





A Cloud 3D Dataset and Application-Specific Learned Image Compression in Cloud 3D

Tianyi Liu , Sen He , Vinodh Kumaran Jayakumar , and Wei Wang 

The University of Texas at San Antonio

tianyi.liu@utsa.edu, sen.he@utsa.edu, vinodhkumaran.jayakumar@my.utsa.edu,
wei.wang@utsa.edu

Abstract. In Cloud 3D, such as Cloud Gaming and Cloud Virtual Reality (VR), image frames are rendered and compressed (encoded) in the cloud, and sent to the clients for users to view. For low latency and high image quality, fast, high compression rate, and high-quality image compression techniques are preferable. This paper explores computation time reduction techniques for learned image compression to make it more suitable for cloud 3D. More specifically, we employed slim (low-complexity) and application-specific AI models to reduce the computation time without degrading image quality. Our approach is based on two key insights: (1) as the frames generated by a 3D application are highly homogeneous, application-specific compression models can improve the rate-distortion performance over a general model; (2) many computer-generated frames from 3D applications are less complex than natural photos, which makes it feasible to reduce the model complexity to accelerate compression computation. We evaluated our models on six gaming image datasets. The results show that our approach has similar rate-distortion performance as a state-of-the-art learned image compression algorithm, while obtaining about 5x to 9x speedup and reducing the compression time to be less than 1 second (0.74s), bringing learned image compression closer to being viable for cloud 3D. Code is available at <https://github.com/cloud-graphics-rendering/AppSpecificLIC>.

Keywords: Cloud Gaming, Cloud Virtual Reality, Learned Image Compression, Model Simplification, Application-specific Modeling, Model-Task Balance

1 Introduction

Image compression plays an important role in cloud 3D, including cloud gaming and cloud virtual reality (VR). Compared with local (non-cloud) 3D applications, cloud 3D needs an extra step, image/video encoding and streaming, to transmit compressed (encoded) frames of 3D applications to end users. The smaller the file size of the compressed image (i.e., higher compression rate), the lower network bandwidth usage and network latency will be. However, high compression rate usually either implies longer compression/decompression time or lower

image quality. Therefore, fast, high compression rate, and high-quality image compression techniques are highly preferable.

Recently, learned image compression [17,4,5,29,19,10,28,23,27,14,3,41,12,9] has shown great potential for further increasing the compression rate, while maintaining similar image quality. By replacing traditional discrete cosine transform (DCT) [40] or discrete wavelet transform (DWT) [32] with deep neural networks, the latent representation after transform operation can have smaller data sizes (less spatial redundancy). Typical learned image compression frameworks [17,4,5,29] utilize cascaded auto-encoder/decoder layers to optimize the entropy encoding. Some studies [17,23] also aggregate information from all the decoder layers to get a better compression rate and reconstruction quality. Therefore, learned image compression has demonstrated better or comparable compression capability, compared to traditional methods [40,32,6,7].

Although learned image compression has made great progress, there is still a major challenge that hinders its application to cloud 3D: **The high inference latency of compression/decompression violates the timing constraint of cloud 3D**. For example, the Coarse-to-Fine model [17] takes about 5 and 7 seconds to compress and decompress an image with $1920 \times 1080p$ resolution. Compression and decompression at several seconds are not tolerable for cloud 3D, as its latency is typically required to be less than a second [11].

In this work, we focus on reducing the computation time for learned image compression for cloud 3D while preserving low bitrate and high image quality. More specifically, we employed low-complexity and application-specific AI models to reduce the computation time without degrading image quality. Our approach is based on two key insights: (1) as the frames generated by a 3D application are highly homogeneous, application-specific models can improve the rate-distortion overall a general model; (2) many of the computer-generated frames from real 3D applications are less complex than natural photos, which also makes it feasible to employ less complex models to accelerate computation.

We collected six gaming image datasets to assist the development of our learned image compression method. We evaluated our simplified application-specific models on these datasets. The results show that our approach has similar rate-distortion performance as a state-of-the-art (SOTA) learned image compression algorithm, while obtaining 5x to 9x speedup and reducing the compression time to be less than 1 second, bringing learned image compression closer to being viable for cloud 3D. Our method also has a lower bitrate than BPG and VTM.

Note that, in this work, we focus on image compression than video-based encoding. Our results show that, for cloud 3D, the bitrate of our image compression solution is about 3 times better than H.264 videos (about 20 Mbps V.S. 60 Mbps), which is commonly used for cloud gaming [26]. Moreover, this image compression is similar to the I-Frames (independent frames) in videos. Using images (i.e., only I-Frames) have better tolerance over lost frames, which is common in cloud 3D, as their decoding does not depend on previous (lost) frames, and thus, potentially providing a better user experience. Finally, our conclusions can also be applied to the encoding of I-Frames for video-based cloud 3D.

In summary, the contributions of this work include:

1. Six gaming image datasets for learned image compression research on rendered images.
2. A low-complexity and application-specific approach for learned image compression for images generated/rendered by 3D applications.
3. A thorough evaluation of the proposed approach with various parameters which showcases the viability of learned image compression for cloud 3D.

The rest of this paper is organized as following: Section 2 discusses the related work; Section 3 explains the problem and presents our approach; Section 4 demonstrates the effectiveness of our methodologies. Section 5 discusses special design issues, and Section 6 concludes the paper.

