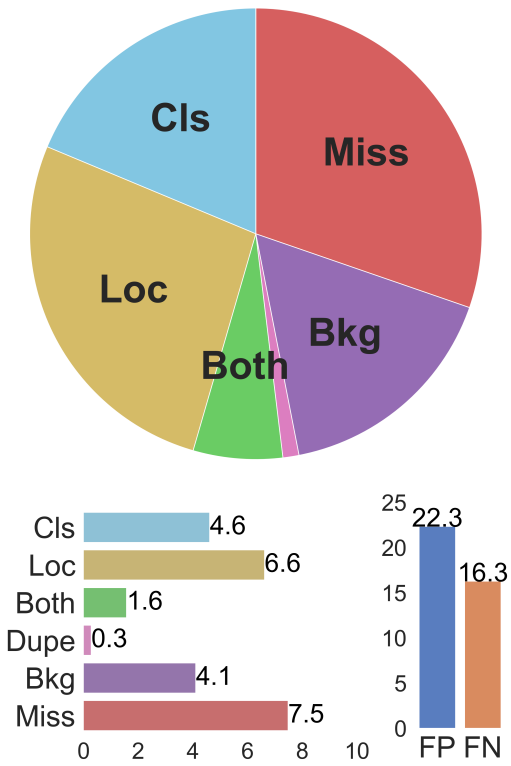
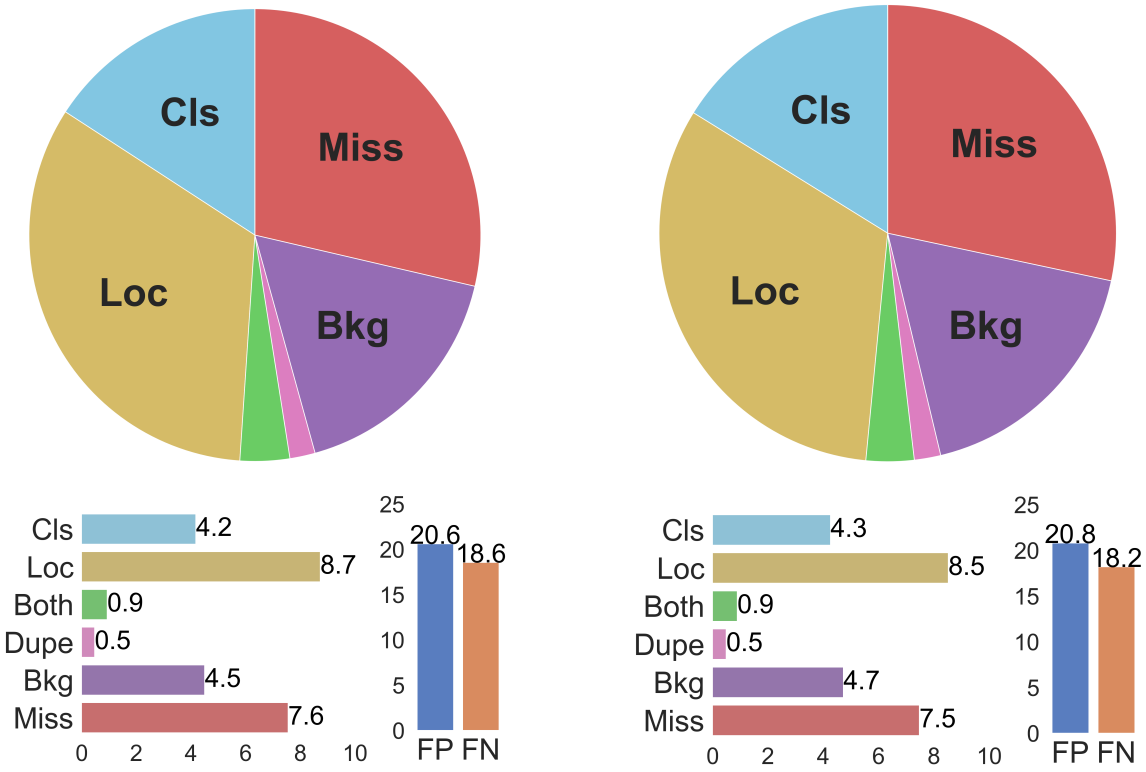



Figure 5.8: Pruned model

Figure 5.9: Error analysis of unpruned vs. pruned models on object detection. The error types of unpruned and pruned models are nearly the same.

