MobileCodec: Neural Inter-frame Video Compression on Mobile Devices

Hoang Le
Qualcomm AI Research
San Diego, USA

Liang Zhang
Qualcomm AI Research
San Diego, USA

Amir Said
Qualcomm AI Research
San Diego, USA

Guillaume Sautiere
Qualcomm AI Research
Amsterdam, Netherlands

Yang Yang
Qualcomm AI Research
San Diego, USA

Pranav Shrestha
Qualcomm AI Research
San Diego, USA

Reza Pourreza
Qualcomm AI Research
San Diego, USA

Auke Wiggers
Qualcomm AI Research
Amsterdam, Netherlands

ABSTRACT
Realizing the potential of neural codecs on real-world mobile devices is a big technological challenge due to the inherent conflict between the computational complexity of deep networks and the power-constrained mobile hardware performance. We demonstrate practical feasibility by leveraging Qualcomm’s innovation and technology, bridging the gap from neural network-based model simulations to operation on a mobile device powered by Snapdragon® technology. We show the first-ever inter-frame neural video decoder running on a commercial mobile phone, decompressing high-definition videos in real-time while maintaining a low bitrate and high visual quality, comparable to conventional codecs.

CCS CONCEPTS
• Computer systems organization → Real-time system architecture.

KEYWORDS
video compression, neural codec, efficient architecture, mobile

ACM Reference Format:
Hoang Le, Liang Zhang, Amir Said, Guillaume Sautiere, Yang Yang, Pranav Shrestha, Fei Yin, Reza Pourreza, and Auke Wiggers. 2022. MobileCodec: Neural Inter-frame Video Compression on Mobile Devices. In 13th ACM Multimedia Systems Conference (MMSys ’22), June 14–17, 2022, Athlone, Ireland. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3524273.3532906

1 INTRODUCTION
Video compression technologies have been actively researched and engineered over decades to obtain broad video adoption across a wide range of devices, distribution media, and services. As neural network (NN) technologies have been revolutionizing the world, NN-based video coding technology, especially deep generative model-based approaches, has a strong potential to further extend the capabilities and efficiency of video compression. Instead of manually designing a sophisticated pipeline with multiple stages and optimizing each stage separately, a NN-based codec can learn to directly extract and code a low-dimension feature representation of the input data in an end-to-end process to offer a better rate-distortion tradeoff. Moreover, it is easier to upgrade and deploy new NN-based codecs, since their models are trained in a relatively short amount of time in comparison to the development of conventional video compression codecs. Equally important, they can leverage AI hardware accelerators that are already ubiquitous in many different computing platforms.

Recent advances in deep learning have revolutionized the way lossy data compression algorithms are designed. Specifically, we have seen remarkable success in image and video compression, achieved by employing a convolutional autoencoder as a flexible learned non-linear transform [2, 4], deep generative models as powerful entropy models [5, 25], and various neural motion estimation and compensation algorithms for high-quality frame interpolation and extrapolation [44, 45]. After a few years of development, neural video codecs [1, 18, 29] are now on-par or even outperform traditional solutions such as HEVC [38], and have become a vibrant research area.

Besides competitive rate-distortion performance, learnable codecs also open new opportunities such as easy adaptation to different data distributions [34], enable fast parallel entropy coding, region-of-interest coding to allocate more bits to semantically interesting regions, and use of perceptual metric for direct optimization of visual quality [21, 23].

However, most of the studies use wall-powered high-end GPUs with floating-point computation, and the neural network models are often not optimized for fast and practical inference, especially being able to run in real-time on a mobile device. In this paper,
we demonstrate how our resource-aware, efficient neural architecture design, paired with a novel parallel entropy coding for neural codecs, and leveraging the Qualcomm® AI Engine, can achieve the first real-time neural inter-frame decoding on a commercial mobile device.

The main contributions of this paper are:
- We present a demonstration of the first real-time neural inter-frame video decoder on a mobile device.
- We present an efficient network architecture for a neural inter-frame codec specifically designed for deployment on a mobile platform.
- We introduce a parallel entropy coding algorithm tailored for NN-based entropy estimation models.