2 Related Work

2.1 Traditional Image Compression

Traditional image compression frameworks employ several processes: image transformation, quantization, and entropy encoding. JPEG, first proposed by Wallace in 1992 [40], utilizes Discrete Cosine Transform (DCT) to make key information compact at the left-up corner of an image. JPEG2000 [32] was proposed later, which uses Discrete Wavelet Transform (DWT) to further improve the compression rate. Based on HEVC video encoding standard, Bellard proposed a new compression solution, BPG [6], which uses intra image prediction techniques and has been the state-of-the-art algorithm until VTM. Based on the latest video encoding standard, VVC or H.266 [7], VTM uses improved methods, such as larger coding units, more intra-prediction modes, and more transform types, to increase image compression rate. However, VTM is complex and may spend several hundred seconds in compression computation.

2.2 Learned Image Compression

As deep neural networks (DNN) have demonstrated excellent modeling and representation ability on computer vision tasks, many deep learning-based image compression schemes are proposed to improve the rate-distortion performance of image compression. Based on network types, these algorithms fall into two categories: recurrent models (RNN) [37,38,22,20] and convolutional models (CNN) [4,5,29,19,17,10,28,23,27,14,3,41,12,9].

Recurrent neural network (RNN) models typically compress images or the residual information progressively. Toderici et al. proposed a variable-rate RNN model [37], which is the first to utilize convolutional LSTM to compress images. Their approach supports end-to-end training and can generate multiple bitrates through a single model. Later, Toderici et al. and Johnston et al. introduced full-resolution RNN [38] and priming RNN [20], both of which achieve comparable and even better performance when compared with BPG in terms of image quality (MS-SSIM). Spatial RNN [22] excels BPG significantly by removing the

redundant information among larger pixel ranges. However, RNN models tend to have high complexity, especially for higher ranges of bit-rates.

Being less complex, CNN methods are widely studied recently. Ballé explored an auto-encoder/decoder architecture with GDN network [4] to compress and reconstruct images. Later, a hyperprior model [5] is invented to predict the probability distribution of pixels to help with the entropy coding, which is comparable to BPG. The autoregressive entropy model [29] is based on the hyperprior model and utilizes context information to further remove spatial redundancy. EDIC model [23] and checkerboard context model [14] accelerate the decoding process for the autoregressive entropy model [29]. Coarse-to-Fine model [17] is proposed in 2020 and becomes the SOTA image compression solution. Coarse-to-Fine adds one more auto-encoder/decoder layer to the hyperprior architecture and aggregated information from all the auto-encoder/decoder layers to get better compression rate and reconstruction quality. However, the compression and decompression time of Coarse-to-Fine is usually high [34]. Slimmable network architectures [41,43,42] provide an effective solution for variable rate and adaptive complexity, making the AI models suitable for resource-limited devices.

Our work is inspired by these prior studies. Nonetheless, more work is still required to speed up learned image compression for real-time processing to meet cloud 3D’s latency and quality requirements.

2.3 Formulation of Learned Image Compression

A basic learned image compression framework [4] follows the equations below:

$$\begin{aligned} y &= g_a(x, \alpha) \\ \hat{y} &= Q(y); b \leftarrow \xi(\hat{y}); \hat{y} \leftarrow \xi^{-1}(b) \\ \hat{x} &= g_s(\hat{y}, \beta) \end{aligned} \tag{1}$$

where g_a and g_s are analysis and synthesis transforms that are used to reduce image redundancy and reconstruct the image. α and β are optimized parameters for analysis and synthesis transforms. x and \hat{x} denote the input raw image and the reconstructed image. y is the latent representation, which is quantized by Q . After quantization, the result \hat{y} is imported into ξ for entropy encoding. The symbol b denotes the bitstream after image compression. For image decompression, the bitstream b is imported into ξ^{-1} for entropy decoding. The entropy encoding and decoding processes are lossless [34,35]. Hence, the quantized latent representation \hat{y} can be fully recovered after these two steps. Finally, g_s reconstructs the raw image from \hat{y} . Since quantization is a lossy process, the reconstructed information \hat{x} may be different from the original input x . The difference between x and \hat{x} is the reconstruction distortion.

For quantization, round-based quantization values can be used to generate \hat{y} . However, learned image compression usually employs end-to-end models, and the round-based quantization method is non-differentiable, making it challenging for end-to-end training. Prior work [4] solved this problem by adding a uniform noise $u(-0.5, +0.5)$ to signal y during model training. This method creates a

noisy signal \tilde{y} , which is used to approximate \hat{y} during model training. That is, this solution makes the quantization process differentiable.

For entropy coding, prior work [4] used a basic factorized model, and later the authors proposed a more advanced trainable hyperprior model [5]. The hyperprior model added another auto-encoder/decoder layer to the baseline model, and the new layer is used for probability estimation. To model long term dependency, the Coarse-to-Fine model [17] proposed hierarchical layers of hyperpriors to conduct more comprehensive analysis, which can be formulated as,

$$\begin{aligned}
 y &= g_a(x, \alpha); z = h_{a1}(y, \alpha_1); w = h_{a2}(z, \alpha_2) \\
 \hat{y} &= Q(y); \hat{z} = Q(z); \hat{w} = Q(w); \\
 p_{\hat{z}|\hat{w}}(\hat{z}|\hat{w}) &\leftarrow h_2 \leftarrow h_{s2}(\hat{w}; \theta_{s2}) \\
 p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z}) &\leftarrow h_1 \leftarrow h_{s1}(\hat{z}; \theta_{s1}) \\
 \hat{x} &\leftarrow g_s(\hat{y}, \beta), h_1, h_2
 \end{aligned} \tag{2}$$