#### 5.7.2.2 Keypoint Estimation

We use [176] to perform a similar analyses for sparse and dense models on the task of keypoint estimation. In case of keypoints, we compute the Precision Recall Curve of the model while removing the impact of individual errors of following kinds — (i) 'Miss' - large localization errors, (ii) 'Swap' - confusion between same keypoint of two different persons, (iii) 'Inversion' - confusion between two different keypoints of the same person, (iv) 'Jitter' - small localization error, and (v) 'FP' - background false positives. Fig. 5.12 summarizes the results. As in the previous case, it appears that both the dense and sparse models make similar mistakes.




(a) Unpruned model

(b) Pruned model

Figure 5.10: Error analysis of unpruned vs. pruned models on instance segmentation. The error types of unpruned and pruned models are quite similar.

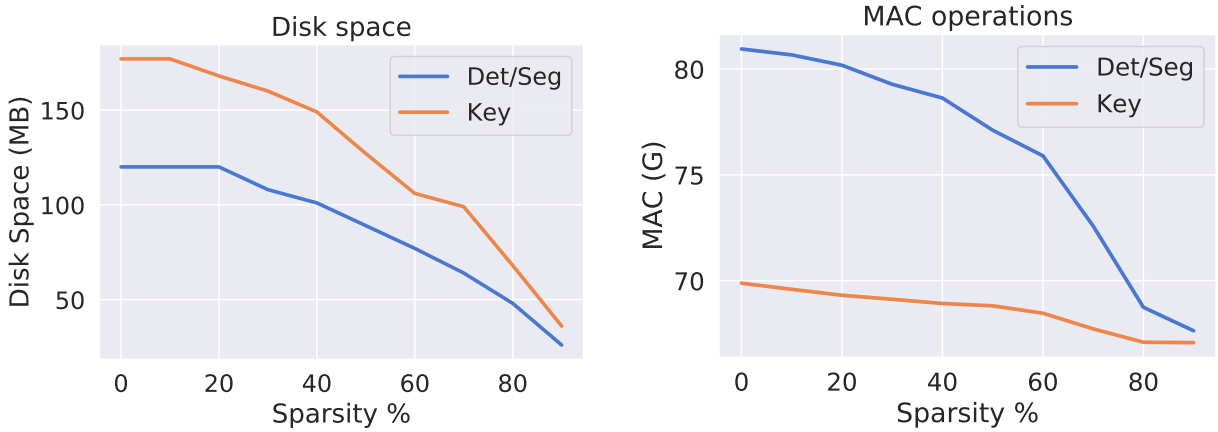
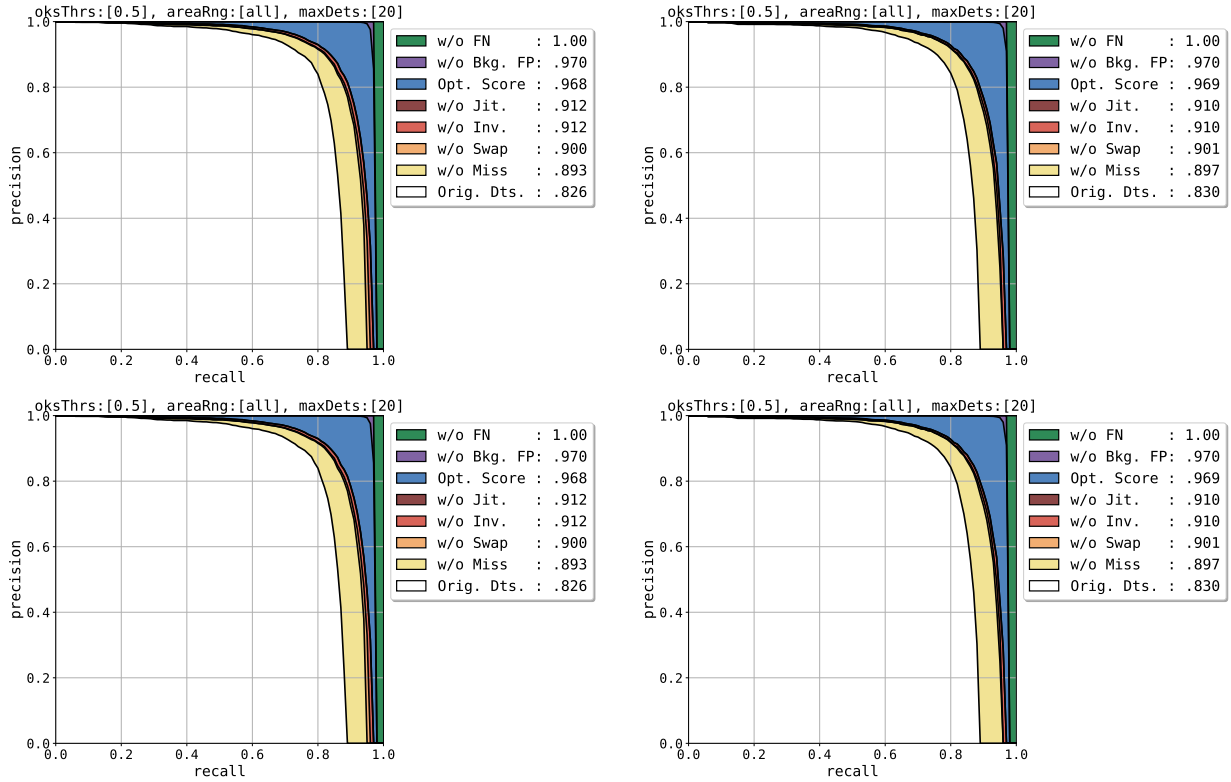