2 RELATED WORK

Neural image compression: There has been significant progress in the development of neural image compression solutions in recent years. While earlier works proposed different types of generative models [22, 32, 39, 40], many recent works are based on the hierarchical hyperprior architecture [3, 5, 6, 20] and its variants equipped with autoregressive models [25, 26] and attention mechanisms [8] are currently the most widely adopted architectures.

Neural video compression: Neural video compression has become an emerging topic in the last couple of years and there are existing solutions for both low-latency [16, 18, 19] and video streaming applications [9, 10, 12, 29, 42]. The low-latency solutions often include unidirectional motion estimation/compensation followed by a residual correction [1, 7]. Recent works have introduced recurrence [11, 33] or longer range dependencies [15, 31] to improve performance. In the settings where it is acceptable to incur latency by using reference frames from future timepoints, the existing solutions include bidirectional motion estimation/compensation [10] or frame interpolation [9, 42] followed by residual correction, or implicit multi-frame coding based on 3D convolutions [12].

Efficient solutions: Despite the great progress in developing neural image and video compression solutions, the existing literature to address their efficiency and practicality is relatively limited. Efficient architectures are explored in [14, 17, 31, 32] to reduce the number of operations. Overfitting to a video or segments of a video is another means to improve the rate-distortion of a neural codec that can eventually lead to lightweight models [41]. Yet, these methods, in their current form, are far from running real-time on resource-constrained mobile devices and hence better suited to run on high-end GPUs. To the best of our knowledge, there is a single successful attempt to deploy neural video codecs on mobile devices [30] that runs in all-intra mode, i.e., all frames are coded as images.

3 APPROACH

Our approach to enable real-time neural inter-frame decoder on a mobile device includes three main steps. First, we design a mobile-friendly neural network architecture, MobileCodec, optimized for highly efficient performance on a mobile device (Section 3.1). Secondly, we exploit quantization-aware training [27] to quantize network parameters and activations for efficient fixed-point operations on a mobile device while preserving the network’s original floating-point performance (Section 3.2). Finally, we develop a parallel entropy coding algorithm that leverages the parallelism property of neural network-based codecs to optimize the entropy coding process (Section 3.3). We describe these steps in more details in the following sections.

3.1 Mobile friendly network architecture

To optimize our MobileCodec for performance on a mobile device, we use a network consisting only of highly efficient convolutional layers. Figure 1 shows the overall design of our network architecture. Specifically, our MobileCodec consists of two codecs: an intra-frame MobileCodec that compresses each video frame independently (as an image), and an inter-frame MobileCodec that compresses a frame conditionally on a previously decoded frame. The inter-frame MobileCodec is often used to compress the first frame of each video segment while the inter-frame MobileCodec is used to compress the rest of the frames of that segment.

Intra-frame MobileCodec. The design of our Intra-frame MobileCodec is derived from the image compression method by Balle et al. [5]. Please refer to the original paper for a detailed description of the method. Briefly, as shown in Figure 1, the encoder (or analysis transform) \( I_0 \) transforms each frame \( x_0 \) into its embedded latent \( y^f = I_0(x_0) \). This embedded latent \( y^f \) is then quantized by the rounding operator \( Q \), and entropy coded by an arithmetic encoder (AE) before being sent to the receiver where it will be decoded by an arithmetic decoder (AD) and reconstructed back to the image \( \hat{x}_0 \) via the synthesis decoder \( I_f: \hat{x}_0 = I_f(y^f) \). Following [5], we also use a hypercodec network to model the density of the encoded latent \( y^f \). Specifically, the hyper-encoder \( H^f \) transforms the latent \( y^f \) into \( z^f = H^f(y^f) \). \( z^f \) is also quantized, entropy coded, and sent to the receiver where it will be decoded and fed into a hyper-decoder network \( H^0 \) to predict the scale \( \sigma^f \) of the latent \( \hat{y}^f: \hat{y}^f = H^0(z^f) \). The predicted scale \( \sigma^f \) is used for arithmetic encoding (AE) only in the transmitter side and arithmetic decoding (AD) in both transmitter and receiver sides for losslessly coding the quantized latent \( \hat{y}^f \). Note that, in contrast to [5], we use ReLU as the nonlinearity after each layer of the network instead of a GDN block. Our study showed that using ReLU is more quantization friendly for MobileCodec while still being able to obtain comparable performance.