where $g_a, g_s, x, \hat{x}, y, \hat{y}, \alpha, \beta$, and Q are same as the Formula 1. h_{a1} and h_{s1} are the auto-encoder and decoder in the first auxiliary layer, while h_{a2} and h_{s2} are the auto-encoder and decoder in the second auxiliary layer. After the transformation of g_a , the latent representation y (output of transformation) is imported into h_{a1} . h_{a1} 's output z is in turn imported into h_{a2} to produce w . Then, y, z , and w are further quantized into \hat{y}, \hat{z} , and \hat{w} . h_1 and h_2 are side information from h_{s1} and h_{s2} , which help reconstruct the original image, along with $g_s(\hat{y}, \beta)$. $p_{\hat{z}|\hat{w}}(\hat{z}|\hat{w})$ is the estimated distribution conditioned on \hat{w} , and $p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z})$ is the estimated distribution conditioned on \hat{z} . $p_{\hat{z}|\hat{w}}(\hat{z}|\hat{w})$ and $p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z})$ are used to do entropy coding of \hat{y} and \hat{z} . The distribution of \hat{w} uses the typical factorized model.

The typical loss function for learned image compression is a rate-distortion trade-off. For the Coarse-to-Fine model, the bitrate includes three parts: R_y, R_z , and R_w . The distortion, $D(x, \hat{x})$, is the difference between x and \hat{x} . The loss function can be described as,

$$\begin{aligned}
 L &= R + \lambda D = R_y + R_z + R_w + \lambda D(x, \hat{x}) \\
 &= E[-\log_2 p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z})] + E[-\log_2 p_{\hat{z}|\hat{w}}(\hat{z}|\hat{w})] + E[-\log_2 p_{\hat{w}}(\hat{w})] + \lambda D(x, \hat{x}),
 \end{aligned} \tag{3}$$

where λ is a trade-off parameter. Larger λ leads to better (i.e., less) distortion, and hence, higher bitrate. According to information theory, the minimum bitrate to encode signal x is the cross entropy of the real and estimated distribution of x . The $E(\cdot)$ calculates the size of output bitstream from each layer.

3 Proposed Research

3.1 Application-Specific Learned Image Compression in Cloud 3D: A Practical Use Case

Learned image compression is a data-driven method, and the quality of the reconstructed image is highly dependent on the training dataset and the practical

use case. The image quality would become poor if obvious differences are found between the training dataset and practical use case. To solve the problem of image quality drift (degradation), we propose application-specific learned image compression, which collects datasets and trains AI models for each specific use case. To make this solution more practical, we are targeting the cloud 3D system.

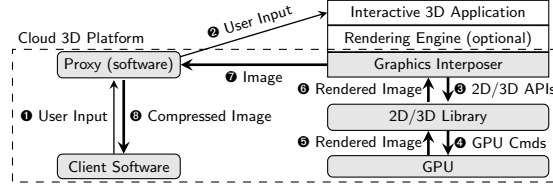


Fig. 1. A typical architecture of cloud 3D system.

Figure 1 illustrates a typical architecture of a cloud 3D system. Cloud 3D platform and 3D applications are the two main components, and the 3D applications run above the cloud 3D platform. For the cloud 3D platform, it involves server-side software (server proxy) and client-side software, and these two parts are separated by the network. On the client-side, the user interacts with the client software that is also responsible for displaying frames/images of remote 3D applications. The server-side includes several modules: proxy software, graphics interposer, 2D/3D library, and GPU. The proxy receives user inputs (step 1), forwards inputs to corresponding 3D applications (step 2), and transmits rendered frames from the server-side to client-side (step 3). The graphics interposer intercepts API calls from 3D applications and invokes the real 2D/3D library in the cloud (step 3). The GPU at the bottom of cloud 3D platform is responsible for executing rendering commands (step 4). In cloud 3D, 3D applications are offloaded from local computers to the cloud, and the GUIs of 3D applications are transmitted to the client-side in the form of compressed images (step 8). Therefore, image compression plays an important role in cloud 3D streaming.

Note that, an AI model is trained for each 3D application to conduct application-specific image compression/decompression. Hence, instead of downloading and installing a 3D application on local computers, an AI model (which is typically smaller than a 3D application) will be downloaded for decoding.

3.2 Cloud 3D Image Dataset

Images in cloud 3D systems are rendered by programs. To study learned image compression for cloud 3D images, datasets are collected from six 3D applications.

Dataset Collection Pictor benchmark suite [25] is created for cloud 3D research. We used this benchmark suite to collect GUI frames of each 3D application automatically. More specifically, after rendering a frame on GPU, the pixels of the current frame will be copied from the GPU memory to CPU memory (step 5, 6, and 7 in Figure 1). Before sending the frame to client side, we

compress the frame into the PNG format [1] and save it to disk (in the graphics interposer or the server proxy of Figure 1).