Figure 5.11: Disk space and MAC operations of pruned models with ResNet18 backbone for the various tasks on the COCO dataset.

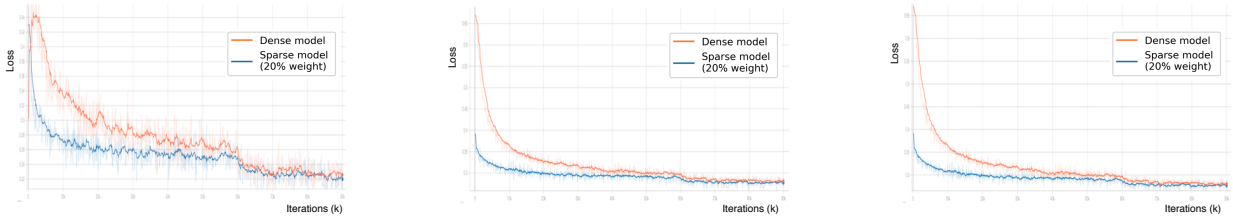



(a) Unpruned model (b) Pruned model

Figure 5.12: Error analysis of unpruned vs. pruned models on keypoint estimation. The error types of unpruned and pruned models are quite similar while the unpruned one has slightly better performance.

### 5.7.3 Sparse subnetworks converge faster

The LTH paper [144] claimed that sparse subnetworks obtained by pruning, often converge faster than their dense counterparts. In this section we verify the claims of the paper on object recognition tasks and found them to hold true. We plot the validation loss during training for the dense unpruned model, and the sparse subnetwork obtained by keeping only 20% of the weights of dense model. Both the models achieve a similar mAP after convergence. Fig. 5.13 shows the task loss against the number of epochs during training. The comparisons confirm that the sparse subnetwork initialized from the winning




(a) Object detection (b) Instance segmentation (c) Keypoint estimation

Figure 5.13: Training curves of dense model vs. sparse model

ticket weights converge much faster. This observation is consistent for heterogeneous tasks, *e.g.*, object detection, instance segmentation, and keypoint estimation.

### 5.7.4 Disk space and compute operations

Finally, we analyze the disk space and MAC operations of pruned models in Figure 5.11. We store the index and values of only the non zero weights, when above a threshold sparsity, while we store the full weight values for denser sparsity levels. As expected, we observe significant reductions in disk space for higher levels of sparsity. However, number of operations decreases at a much slower rate. This can possibly be improved further with dedicated hardware for sparse operations.

# Chapter 6: Future Work and Conclusion

## 6.1 Future work

#### 6.1.0.1 Meta Learning

Meta learning involved learning a prior of the world in which the target model operates, with the aim of quick adaptation towards the end goal. In our paradigm, it would translate to having a good model initialization for faster encoding of INRs. In neural networks, meta-learning typically involves a two-loop optimization process. The outer loop optimizes the meta-parameters (often the initialization of the network), while the inner loop performs task-specific adaptation. This approach allows the model to learn how to learn, enabling rapid adaptation to new tasks with minimal data. Meta-learning algorithms like Model-Agnostic Meta-Learning (MAML) [99] and Reptile [177] have shown promising results in few-shot learning scenarios. To formalize this concept, let's consider a simple meta-learning setup:

For a given task $T_i$ with dataset $D_i$, we perform gradient descent to adapt the parameters $\theta$. Let $\mathcal{L}_{T_i}$ be the loss function for task $T_i$. Update the parameters using:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(\theta)$$

where $\alpha$ is the learning rate for the inner loop. The meta-objective is to minimize the loss across all tasks after adaptation:

$$\min_{\theta} \sum_{i} \mathcal{L}_{T_i}(\theta_i')$$

Update the meta-parameters using:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{i} \mathcal{L}_{T_i}(\theta_i')$$

where $\beta$ is the learning rate for the outer loop. In our case, the dataset $D_i$ can be a large collection of images/videos which is used in learning global priors and the task specific loss refers to overfitting on a new video. Works like [26], [96] have shown that this technique works for image data. I propose to extend these techniques to videos to not only improve encoding speeds, but also reduce storage as the global priors can be considered as a constant, one time cost.