Inter-frame MobileCodec. We follow the recent design of neural inter-frame codec methods [1, 18] and introduce our inter-frame MobileCodec, which includes two consecutive sub-networks: a motion network and a residual network. First, the motion net takes a previously decoded frame \( \tilde{x}_{t-1} \) and the current frame \( x_t \) to extract a latent motion between them and use it to reconstruct the frame \( \tilde{x}_t \) at the receiver end. The residual network encodes the difference \( r_t \) between the reconstructed frame \( \tilde{x}_t \) and the current frame \( x_t \). This reconstructed residual \( \hat{r}_t \) is then used to refine the reconstructed frame \( \tilde{x}_t \) to obtain the final reconstructed frame \( \hat{x}_t \) at the receiver end.

3.1.1 Flow-Agnostic Motion Compensation. Motion compensation is an important component that contributes significantly in saving bitrate for an inter-frame codec by exploiting temporal redundancies in contrast to an all-intra video codec, which compresses each frame independently as image compression. Most of the existing
methods implement motion compensation either by pixel warping [1, 18] or feature warping [13]. These methods require large memory buffering to enable memory access to the arbitrary location of a video frame (or its features). However, these required operations are often not efficient on a mobile device due to its high-computation and memory requirements. Moreover, these operations do not scale well with image resolution. In contrast, convolutional operations operate on each image block and are highly efficient on mobile devices. For this reason, we design a flow-agnostic motion compensation algorithm that leverages fully convolutional operations while avoiding explicitly using less-efficient pixel or feature warping operations. As illustrated in Figure 1, our motion compensation method includes two feature extractors $F_{\text{prev}}$ and $F_{\text{curr}}$ to extract representative features $f_{t-1}$ and $f_t$ of the previously decoded frame $\hat{x}_{t-1}$ and the current frame $x_t$, respectively:

$$f_{t-1} = F_{\text{prev}}(\hat{x}_{t-1})$$  \hfill (1)

$$f_t = F_{\text{curr}}(x_t)$$  \hfill (2)
The extracted features \( f_{t-1} \) and \( f_t \) are concatenated as input into the feature correlation module \( M_c \) to extract the motion features between the two frames:

\[
y^M = M_c(concat(f_{t-1}, f_t))
\]  

These motion features are quantized via the quantizer \( Q \), then coded via arithmetic coding \( AE \), and sent to the receiver.

At the receiver end, the decoded motion features \( \hat{y}^M \) are concatenated with the feature \( f_{t-1} \) extracted from the previous frame \( \hat{x}_{t-1} \), which is available for both transmitter and receiver. These features are then fed into an image synthesis network \( M_s \) to reconstruct the current frame \( \hat{x}_t \).

\[
\hat{x}_t = M_s(concat(f_{t-1}, \hat{y}^M))
\]  

Next, the residual error \( r_t = x_t - \hat{x}_t \) of the reconstructed image \( \hat{x}_t \) is fed into a residual analysis encoder \( R_a \) to extract the latent feature \( y^R \).

\[
y^R = R_a(r_t)
\]  

This latent feature \( y^R \) is then also quantized via the quantizer \( Q \) and entropy coded via \( AE \), and sent to the receiver. On the receiver side, the decoded latent \( \hat{y}^R \) is input into the residual decoder \( R_s \) to reconstruct the residual \( \hat{r}_t \).

\[
\hat{r}_t = R_s(\hat{y}^R)
\]  

The final reconstructed frame is generated by adding the reconstructed residual \( \hat{r}_t \) to the motion compensated image \( \hat{x}_t \): 

\[
\hat{x}_t = \hat{x}_t + \hat{r}_t
\]  

We follow [5] and use a hyperprior network to predict the scale \( \sigma^M \) and \( \sigma^R \) for each latent feature in \( y^M \) and \( y^R \) to be entropy coded respectively. Specifically, the hyperprior analysis encoder \( H_a^M \) transforms the motion latent \( y^M \) into motion hyperlatent \( z^M \).

\[
\hat{z}^M = H_a^M(y^M)
\]  

This motion hyperlatent \( z^M \) is quantized, then entropy coded and sent to the receiver. At the receiver end, the decoded latent \( \hat{z}^M \) is input into a hyperdecoder network \( H_s^M \) to predict the scale \( \sigma_M \) which is then used for entropy encoding and decoding for the latent \( \hat{y}^M \). The same process is also employed for the entropy coding of the residual latent \( \hat{y}^R \).