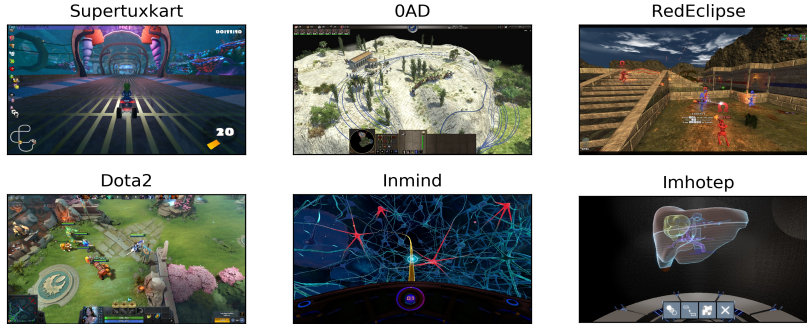


Fig. 2. A sample of cloud 3D image dataset from six 3D applications. Each 3D application has 2000 images. 1000 images are 1080p, and the other 1000 images are 720p.

Dataset Construction We have collected six datasets from six 3D applications with two resolutions, 1920×1080 and 1280×720). Each 3D application has 2000 images. 1000 images are 1080p, and the other 1000 images are 720p. For each resolution, there are 800 images for training, 100 images for validation, and 100 images for testing. With each game having 2000 images, there are 12000 images in total. Figure 2 gives a sample of the cloud 3D image dataset. Supertuxkart [16] is an open-source racing game; 0AD [13] is an open-source real-time strategy game; Red Eclipse [33] is a first-person shooting game, and it is also open-source. Dota2 [39] is a highly popular online battle arena game. Inmind [30] and Imhotep [31] are VR applications.

Dataset Analysis Rendered images from 3D applications usually have fewer details and are relatively simpler than natural photos. To verify this intuition, we analyzed the complexity of rendered images and natural photos that are from the cloud 3D dataset and DIV2K_train_HR dataset respectively. Figure 3 provides quantitative evidence by comparing the difference between these two datasets in terms of entropy density. Entropy, $H(X) = -\sum_{i=1}^n p(x_i) \log p(x_i)$, is

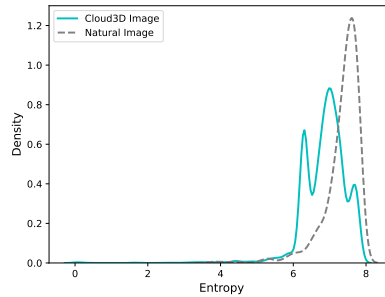


Fig. 3. Entropy density comparison of Cloud 3D images and natural images. The natural image dataset is from DIV2K_train_HR [2,18].

a typical metric to evaluate the complexity of an image. Figure 3 shows that the majority of natural photos’ entropy is on the right side of (i.e., larger than) the cloud 3D datasets, indicating that **cloud 3D images statistically have lower entropy and are simpler than natural photos**. Interestingly, Figure 3 also shows that the entropy of cloud 3D images has a wide distribution, and one of the peak values is very close to that of natural photos, indicating that some rendered images have comparable complexity with natural photos.

3.3 A Slim Framework: How Slim the Framework Can Be?

The SOTA learned image compression [17] has outperformed the traditional image compression solutions. However, it has complex structures, such as cascaded DNNs. These structures make image compression and decompression time-consuming. As discussed in Section 3.2, rendered images are simpler and have fewer details than natural photos. However, it is unclear if these simpler images can lead to more lightweight frameworks with lower bitrates to make it feasible for compressing gaming images. To further explore this feasibility, we studied the large, medium, small, xsmall models, and pruned models.

Table 1. Simplifying the Coarse-to-Fine model by continuously reducing the number(♯) of channels in the main and auxiliary auto-encoder/decoders.

Model Name	#of chnls in g_a/g_s	#of chnls in h_{a1}/h_{s1}	#of chnls in h_{a2}/h_{s2}
Large Com.	(3, [384, 384, 384, 384])	(384, [768, 1536, 256])	(256, [512, 256, 128])
Large Dec.	(384, [384, 384, 384, 3])	(256, [1536, 1536, 384])	(128, [512, 512, 256])
Medium Com.	(3, [192, 192, 192, 192])	(192, [384, 768, 256])	(256, [512, 128, 128])
Medium Dec.	(192, [192, 192, 192, 3])	(256, [768, 384, 192])	(128, [128, 256, 256])
Small Com.	(3, [96, 96, 96, 96])	(96, [192, 384, 256])	(256, [256, 128, 128])
Small Dec.	(96, [96, 96, 96, 3])	(256, [384, 192, 96])	(128, [128, 256, 256])
XSmall Com.	(3, [48, 48, 48, 48])	(48, [96, 192, 256])	(256, [256, 128, 128])
XSmall Dec.	(48, [48, 48, 48, 3])	(256, [192, 96, 48])	(128, [128, 256, 256])
Pruning Com.	Same as above	Same as above	pruned
Pruning Dec.	Same as above	Same as above	pruned

Large, Medium, Small, and Extra-Small Models To reduce the computation time of the Coare-to-Fine model, we aim to obtain a slim framework by continuously reducing the number of channels in g_a , g_s , h_{a1} , h_{s1} , h_{a2} and h_{s2} (Please refer to Formula 2 and prior work [17]). Table 1 shows the details of large, medium, small, and xsmall models. “*Com” denotes image compression (analysis), while “*Dec” denotes image decompression (synthesis). Each model has separate analysis and synthesis processes. In each row of Table 1, the numbers stand for the input and output channels (chnls) of a CNN layer. E.g., in (3,

[384, 384, 384, 384]), 3 refers to the RGB channels of an image, and [384, 384, 384, 384] denotes 4 convolutional layers each with 384 filters.