#### 6.1.0.2 Sampling

Another important area which has out-sized influence on training times of INRs is the coordinate sampling. While training implicit networks, the actual input to the network either tends to be coordinate values of each point or in some cases, the centroids of patches which are then mapped to the signal values (with an optional conditional vector). From the perspective of the model, there are $N$ inputs from which we can sample in every forward pass. In the proposed methods, we just take pass all inputs at once and train till convergence which is inefficient.

The DCT least squares theorem states that a reconstruction loss always favours the

low frequency components. This translates to large variation in learning with respect to the signal. For example, a region of the image which represents clear blue sky will be much easy to learn that a region having animal fur. Incorporating this knowledge into the learning process can result in huge speedups of encoding times. The simplest method to try would be to complete the forward pass on all coordinates, but retain only top-K based on loss for the backward pass. More efficient methods which involve Monte-Carlo estimation and soft-mining to achieve sampling during forward pass too [178] have shown impressive results for 3D-scenes. I propose to adapt these techniques to speed-up the encoding times for Video-INRs.

### 6.1.1 Random Networks

In Chapter 5, we talked about how sparse sub-networks found within trained dense networks contain enough information to reproduce the same results. This begs the question - can we find these sparse networks without the additional overhead of re-training? In their work [117] show that a sufficiently large random network can contain sub-networks which match its accuracy. This result has direct impact on INRs for data compression as we only need to store a sparse mask that "selects" this sub-network, while an integer random seed can recover rest of the network. It is important to note that there is an inevitable trade-off in training INR models. More parameters leads to decreased training times, but that comes at the cost of storage. Tricks like quantization aware training can mitigate this but inevitably end up increasing training times! In their work [179] show that it is possible to train NeRV [3] model within such a framework. [180] take a similar approach

for latent based INR frameworks where they use a random matrix as a hypernetwork to predict network weights, resulting in high rates of compression. I propose to extend these techniques to encode videos in order to achieve faster encoding with extremely low bitrates.

## 6.2 Conclusion

This thesis began with a simple question - Can neural networks store data? or are they just analyzers of data? We explored this question in a Rashomon-esque manner, exploring different aspects of building such a system We started with NIRVANA, a practical reimagining of video compression through the lens of implicit neural representations. By adopting patch-wise autoregressive modeling and model-based compression, we showed that INRs can offer a scalable, fast, and resolution-agnostic alternative to traditional codecs.

We then pushed this line of work further in SIEDD, marrying speed and performance. The key innovation was a shared encoder-decoder paradigm that enabled not just faster encoding, but also better scaling, bringing the gap between INR codecs and real-world viability.

But speed and quality weren't enough. Compression without semantics is merely storage. So in Latent-INR, we proposed a framework that infused meaning into compression—enabling retrieval, interpolation, and even video-language tasks through learned per-frame latents. This was the turning point where INRs evolved from being just signal approximators to true representations.

To examine how compression generalizes beyond INRs, we also explored pruning and sparsity via the Lottery Ticket Hypothesis. Through a comprehensive empirical study, we

highlighted how extreme sparsity, if done right, preserves performance across classification, detection, and segmentation. These results deepen our understanding of model compression at scale.

Together, these works argue for a paradigm shift: from compressing data to learning functions that are *the data*. While open questions on generalization, real-time encoding, and universal codecs remain, the trajectory is clear: Compression, when rooted in structure and semantics, becomes a form of intelligence. After all, great compression is understanding and vice versa.

# Bibliography


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

[2] 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.

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

[4] G.K. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992. doi: 10.1109/30.125072.

[5] Vivienne Sze, Madhukar Budagavi, and Gary J Sullivan. High efficiency video coding (hevc). In *Integrated circuit and systems, algorithms and architectures*, volume 39, page 40. Springer, 2014.

[6] Yue Chen, Debargha Murherjee, Jingning Han, Adrian Grange, Yaowu Xu, Zoe Liu, Sarah Parker, Cheng Chen, Hui Su, Urvang Joshi, et al. An overview of core coding tools in the av1 video codec. In *2018 Picture Coding Symposium (PCS)*, pages 41–45. IEEE, 2018.

[7] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[8] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

[9] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018.