### 3.1.2 Asymmetric neural network architecture.

We leverage an asymmetric architecture to enable real-time decoding on a mobile device while achieving high-quality decoded results by having a bigger encoder. This design is also in line with the recent approach applied in [31]. Figure 2 shows the specific design of our asymmetric architecture which allocates more computational resource to the transmitter side than the receiver side to enable real-time decoding on mobile device.

#### 3.2 Channel-wise Quantization Aware Training

While using 32 bit floating-point operations is a default operating mode on wall-powered computing devices like a workstation or a server, it is often not efficient on power-constrained mobile devices. To enable efficient operation, we need to quantize the network parameters and activations to low-precision (e.g., 8 bit) integers. However, naive post-training quantization of neural network to 8 bit integers would significantly reduce the rate-distortion performance.

To alleviate this issue, we finetune our MobileCodec using the quantization-aware training (QAT) pipeline as described in Nagel et al. [27], which allows learning the quantization binwidth for each layer weight and activations by introducing simulated straight-through gradient estimation.

For the convolutional weights, we use separate quantization parameters for each output channel (known as per-channel quantization) [28], as we observe the range of both weight and activations vary widely between channels. This is in line with recent work [26] which shows that different channels of the latent can play impact the compression quality differently.

In addition, we pay particular attention to the variables that are important for entropy encoding. The scale \( \sigma \) in the hyperprior must cover several orders of magnitude, and would require very high precision if directly quantized. A better approach is to have the hyperprior networks learn an entropy coding parametrization suitable for low-precision representations. This enhanced entropy coding approach is described with more detail in [37]. For this demo, a simplified version of that approach was used, where the hyperencoder and hyperdecoder learn a logarithmic scale (i.e., they learn \( \ln(\sigma) \) instead of \( \sigma \)), which is then quantized to a single 8-bit integer in order to match the parallel entropy coding pre-computed tables (Section 3.3). The value of this 8-bit integer \( n \) that is fed to the entropy encoder and decoder is computed using:

\[
n = \max \left[ 0, \min \left[ 255, \lfloor \gamma \ln(\sigma) + \theta \rfloor \right] \right]
\]  

We set \( \gamma = 32 \) and \( \theta = 70 \) to cover the range of \( \sigma \) needed for the demo. Reference [37] presents more examples of alternative parametrizations.

#### 3.3 Parallel Entropy Coding

Entropy coding is employed to losslessly compress the quantized latents and hyperlatents to further reduce the rate. This is the stage where the bitstream representing the video is created (at
the transmitter) or parsed (at the receiver). For high resolution and quality levels, the entropy coding must support very high throughputs, which can be difficult and computationally expensive without parallelization, even with custom hardware.

Although the probability models in our MobileCodec are learned, the probability of each element in $y$ and $z$ is (conditionally) independent and thus can be coded in parallel. This was implemented by partitioning and indexing the bitstream, providing entry points for parallel decoding [36]. Figure 3 shows an illustration of this process. Concurrent encoders save data to temporary memory buffers, and after frame encoding is completed, those bitstreams are concatenated to create the combined bitstream for that frame. Note that this bitstream must include a header indicating starting positions for each independent part, which are used as entry points for parallel decoding.

The coding method is an implementation of static (instead of adaptive) arithmetic coding using 32-bit registers, with 16-bit values for probabilities and range, and byte-based renormalization [35]. Since it uses only simple arithmetic and logic operations, it can be executed with multiple threads, without specialized hardware.

The integer cumulative distribution arrays needed for arithmetic coding [35] are pre-computed, according to the quantized entropy coding parameter defined by eq. (9), assuming that the latent variables to be encoded have normal probability distributions, and are converted to integers using unit-step uniform quantization (i.e., simple rounding). Those arrays are stored at the transmitter and receiver.

Another important factor for entropy coding on a mobile device is the amount of memory needed for storing code tables. Throughput is maximized when coding tables stay loaded on small caches with the fastest memory. Fortunately, for NN-based codecs these tables are read-only (conventional codecs use adaptive coding requiring reading and writing), and the techniques and analysis developed for low-precision quantization in [37] could also be employed to further reduce table memory to a few kilobytes.