Framework Pruning Coarse-to-Fine proposed hierarchical hyperprior layers to estimate the probability for entropy coding. Other studies [9,23] also proved that side information can benefit image reconstruction. Therefore, our framework kept these structures. Nonetheless, although adding one more hyperprior layer benefits probability estimation, it also increases the complexity of the model and the computation time. As we target cloud 3D, which has simpler images, we removed the last hyper-prior layer to further speed up the computation time (see Figure 4 and the ‘‘Pruning *’’ rows in Table 1).

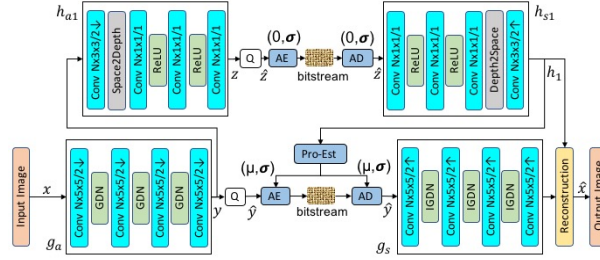


Fig. 4. A slim framework for image compression and decompression.

After pruning h_{a2} and h_{s2} , the original model becomes a slim framework. In Figure 4, for image compression, the raw image x is first transformed by g_a , and then the output y is quantized into \hat{y} for arithmetic encoding (AE) in the bottom path. In the top path, y is sent to h_{a1} . After quantization and AE, y is compressed into bitstream. The h_{s1} in the top path generates h_1 , which is imported into a probability estimation (Pro-Est) module to help with AE in the bottom path. For image decompression, the bitstream in the top path is imported into the arithmetic decoding (AD) and further synthesized by h_{s1} . Finally, with h_1 , the image is reconstructed after g_s processes \hat{y} . Consequently, inference and loss function can be simplified as Formula 4 and 5,

$$\begin{aligned}
 y &= g_a(x, \alpha); z = h_{a1}(y, \alpha_1) \\
 \hat{y} &= Q(y); \hat{z} = Q(z); \\
 p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z}) &\leftarrow h_1 \leftarrow h_{s1}(\hat{z}; \theta_{s1}) \\
 \hat{x} &\leftarrow g_s(\hat{y}, \beta), h_1
 \end{aligned} \tag{4}$$

$$\begin{aligned}
 L &= R + \lambda D = R_y + R_z + \lambda D(x, \hat{x}) \\
 &= E[-\log_2 p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z})] + E[-\log_2 p_{\hat{z}}(\hat{z})] + \lambda D(x, \hat{x})
 \end{aligned} \tag{5}$$

Although most of the symbols have been described in this section, please also refer to Formula 2 and 3 for more details about other symbols.

3.4 Model-Task Balance on GPU

GPUs usually have limited video memory, which makes it challenging to run large-scale AI models with high-resolution images. Splitting big images (e.g., 1080p/2K/4K/8K) into smaller tiles (e.g., 256×256) might help learned image compression. These small tiles can fit into GPU memory, allowing large AI model processing in GPU without memory error. However, processing these tiles in a sequential manner is suboptimal.

To further speed up the image processing on GPU, we employed Model-Task Balance (M-T-Balance). **The key idea is to increase the task size (size of input tile) as we switch to more lightweight learned image compression models.** More specifically, as we continuously simplify or prune the compression model, the input size can be increased accordingly. After getting a slim framework (section 3.3), the AI model occupies less video memory, while the saved video memory can be utilized to accommodate larger image tiles. In this way, the computing resources in GPU can be fully utilized, which, in turn, increases the parallelism and further speeds up the compression and decompression process. In our evaluation, our final slim framework allows whole image frames to be stored in GPU memory.

4 Evaluation

Rate-distortion performance, compression/decompression speedup, and visualization quality are evaluated in our experiments. For each dataset, the model was trained with an initial λ value of 0.004. However, as a hyperparameter, λ was tuned for each application-specific model, with potential values of 0.01, 0.02, 0.04, 0.08, 0.16, 0.002, 0.001, 0.0005, 0.0002, and 0.0001. Each model was trained for 2500 epochs until the loss became stable. The speedup evaluation was conducted on a server with an 8-core Intel i7-7820x CPU, 16GB memory, and an NVIDIA GTX1080Ti GPU with 11GB GPU memory.

4.1 Performance Comparison

Rate-Distortion Comparison. Our method (slim framework with application-specific learned image compression) has similar or better rate-distortion performance, when compared with other algorithms. Figure 5 compares the rate-distortion performance of our approach with the Coarse-to-Fine model, BPG, and VTM. In the first five subfigures of Figure 5, our method is very close to the performance of the SOTA model. In the last subfigure for Imhotep, our method has significant performance improvement and saved about 0.3 bpp (bits per pixel) with a PSNR of 36dB. When comparing with BPG and VTM, our method has similar performance on Supertuxkart, 0AD, and Dota2 datasets. For the other three datasets, our method outperformed BPG and VTM. The Imhotep dataset is particularly interesting, as it only has a liver in the image with regular patterned background (see Figure 2), our compressing scheme is more effectively on this dataset over other algorithms.