[10] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[11] Mingtian Zhang, Andi Zhang, and Steven McDonagh. On the out-of-distribution generalization of probabilistic image modelling. *Advances in Neural Information Processing Systems*, 34:3811–3823, 2021.

[12] Linfeng Cao, Aofan Jiang, Wei Li, Huaying Wu, and Nanyang Ye. Oodhdr-codec: Out-of-distribution generalization for hdr image compression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):158–166, 6 2022. doi: 10.1609/aaai.v36i1.19890. URL `https://ojs.aaai.org/index.php/AAAI/article/view/19890`.

[13] 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.

[14] 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. *ECCV*, 2020.

[15] 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 *European Conference on Computer Vision*, pages 308–324. Springer, 2022.

[16] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 41(4):1–15, 2022.

[17] Sharath Girish, Kamal Gupta, Saurabh Singh, and Abhinav Shrivastava. Lilnetx: Lightweight networks with extreme model compression and structured sparsification. *ArXiv*, abs/2204.02965, 2022.

[18] 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*, MMSys '20, page 297–302, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368452. doi: 10.1145/3339825.3394937. URL `https://doi.org/10.1145/3339825.3394937`.

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

[20] Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: Adaptive coordinate networks for neural scene representation. *arXiv preprint arXiv:2105.02788*, 2021.

[21] Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard G Baraniuk, and Ashok Veeraraghavan. Miner: Multiscale implicit neural representations. *arXiv preprint arXiv:2202.03532*, 2022.

[22] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J. Zico Kolter. Multiplicative filter networks. In *ICLR*, 2021.

[23] David B Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. Bacon: Band-limited coordinate networks for multiscale scene representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16252–16262, 2022.

[24] Shayan Shekarforoush, David B Lindell, David J Fleet, and Marcus A Brubaker. Residual multiplicative filter networks for multiscale reconstruction. *arXiv preprint arXiv:2206.00746*, 2022.

[25] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020.

[26] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, 2021.

[27] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. *arXiv preprint arXiv:2112.04267*, 2021.

[28] Yinbo Chen and Xiaolong Wang. Transformers as meta-learners for implicit neural representations. In *European Conference on Computer Vision*, 2022.

[29] Jaeho Lee, Jihoon Tack, Namhoon Lee, and Jinwoo Shin. Meta-learning sparse implicit neural representations. In *Advances in Neural Information Processing Systems*, 2021.

[30] Daniel Rho, Junwoo Cho, Jong Hwan Ko, and Eunbyung Park. Neural residual flow fields for efficient video representations. *arXiv preprint arXiv:2201.04329*, 2022.

[31] Yunfan Zhang, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. Implicit neural video compression. *arXiv preprint arXiv:2112.11312*, 2021.

[32] Yunpeng Bai, Chao Dong, and Cairong Wang. Ps-nerv: Patch-wise stylized neural representations for videos. *arXiv preprint arXiv:2208.03742*, 2022.

[33] Zizhang Li, Mengmeng Wang, Huaijin Pi, Kechun Xu, Jianbiao Mei, and Yong Liu. E-nerv: Expedite neural video representation with disentangled spatial-temporal context. *arXiv preprint arXiv:2207.08132*, 2022.

[34] Bo He, Xitong Yang, Hanyu Wang, Zuxuan Wu, Hao Chen, Shuaiyi Huang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. Towards scalable neural representation for diverse videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

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

[36] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[37] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[38] Sharath Girish, Shishira R Maiya, Kamal Gupta, Hao Chen, Larry S Davis, and Abhinav Shrivastava. The lottery ticket hypothesis for object recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 762–771, 2021.

[39] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[40] Andrei Ivanov, Nikoli Dryden, and Torsten Hoefler. Sten: An interface for efficient sparsity in pytorch.

[41] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[42] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

[43] Frederick Tung and Greg Mori. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7873–7882, 2018.

[44] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017.

[45] D. Oktay et al. Scalable model compression by entropy penalized reparameterization. In *ICLR*, 2020.

[46] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[47] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016.

[48] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[49] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. Flip: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2), 8 2020. doi: 10.1145/3406183. URL `https://doi.org/10.1145/3406183`.

[50] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward A Practical Perceptual Video Quality Metric, 2016.

[51] Mengyu Chu, You Xie, Jonas Mayer, Laura Leal-Taixé, and Nils Thuerey. Learning temporal coherence via self-supervision for gan-based video generation. *ACM Transactions on Graphics (TOG)*, 39(4):75–1, 2020.

[52] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2004.

[53] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[54] Benjamin Bross, Jianle Chen, Jens-Rainer Ohm, Gary J. Sullivan, and Ye-Kui Wang. Overview of the versatile video coding (vvc) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021.

