OVERFITTING FOR FUN AND PROFIT: INSTANCE-ADAPTIVE DATA COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural data compression has been shown to outperform classical methods in terms of rate-distortion (RD) performance, with results still improving rapidly. At a high level, these methods are based on an autoencoder that tries to reconstruct the input instance from a (quantized) latent representation, coupled with a prior that is used to losslessly compress these latents. Due to limitations on model capacity and imperfect optimization and generalization, such models will suboptimally compress test data in general. However, one of the great strengths of learned compression is that if the test-time data distribution is known and relatively low-entropy (e.g. a camera watching a static scene, a dash cam in an autonomous car, etc.), the model can easily be finetuned or *adapted* to this distribution, leading to improved RD performance. In this paper we take this concept to the extreme, adapting the *full* model to a *single* video, and sending model updates (quantized and compressed using a parameter-space prior) along with the latent representation. Unlike previous work, we finetune not only the encoder/latents but the entire model, and - during finetuning - take into account both the effect of model quantization and the additional costs incurred by sending the model updates. We evaluate an image compression model on I-frames (sampled at 2 fps) from videos of the Xiph dataset, and demonstrate that full-model adaptation improves RD performance by ~ 1 dB, with respect to encoder-only finetuning.

1 INTRODUCTION

The most common approach to neural lossy compression is to train a variational autoencoder (VAE)-like model on a training dataset to minimize the *expected* RD cost $D + \beta R$ (Theis et al., 2017; Kingma & Welling, 2013). Although this approach has proven to be very successful (Ballé et al., 2018), a model trained to minimize expected RD cost over a full dataset is unlikely to be optimal for every test instance because the model has limited capacity, and both optimization and generalization will be imperfect. The problem of generalization will be especially significant when the testing distribution is different from the training distribution, as is likely to be the case in practice.

Suboptimality of the encoder has been studied extensively under the term *inference suboptimality* (Cremer et al., 2018), and it has been shown that finetuning the encoder or latents for a particular instance can lead to improved compression performance (Lu et al., 2020; Campos et al., 2019; Yang et al., 2020b; Guo et al., 2020). This approach is appealing as no additional information needs to be added to the bitstream, and nothing changes on the receiver side. Performance gains however are limited, because the prior and decoder can not be adapted.

In this paper we present a method for *full-model instance-adaptive compression*, i.e. adapting the *entire* model to a *single* data instance. Unlike previous work, our method takes into account the costs for sending not only the latent prior, but also the decoder model updates, as well as quantization of these updates. This is achieved by extending the typical RD loss with an additional model rate term M that measures the number of bits required to send the model updates under a newly introduced *model prior*, resulting in a combined RDM loss.

As an initial proof of concept, we show that this approach can lead to very substantial gains in RD performance (~ 1 dB PSNR gain at the same bit rate) on the problem of I-frame video coding, where a set of key frames, sampled from a video at 2 fps, are independently coded using an I-frame (image compression) model. Additionally, we show how the model rate bits are distributed across

the model, and (by means of an ablation study) quantify the individual gains achieved by including a model-rate loss and using quantization-aware finetuning.

The rest of this paper is structured as follows. Section 2 discusses the basics of neural compression and related work on adaptive compression. Section 3 presents our method, including details on the RDM loss, the choice of the model prior, its quantization, and the (de)coding procedure. In Sections 4 and 5 we present our experiments and results, followed by a discussion in Section 6.

2 Preliminaries and Related Work

2.1 Neural data compression

The standard approach to neural compression can be understood as a particular kind of VAE (Kingma & Welling, 2013). In the compression literature the encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is typically defined by a neural network parameterized by ϕ, with either deterministic output (so $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is one-hot) (Habibian et al., 2019) or with fixed uniform $[0, 1]$ noise on the outputs (Ballé et al., 2018). In both cases, sampling $z \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is used during training while quantization is used at test time.

The latent $\boldsymbol{z}$ is encoded to the bitstream using entropy coding in conjunction with a latent prior $p_\theta(\boldsymbol{z})$, so that coding $\boldsymbol{z}$ takes about $-\log p_\theta(\boldsymbol{z})$ bits (up to discretization). On the receiving side, the entropy decoder is used with the same prior $p_\theta(\boldsymbol{z})$ to decode $\boldsymbol{z}$ and then reconstruct $\hat{\boldsymbol{x}}$ using the decoder network $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ (note that we use the same symbol θ to denote the parameters of the prior and decoder jointly, as in our method both will have to be coded and added to the bitstream).