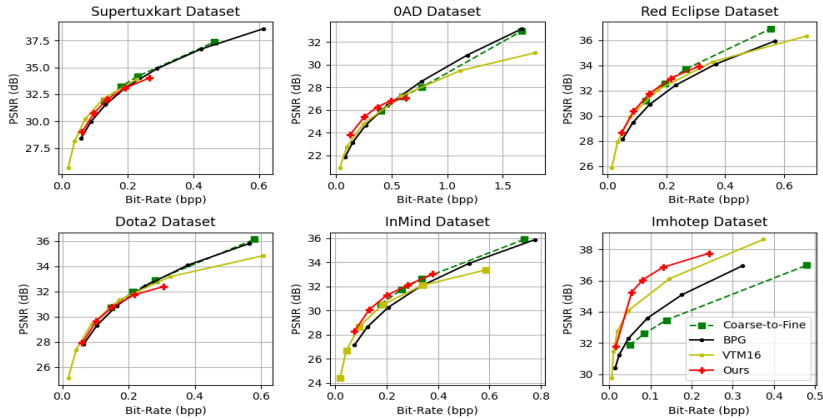


Fig. 5. Rate-Distortion curves (PSNR) of Coarse-to-Fine model [17], handcrafted BPG4:4:4 [6], VTM16-4:2:0 [36], and our method on six gaming datasets.

Computing Speedup. The goal of this work is to speed up the compression and decompression of learned image compression without harming the rate-distortion performance. The previous evaluation has shown that our method has better image quality at similar bitrates than other methods, the computing time will be further evaluated. Table 2 compares the computing time of our approach with the Coarse-to-Fine (SOTA) model, BPG, and VTM. Table 2 shows that our method outperforms the other three methods on image compression speed. More specifically, the compression time is reduced to 0.74s, which is about 9x and 156x faster than Coarse-to-Fine model and VTM, respectively. For image decompression, our method is about 4.7x faster than Coarse-to-Fine model, but slower than BPG and VTM. Further reduction of computing time will be investigated in the future (See Section 5 for more discussion).

Table 2. Computing time comparison (in seconds) of Coarse-to-Fine model [17], BPG [6], VTM-16 [36], and our method on six gaming image datasets.

Computing Time (s)	Coarse-to-Fine (SOTA)	BPG-4:4:4	VTM16-4:2:0	Ours
Compression	6.52	0.76	115.16	0.74
Decompression	7.72	0.38	0.39	1.64

4.2 Ablation Study

Application-Specific Compression. To illustrate the impact of application-specific model, we also report the performance of the application-specific models

with the “large” size. Figure 6 compares the rate-distortion performance of the Coarse-to-Fine model (green curve) and the application-specific models (blue curve with square marker). Figure 6 shows that the application-specific scheme improves the rate-distortion performance of all applications significantly on both high and low bitrates. In particular, Imhotep has the largest improvement, and the reason of the large improvement is discussed in Section 4.1. Similarly, given the same bitrate, our application-specific method can improve PSNR by 2 to 4dB in most cases.

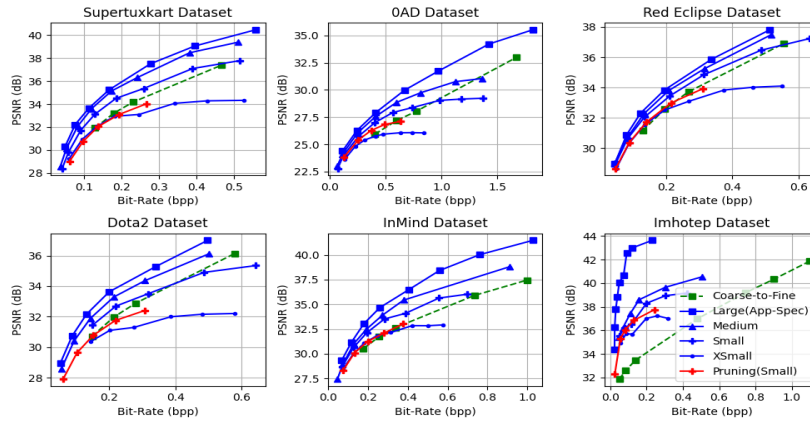


Fig. 6. Rate-Distortion curves of the Coarse-to-Fine model [17] and application-specific learned image compression (with model pruning) on six datasets.



Fig. 7. Comparison on visual quality with the Coarse-to-Fine model, BPG, VTM, and our method. This sample image is 665.png under “test/dota2-1080p/” directory.

Rate-Distortion Comparison of Model Simplification and Pruning.

To determine the proper size of a slim framework, an ablation study on each simplified model is also conducted. As shown in Table 1, large, medium, small, and xsmall models are trained and evaluated, and the rate-distortion performance is shown in Figure 6. From Figure 6, it is observed that the performance

of these models gradually approaches the Coarse-to-Fine model as we keep simplifying the original model. The xsmall model has worse performance than the Coarse-to-Fine model, whereas the other three models behave well. As the small model has lower complexity, it is selected to perform hyperprior layer pruning, which achieves similar or better performance when compared with Coarse-to-Fine as shown in Figure 6.

Computing Speedup. We also analyze the compression and decompression time for the simplified and pruned models. Their computing time is shown in Table 3. The M-T-Balance scheme is implemented on the “Small” model, and the pruning scheme is based on the “Small” model and employs the M-T-Balance. Table 3 shows that our model simplification can effectively reduce the computing time for both compression and decompression. Moreover, M-T-Balance achieved impressive speedup, and its average computing time is reduced by about 37%. At last, pruning the last hyperprior layer saved another 0.3 seconds.