[55] Li Li, Dong Liu, and Shiqi Wang. Deep contextual video compression. In *Advances in Neural Information Processing Systems*, volume 34, pages 17572–17583, 2021.

[56] Li Li, Dong Liu, and Shiqi Wang. Neural video compression with feature modulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1–10, 2024.

[57] Zhaoyang Jia, Bin Li, Jiahao Li, Wenxuan Xie, Linfeng Qi, Houqiang Li, and Yan Lu. Towards practical real-time neural video compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

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

[59] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI*, pages 74–91. Springer, 2022.

[60] Hao Chen, Saining Xie, Ser-Nam Lim, and Abhinav Shrivastava. Fast encoding and decoding for implicit video representation. In *European Conference on Computer Vision*, pages 402–418. Springer, 2024.

[61] Kushal Vyas, Ahmed Imtiaz Humayun, Aniket Dashpute, Richard G. Baraniuk, Ashok Veeraraghavan, and Guha Balakrishnan. Learning transferable features for implicit neural representations, 2025. URL `https://arxiv.org/abs/2409.09566`.

[62] Chiheon Kim, Doyup Lee, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Generalizable implicit neural representations via instance pattern composers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11808–11817, 2023.

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

[64] Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models, November 2023. URL `https://mobiusml.github.io/hqq_blog/`.

[65] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[66] Siyue Teng, Yuxuan Jiang, Ge Gao, Fan Zhang, Thomas Davis, Zoe Liu, and David Bull. Benchmarking conventional and learned video codecs with a low-delay configuration. In *2024 IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–5. IEEE, 2024.

[67] Dongze Li, Kang Zhao, Wei Wang, Bo Peng, Yingya Zhang, Jing Dong, and Tieniu Tan. Ae-nerf: Audio enhanced neural radiance field for few shot talking head synthesis. *arXiv preprint arXiv:2312.10921*, 2023.

[68] Ho Man Kwan, Ge Gao, Fan Zhang, Andrew Gower, and David Bull. Hinerv: Video compression with hierarchical encoding-based neural representation. *Advances in Neural Information Processing Systems*, 36:72692–72704, 2023.

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

[70] Jiancheng Zhao, Yifan Zhan, Qingtian Zhu, Mingze Ma, Muyao Niu, Zunian Wan, Xiang Ji, and Yinqiang Zheng. Tree-nerv: A tree-structured neural representation for efficient non-uniform video encoding. *arXiv preprint arXiv:2504.12899*, 2025.

[71] Hao Yan, Zhihui Ke, Xiaobo Zhou, Tie Qiu, Xidong Shi, and Dadong Jiang. Ds-nerv: Implicit neural video representation with decomposed static and dynamic codes. *arXiv preprint arXiv:2403.15679*, 2024.

[72] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

[73] J Frankle, G K Dziugaite, D M Roy, and M Carbin. Stabilizing the lottery ticket hypothesis. In *ICML*, 2020.

[74] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[75] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[76] Aaron Defazio, Xingyu Alice Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled, 2024. URL `https://arxiv.org/abs/2405.15682`.

[77] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. FLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(2):15:1–15:23, 2020. doi: 10.1145/3406183.

[78] Kirill Aistov and Maxim Koroteev. Vmaf re-implementation on pytorch: Some experimental results, 2024. URL `https://arxiv.org/abs/2310.15578`.

[79] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`.

[80] Yiping Ji, Hemanth Saratchandran, Cameron Gordon, Zeyu Zhang, and Simon Lucey. Efficient learning with sine-activated low-rank matrices, 2025. URL `https://arxiv.org/abs/2403.19243`.

[81] Gregory K Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

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

[83] Max Ehrlich and Larry S. Davis. Deep residual learning in the jpeg transform domain. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[84] Johannes Ballé, David C. Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *ArXiv*, abs/1802.01436, 2018. URL `https://api.semanticscholar.org/CorpusID:3611540`.

[85] Fabian Mentzer, George Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *ArXiv*, abs/2006.09965, 2020. URL https://api.semanticscholar.org/CorpusID:219721015.

[86] Mingtian Zhang, Andi Zhang, and Steven G. McDonagh. On the out-of-distribution generalization of probabilistic image modelling. In *Neural Information Processing Systems*, 2021. URL https://api.semanticscholar.org/CorpusID:237431305.

[87] Linfeng Cao, Aofan Jiang, Wei Li, Huaying Wu, and Nanyang Ye. Oodhdr-codec: Out-of-distribution generalization for HDR image compression. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 158–166. AAAI Press, 2022. doi: 10.1609/AAAI.V36I1.19890. URL https://doi.org/10.1609/aaai.v36i1.19890.