4 EXPERIMENTS

The goal of our experiments is to assess the ability of our neural codec to run in real-time on a mobile device while outputting high visual quality videos. To achieve this goal, we collected a set of off-the-self videos to showcase our video codec. Specifically, we
collected 13 videos of various categories at 4k resolution from www.pexels.com. These videos were then downsampled to a HD 720p resolution to reduce the artifacts from compression and to be used as input for MobileCodec. Figure 5 shows snapshots of the videos. Following the existing neural codec [1], our MobileCodec was first trained on Vimeo-90k dataset [43] for one million iterations to obtain a floating-point model. This model is then finetuned with quantization aware training for another 100 thousands iterations. The result is an efficient 8-bit quantized neural codec model.

4.1 Real-time HD video decoding

We tested the performance of our MobileCodec on these videos in a real-time decoding configuration. Figure 4 shows the setup of our experiments. Specifically, the input videos were encoded offline, stored as a bitstream, and then sent to a receiver which is a mobile phone powered by a Snapdragon 8 chip. The receiver then decoded the bitstream into video frames at real-time speed. Even though our demo was configured as an offline setup, the videos were encoded in low-delay mode which could also support live streaming mode in future work.

In Table 1, we show the quantitative measurement of our MobileCodec on each demo video. The results show that our Mobile codec can consistently run at faster than real-time speeds while obtaining satisfactory visual quality results. Specifically, our MobileCodec took 32ms to decode each frame, which is equivalent to 31 frames per second (FPS), while obtaining an average PSNR of 40.18dB.

4.2 Quantitative Evaluation

To further access the quality of our MobileCodec we also test its performance on the video compression dataset UVG [24] which is widely used for benchmarking video codecs. To the best of our knowledge, our MobileCodec is the first neural inter-frame video codec that can run real-time decoding on a mobile device. Thus, we compare MobileCodec’s performance with H.264 and H.265 which are among the most popular video codecs used in both regular and mobile platforms. Figure 6 shows the performance of our MobileCodec in comparison with the commonly used conventional codecs H264 and H265 in both ultrafast, and medium mode. The results show that our MobileCodec can obtain favorable results in comparison to the baseline even though MobileCodec is designed and executed on a mobile device while the ffmpeg implementation of H.264 and H.265 were used on a wall-powered workstation.

| Video          | MSSSIM | PSNR  | BPP | Frames | Time (ms) |
|----------------|--------|-------|-----|--------|-----------|
| Sports         | 0.9886 | 42.31 | 0.18| 300    | 30        |
| Animal         | 0.9891 | 39.55 | 0.35| 265    | 33        |
| Ocean          | 0.9842 | 40.21 | 0.35| 300    | 33        |
| Driving        | 0.9907 | 41.4  | 0.20| 173    | 30        |
| Driving        | 0.9904 | 40.56 | 0.31| 274    | 33        |
| Sports         | 0.9905 | 41.78 | 0.22| 189    | 31        |
| Horsebackriding| 0.9921 | 45.55 | 0.28| 300    | 31        |
| Barbequing     | 0.9872 | 38.08 | 0.36| 300    | 33        |
| Welding        | 0.9859 | 37.56 | 0.48| 300    | 33        |
| Bartending     | 0.9901 | 37.89 | 0.37| 300    | 32        |
| Food_Restaurant| 0.9915 | 39.16 | 0.35| 300    | 33        |
| Cutting        | 0.9910 | 42.62 | 0.13| 300    | 29        |
| Food_Restaurant| 0.9857 | 39.67 | 0.28| 300    | 33        |
| Average        | 0.9890 | 40.18 | 0.30| 277    | 32        |

5 CONCLUSION

This paper presents the design and implementation of MobileCodec, the first efficient neural inter-frame video codec running on a mobile device. Specifically, we introduce the design of an efficient neural inter-frame video compression architecture designed specifically for a mobile platform. We also present an approach to effectively deploy this video codec on a mobile device, including a quantization aware training step to quantize the model using 8-bit integer and a parallel entropy coding algorithm to effectively code the bitstream. The result is the first demo of real-time (≥30 FPS) decoding of HD 720p videos using a neural inter-frame video codec on a mobile device.