Table 3. Computing time comparison of application-specific learned image compression with different model sizes, model-task balance (based on Small model), and model pruning (based on the small model and M-T-Balance).

Computing Time (s)	Large	Medium	Small	XSmall	M-T-Balance (Small)	Pruning
Compression	4.16	2.24	1.63	1.43	0.93	0.74
Decompression	5.27	3.42	2.60	2.25	1.75	1.64

4.3 Visualization Study

To exemplify the effectiveness and image quality of our approach, visual examples of some reconstructed images are presented in Figure 7. A relatively complex image is selected from the Dota2 dataset (as Dota2 is a popular 3D game and is welcomed by many players). In this figure, we compared our approach with the learned image compression algorithm, Coarse-to-Fine, as well as hybrid coding methods (BPG and VTM). Figure 7 shows that these images are very close, and we can barely distinguish these four images based on visual qualities. This is coherent with the quantitative results presented in Figure 5.

In summary, our method has a lower compression time, similar or better rate-distortion performance, compared with Coarse-to-Fine, BPG, and VTM.

5 Discussion and Future Work

Model Size in Bytes. The model size of our solution is about 30.9 MB. As our models are application-specific, to use our models in cloud 3D, users need to download a model for each 3D application. As the models are small, this

download should not be an obstacle, especially when considering many existing mobile games are significantly larger than our models.

Comparison with H.264 Video Encoding. Contemporary cloud gaming, such as Google’s Stadia [8], primarily employed video encoding to stream the game images to the client. The most common encoding standard used is H.264 [26,15,24,21,8]. Due to space limitation, we cannot provide a detailed comparison with H.264 video-based encoding. The results are summarized there. On average, at PSNR 30 to 35, the H.264 video bitrate for our datasets is more than 60Mbps, which is higher than our solution. That is, our solution (and learned image compression in general) has better bitrates than H.264 encoding.

The main benefit of H.264 is its faster encoding and decoding, especially with hardware accelerators. This fast computation makes H.264 more suitable for real-time encoding for cloud 3D than the general learned image compression. However, our application-specific solution can significantly reduce the compression/decompression time for learned compression. Although our solution is still slower than H.264, further optimizations (more in the next paragraph) may make learned image compression’s computation time closer to H.264. With smaller bitrates (thus smaller network latency), the overall latency of learned image compression can be potentially lower than H.264 for cloud 3D.

Further Reduction of Computation Time. This paper focuses on the slim and application-specific model design for cloud 3D image compression. Therefore, we only conducted limited optimization (i.e., the Model-to-Task balance) on the implementation of our models. In the future, we plan to explore more optimization options, such as GPU code optimization, FPGA implementation, and/or ASIC (Application-specific Integrated Circuits) implementations, to eventually make learned compression feasible for cloud 3D.

6 Conclusions

This work focuses on reducing the computing time of learned image compression to make it one step closer to meeting the real-time requirement of cloud gaming and VR. We proposed application-specific compression to reduce the model complexity to speed up model computation time. Evaluations show that our approach significantly accelerated compression/decompression without degrading image quality, making learned compression potentially viable for cloud 3D.

Acknowledgment

This work was partially supported by the National Science Foundation under grants, 2221843, 2155096, 2202632, and 2215359. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of NSF. The authors would like to thank the anonymous reviewers for their insightful comments. We would also like to thank Kebin Peng for his valuable input.

References

1. libpng. <http://www.libpng.org/pub/png/libpng.html>, [Online; accessed 17-Sep-2020]
2. Agustsson, E., Timofte, R.: Ntire 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 126–135 (2017)
3. Bai, Y., Liu, X., Zuo, W., Wang, Y., Ji, X.: Learning Scalable Y=+Constrained Near-Lossless Image Compression via Joint Lossy Image and Residual Compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 11946–11955 (June 2021)
4. Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end Optimized Image Compression. arXiv preprint arXiv:1611.01704 (2016)
5. Ballé, J., Minnen, D., Singh, S., Hwang, S.J., Johnston, N.: Variational Image Compression with a Scale Hyperprior. arXiv preprint arXiv:1802.01436 (2018)
6. Bellard, F.: Bpg image format. <http://bellard.org/bpg/> (2017)
7. Bross, B., Chen, J., Liu, S., Wang, Y.K.: Versatile video coding editorial refinements on draft 10. JVET-Q2002-v3 (2020)
8. Carrascosa, M., Bellalta, B.: Cloud-gaming: Analysis of google stadia traffic. CoRR **abs/2009.09786** (2020), <https://arxiv.org/abs/2009.09786>
9. Chen, T., Liu, H., Ma, Z., Shen, Q., Cao, X., Wang, Y.: End-to-End Learnt Image Compression via Non-Local Attention Optimization and Improved Context Modeling. IEEE Transactions on Image Processing **30**, 3179–3191 (2021)
10. Cheng, Z., Sun, H., Takeuchi, M., Katto, J.: Learned Image Compression With Discretized Gaussian Mixture Likelihoods and Attention Modules. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
11. Claypool, M., Claypool, K.: Latency and player actions in online games. Commun. ACM **49**(11), 40–45 (nov 2006)
12. Cui, Z., Wang, J., Gao, S., Guo, T., Feng, Y., Bai, B.: Asymmetric Gained Deep Image Compression With Continuous Rate Adaptation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10532–10541 (June 2021)
13. Games, W.: 0 A.D. <https://play0ad.com/>, [Online; accessed 11-Nov-2018]
14. He, D., Zheng, Y., Sun, B., Wang, Y., Qin, H.: Checkerboard Context Model for Efficient Learned Image Compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 14771–14780 (June 2021)
15. Hegazy, M., Diab, K., Saeedi, M., Ivanovic, B., Amer, I., Liu, Y., Sines, G., Hefeeda, M.: Content-aware video encoding for cloud gaming. In: Proceedings of the 10th ACM Multimedia Systems Conference (2019)
16. Henrichs, J.: SuperTuxKart. https://supertuxkart.net/Main_Page, [Online; accessed 11-Nov-2018]
17. Hu, Y., Yang, W., Liu, J.: Coarse-to-Fine Hyper-Prior Modeling for Learned Image Compression. In: Proceedings of the AAAI Conference on Artificial Intelligence (2020)
18. Ignatov, A., Timofte, R., et al.: PIRM Challenge on Perceptual Image Enhancement on Smartphones: Report. In: European Conference on Computer Vision (ECCV) Workshops (January 2019)