[88] Subin Kim, Sihyun Yu, Jaeho Lee, and Jinwoo Shin. Scalable neural video representations with learnable positional features. *Advances in Neural Information Processing Systems*, 35:12718–12731, 2022.

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

[90] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:231591445.

[91] Hang Zhang, Xin Li, and Lidong Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*, 2023. URL https://arxiv.org/abs/2306.02858.

[92] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard Baraniuk. Wire: Wavelet implicit neural representations. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18507–18516, 2023. URL https://api.semanticscholar.org/CorpusID:255749557.

[93] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41 (4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127. URL https://doi.org/10.1145/3528223.3530127.

[94] Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard G. Baraniuk, and Ashok Veeraraghavan. MINER: multiscale implicit neural representations. *CoRR*, abs/2202.03532, 2022. URL `https://arxiv.org/abs/2202.03532`.

[95] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning*, 2022. URL `https://api.semanticscholar.org/CorpusID:249395684`.

[96] Yannick Strümpler, Janis Postels, Ren Yang, Luc Van Gool, and Federico Tombari. Implicit neural representations for image compression. In *European Conference on Computer Vision*, 2021. URL `https://api.semanticscholar.org/CorpusID:244954443`.

[97] Jonathan Richard Schwarz and Yee Whye Teh. Meta-learning sparse compression networks, 2022.

[98] Jihoon Tack, Subin Kim, Sihyun Yu, Jaeho Lee, Jinwoo Shin, and Jonathan Richard Schwarz. Learning large-scale neural fields via context pruned meta-learning, 2023.

[99] Chelsea Finn, P. Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017. URL `https://api.semanticscholar.org/CorpusID:6719686`.

[100] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32, 2019.

[101] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 34:19313–19325, 2021.

[102] Pei-Ze Chiang, Meng-Shiun Tsai, Hung-Yu Tseng, Wei-Sheng Lai, and Wei-Chen Chiu. Stylizing 3d scene via implicit representation and hypernetwork. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1475–1484, 2022.

[103] Bipasha Sen, Aditya Agarwal, Vinay P Namboodiri, and CV Jawahar. Inr-v: A continuous representation space for video-based generative tasks. *arXiv preprint arXiv:2210.16579*, 2022.

[104] Bipasha Sen, Gaurav Singh, Aditya Agarwal, Rohith Agaram, K Madhava Krishna, and Srinath Sridhar. Hyp-nerf: Learning improved nerf priors using a hypernetwork. *arXiv preprint arXiv:2306.06093*, 2023.

[105] Jonathan Richard Schwarz, Jihoon Tack, Yee Whye Teh, Jaeho Lee, and Jinwoo Shin. Modality-agnostic variational compression of implicit neural representations. *arXiv preprint arXiv:2301.09479*, 2023.

[106] Matthias Bauer, Emilien Dupont, Andy Brock, Dan Rosenbaum, Jonathan Schwarz, and Hyunjik Kim. Spatial functa: Scaling functa to imagenet classification and generation. *arXiv preprint arXiv:2302.03130*, 2023.

[107] Yunpeng Bai, Chao Dong, Cairong Wang, and Chun Yuan. Ps-nerv: Patch-wise stylized neural representations for videos. In *2023 IEEE International Conference on Image Processing (ICIP)*, pages 41–45. IEEE, 2023.

[108] Bo He, Xitong Yang, Hanyu Wang, Zuxuan Wu, Hao Chen, Shuaiyi Huang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. Towards scalable neural representation for diverse videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6132–6142, 2023.

[109] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, 2016. URL `https://api.semanticscholar.org/CorpusID:7037846`.

[110] Huaizu Jiang, Deqing Sun, V. Jampani, Ming-Hsuan Yang, Erik G. Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9000–9008, 2017. URL `https://api.semanticscholar.org/CorpusID:10817557`.

[111] Hao Chen, Gwilliam Matthew, Bo He, Ser-Nam Lim, and Abhinav Shrivastava. Cnerv: Content-adaptive neural representation for visual data. In *BMVC*, 2022.

[112] A scheme for shot detection and video retreival using spatio temporal features. *International Journal of Recent Technology and Engineering*, 2019. URL `https://api.semanticscholar.org/CorpusID:241499090`.

[113] Max Bain, Arsha Nagrani, Gül Varol, and Andrew Zisserman. Frozen in time: A joint video and image encoder for end-to-end retrieval. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1708–1718, 2021. URL `https://api.semanticscholar.org/CorpusID:232478955`.