From these considerations it is clear that the rate R and distortion D can be measured by the two terms in the following loss:

$$\mathcal{L}_{\text{RD}}(\phi, \theta) = \beta \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[-\log p_\theta(\boldsymbol{z})\right]}_{R} + \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[-\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]}_{D}. \tag{1}$$

This loss is equal (up to the tradeoff parameter β and an additive constant) to the standard negative evidence lower bound (ELBO) used in VAE training. The rate term of ELBO is written as a KL divergence between encoder and prior, but since $D_{\text{KL}}(q, p) = R - H[q]$, and the encoder entropy $H[q]$ is constant in our case, minimizing the KL loss is equivalent to minimizing the rate loss.

Neural video compression is typically decomposed into the problem of independently compressing a set of key frames (i.e. I-frames) and conditionally compressing the remaining frames (Lu et al., 2019; Liu et al., 2020; Wu et al., 2018; Djelouah et al., 2019; Yang et al., 2020a). In this work, we specifically focus on improving I-frame compression.

2.2 Adaptive Compression

A compression model is trained on a dataset $\mathcal{D}$ with the aim of achieving optimal RD performance on test data. However, because of limited model capacity, optimization difficulties, or insufficient data (resulting in poor generalization), the model will in general not achieve this goal. When the test data distribution differs from that of the training data, generalization will not be guaranteed even in the limit of infinite data and model capacity, and perfect optimization.

A convenient feature of neural compression however is that a model can easily be finetuned on new data or data from a specific domain. A model can for instance (further) be trained after deployment, and Habibian et al. (2019) showed improved RD gains after finetuning a video compression model on footage from a dash cam, an approach dubbed *adaptive compression* (Habibian et al., 2019).

In adaptive compression, decoding requires access to the adapted prior and decoder models. These models (or their delta relative to a pretrained shared model) thus need to be signalled. When the amount of data coded with the adapted model is large, the cost of signaling the model update will be negligible as it is amortized. However, a tradeoff exists, as the more restricted the domain of adaptation, the more we can expect to gain from adaptation (e.g. an image compared to a video or collection of videos). In this paper we consider the case where the domain of adaptation is a set of I-frames from a single video, resulting in costs for sending model updates which become very relevant.

2.3 CLOSING THE AMORTIZATION GAP

Coding model updates can easily become prohibitively expensive when the model is adapted for every instance. However, if we only adapt the encoder or latents, no model update needs to be added to the bitstream as the encoder is not needed for decoding as the latents are sent anyway. We can thus close, or at least reduce, the *amortization gap* (the difference between $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and the optimal encoder; Cremer et al. (2018)) without paying any bits for model updates. Various authors have investigated this approach: Aytekin et al. (2018); Lu et al. (2020) adapt the encoder, while Campos et al. (2019); Yang et al. (2020b); Guo et al. (2020) adapt the latents directly. This simple approach was shown to provide a modest boost in RD performance.

2.4 ENCODING MODEL UPDATES

As mentioned, when adapting (parts of) the decoder or prior to an instance, model updates have to be added to the bitstream in order to enable decoding. Recent works have proposed ways to finetune parts of the model, while keeping the resulting bitrate overhead small. For instance Klopp et al. (2020) train a reconstruction error predicting network at encoding time, quantize its parameters, and add them to the bitstream. Similarly (Lam et al., 2019; 2020) propose to finetune all parameters or only the convolutional biases, respectively, of an artifact removal filter that operates after decoding. A sparsity-enforcing and magnitude-suppressing penalty is leveraged, and additional thresholding is applied to even more strictly enforce sparsity. The update vector is thereafter quantized using k-means clustering. Finally, Zou et al. (2020) finetune the latents in a meta-learning framework, in addition to updating the decoder convolutional biases, which are quantized by k-means and thereafter transmitted. All these methods perform quantization post-training, leading to a potentially unbounded reduction in performance. Also, albeit the use of regularizing loss terms, no valid proxy for the actual cost of sending model updates is adopted. Finally, none of these methods performs adaptation of the full model.