19. Jiang, F., Tao, W., Liu, S., Ren, J., Guo, X., Zhao, D.: An End-to-End Compression Framework Based on Convolutional Neural Networks. *IEEE Transactions on Circuits and Systems for Video Technology* (2018)
20. Johnston, N., Vincent, D., Minnen, D., Covell, M., Singh, S., Chinen, T., Hwang, S.J., Shor, J., Toderici, G.: Improved lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018)
21. Lai, Z., Hu, Y.C., Cui, Y., Sun, L., Dai, N.: Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In: *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking* (2017)
22. Lin, C., Yao, J., Chen, F., Wang, L.: A Spatial RNN Codec for End-to-End Image Compression. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020)
23. Liu, J., Lu, G., Hu, Z., Xu, D.: A Unified End-to-End Framework for Efficient Deep Image Compression. *arXiv preprint arXiv:2002.03370* (2020)
24. Liu, L., Zhong, R., Zhang, W., Liu, Y., Zhang, J., Zhang, L., Gruteser, M.: Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering. In: *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services* (2018)
25. Liu, T., He, S., Huang, S., Tsang, D., Tang, L., Mars, J., Wang, W.: A Benchmarking Framework for Interactive 3D Applications in the Cloud. In: *Int'l. Symp. on Microarchitecture (MICRO)* (2020)
26. Meng, J., Paul, S., Hu, Y.C.: Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (2020)
27. Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., Gool, L.V.: Practical Full Resolution Learned Lossless Image Compression. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019)
28. Mentzer, F., Gool, L.V., Tschannen, M.: Learning Better Lossless Compression Using Lossy Compression. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020)
29. Minnen, D., Ballé, J., Toderici, G.: Joint Autoregressive and Hierarchical Priors for Learned Image Compression. *arXiv preprint arXiv:1809.02736* (2018)
30. Nival: InMind VR. <https://luden.io/inmind/>, [Online; accessed 22-July-2018]
31. Pfeiffer, M., Kenngott, H., Preukschas, A., Huber, M., Bettscheider, L., Müller-Stich, B., Speidel, S.: IMHOTEP: virtual reality framework for surgical applications. *International Journal of Computer Assisted Radiology and Surgery* **13**(5) (May 2018)
32. Rabbani, M., Joshi, R.: An overview of the jpeg 2000 still image compression standard. *Signal processing: Image communication* (2002)
33. Reeves, Q., Salzman, L.: Red Eclipse: A Free Arena Shooter Featuring Parkour. <https://www.redeclipse.net/>, [Online; accessed 11-Nov-2018]
34. Rippel, O., Bourdev, L.: Real-Time Adaptive Image Compression. In: *Proceedings of the 34th International Conference on Machine Learning*. pp. 2922–2930 (2017)
35. Rissanen, J., Langdon, G.: Universal Modeling and Coding. *IEEE Transactions on Information Theory* (1981)
36. Suehring, K.: VVCSoftware_VTM. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM (2022)

37. Toderici, G., O'Malley, S.M., Hwang, S.J., Vincent, D., Minnen, D., Baluja, S., Covell, M., Sukthankar, R.: Variable Rate Image Compression with Recurrent Neural Networks. arXiv preprint arXiv:1511.06085 (2015)
38. Toderici, G., Vincent, D., Johnston, N., Jin Hwang, S., Minnen, D., Shor, J., Covell, M.: Full Resolution Image Compression With Recurrent Neural Networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
39. Valve: InMind VR. <http://blog.dota2.com/?l=english>, [Online; accessed 22-July-2018]
40. Wallace, G.K.: The jpeg still picture compression standard. IEEE transactions on consumer electronics (1992)
41. Yang, F., Herranz, L., Cheng, Y., Mozerov, M.G.: Slimmable Compressive Autoencoders for Practical Neural Image Compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4998–5007 (June 2021)
42. Yu, J., Huang, T.S.: Universally Slimmable Networks and Improved Training Techniques. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1803–1811 (2019)
43. Yu, J., Yang, L., Xu, N., Yang, J., Huang, T.: Slimmable Neural Networks. arXiv preprint arXiv:1812.08928 (2018)