[114] Huaishao Luo, Lei Ji, Ming Zhong, Yang Chen, Wen Lei, Nan Duan, and Tianrui Li. Clip4clip: An empirical study of clip for end to end video clip retrieval. *Neurocomputing*, 508:293–304, 2021. URL `https://api.semanticscholar.org/CorpusID:233296206`.

[115] Samuele Papa, Riccardo Valperga, David Knigge, Miltiadis Kofinas, Phillip Lippe, Jan-Jakob Sonke, and Efstratios Gavves. How to train neural field representations: A comprehensive study and benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22616–22625, June 2024.

[116] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10753–10764, 2021.

[117] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11893–11902, 2020.

[118] Huilan Luo, Yuan Chen, and Yifeng Zhou. An extremely effective spatial pyramid and pixel shuffle upsampling decoder for multiscale monocular depth estimation. *Computational Intelligence and Neuroscience*, 2022, 2022. URL `https://api.semanticscholar.org/CorpusID:251272212`.

[119] Yansong Tang, Dajun Ding, Yongming Rao, Yu Zheng, Danyang Zhang, Lili Zhao, Jiwen Lu, and Jie Zhou. Coin: A large-scale dataset for comprehensive instructional video analysis, 2019.

[120] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[121] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos, 2017.

[122] Shyamal Buch, Cristóbal Eyzaguirre, Adrien Gaidon, Jiajun Wu, Li Fei-Fei, and Juan Carlos Niebles. Revisiting the "video" in video-language understanding, 2022.

[123] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[124] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[125] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[126] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[127] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[128] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[129] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, pages 181–196, 2018.

[130] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[131] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[132] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965, 2018. URL `http://arxiv.org/abs/1811.06965`.

[133] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[134] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[135] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in neural information processing systems*, pages 963–971, 2014.

[136] Zhiyong Cheng, Daniel Soudry, Zexi Mao, and Zhenzhong Lan. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562*, 2015.

[137] Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. Fully quantized network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2810–2819, 2019.

[138] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[139] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in neural information processing systems*, pages 3084–3092, 2013.

[140] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[141] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[142] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[143] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[144] J Frankle and M Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

[145] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pages 4932–4942, 2019.

[146] R Mehta. Sparse transfer learning via winning lottery tickets. In *NeurIPS*, 2020.

[147] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[148] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[149] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[150] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[151] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[152] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *arXiv preprint arXiv:1912.05671*, 2019.

[153] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

[154] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[155] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011.

[156] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[157] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[158] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

[159] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[160] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29:2074–2082, 2016.

[161] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through $l\_0$ regularization. *arXiv:1712.01312*, 2017.

[162] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[163] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3597–3607, 2019.

[164] Kathrin Grosse and Michael Backes. How many winning tickets are there in one dnn? *arXiv preprint arXiv:2006.07014*, 2020.

[165] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[166] H You, C Li, P Xu, Y Fu, Y Wang, X Chen, R G Baraniuk, Z Wang, and Y Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *ICLR*, 2020. URL `https://openreview.net/forum?id=BJxsrgStvr`.

[167] Andrey Salvi and Rodrigo Barros. An experimental analysis of model compression techniques for object detection. In *Anais do VIII Symposium on Knowledge Discovery, Mining and Learning*, pages 49–56, Porto Alegre, RS, Brasil, 2020. SBC. doi: 10.5753/kdmile.2020.11958. URL `https://sol.sbc.org.br/index.php/kdmile/article/view/11958`.

[168] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[169] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. *arXiv preprint arXiv:2012.06908*, 2020.

[170] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 1998.

[171] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[172] Liqiang Lu, Jiaming Xie, Ruirui Huang, Jiansong Zhang, Wei Lin, and Yun Liang. An efficient hardware accelerator for sparse convolutional neural networks on fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 17–25. IEEE, 2019.

[173] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14629–14638, 2020.

[174] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[175] Daniel Bolya, Sean Foley, James Hays, and Judy Hoffman. Tide: A general toolbox for identifying object detection errors. In *ECCV*, 2020.

[176] Matteo Ruggero Ronchi and Pietro Perona. Benchmarking and error diagnosis in multi-instance pose estimation. In *ICCV*, 2017.

[177] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[178] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. Accelerating neural field training via soft mining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20071–20080, 2024.

[179] Hee Min Choi, Hyoa Kang, and Dokwan Oh. Is overfitting necessary for implicit video representation? In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[180] Cameron Gordon, Lachlan Ewen MacDonald, Hemanth Saratchandran, and Simon Lucey. D'oh: Decoder-only random hypernetworks for implicit neural representations. *arXiv preprint arXiv:2403.19163*, 2024.