The field of model compression is related to our work as the main question to be answered is how to most efficiently compress a neural network without compromising on downstream task performance (Han et al., 2016; Kuzmin et al., 2019). Bayesian compression is closely related, where the model weights are sent under a model prior (Louizos et al., 2017; Havasi et al., 2018) as is the case in our method. Instead of modeling uncertainty in parameters, we however assume a deterministic posterior (i.e. point estimate). Another key difference with these works is that we send the model parameter updates relative to an existing baseline model, which enables extreme compression rates (0.02-0.2 bits/param). This concept of compressing updates has been used before in the context of federated learning (McMahan et al., 2017; Alistarh et al., 2017) as well. We distinguish ourselves from that context, as there the model has to be compressed during every iteration, allowing for error corrections in later iterations. We only transmit the model updates once for every data instance that we finetune on.

3 FULL-MODEL INSTANCE-ADAPTIVE COMPRESSION

In this section we present full-model finetuning on one instance, while (during finetuning) taking into account both model quantization and the costs for sending model updates. The main idea is described in Section 3.1, after which Section 3.2 and 3.3 respectively provide details regarding the model prior and its quantization. The algorithm is described in Section 3.4.

3.1 FINETUNING AT INFERENCE TIME

Full-model instance-adaptive compression entails finetuning of a set of *global model* parameters $\{\phi_{\mathcal{D}}, \theta_{\mathcal{D}}\}$ (obtained by training on dataset $\mathcal{D}$) on a single instance $\boldsymbol{x}$. This results in updated parameters ϕ, θ, of which only θ has to be signalled in the bitstream. In practice we only learn the changes with respect to the global model, and encode the *model updates* $\delta = \theta - \theta_{\mathcal{D}}$ of the decoding model. In order to encode δ, we introduce a continuous model prior $p(\delta)$ to regularize these updates, and use the quantized counterpart $p[\bar{\delta}]$ for entropy (de)coding them (more on quantization in Section 3.3).

The overhead for sending quantized model update $\bar{\delta}$ is given by model rate $\overline{M} = -\log p[\bar{\delta}]$, and can be approximated by its continuous counterpart $M = -\log p(\delta)$ (see Appendix A.1 for justifi-

cation). Adding this term to the standard RD loss using the same tradeoff parameter β, we obtain the instance-adaptive compression objective:

$$\mathcal{L}_{\text{RDM}}(\phi, \delta) = \mathcal{L}_{\text{RD}}(\phi, \theta_{\mathcal{D}} + \bar{\delta}) - \beta \underbrace{\log p(\delta)}_{M}. \tag{2}$$

At inference time, this objective can be minimized directly to find the optimal model parameters for transmitting datapoint $\boldsymbol{x}$. It takes into account the additional costs for encoding model updates (M term), and incorporates model quantization during finetuning ($\mathcal{L}_{\text{RD}}$ evaluated at $\bar{\theta} = \theta_{\mathcal{D}} + \bar{\delta}$).

3.2 Model prior design

A plethora of options exist for designing model prior $p(\delta)$ as any probability density function (PDF) could be chosen. A natural choice for modeling parameter update $\delta = \theta - \theta_{\mathcal{D}}$ is to leverage a Gaussian distribution, centered around the zero-update. Specifically, we can define the model prior on the updates as a multivariate zero-centered Gaussian with zero covariance, and a single shared (hyperparameter) σ, denoting the standard deviation: $p(\delta) = \mathcal{N}(\delta \,|\, \mathbf{0}, \sigma \boldsymbol{I})$. Note that this is equivalent to modeling θ by $p(\theta) = \mathcal{N}(\theta \,|\, \theta_{\mathcal{D}}, \sigma \boldsymbol{I})$.

When entropy (de)coding the quantized updates $\bar{\delta}$ under $p[\bar{\delta}]$, we must realize that even the zero-update, i.e. $\bar{\delta} = \mathbf{0}$, is not for free. We define these initial static costs as $\overline{M}_0 = -\log p[\bar{\delta} = \mathbf{0}]$. Because the mode of the defined model prior is zero, these initial costs $\overline{M}_0$ equal the minimum costs. Minimization of eq. (2) thus ensures that – after overcoming these static costs – any extra bit spent on model updates will be accompanied by a commensurate improvement in RD performance.

Since our method works best when signalling the zero-update is cheap, we want to increase the probability mass $p[\bar{\delta}] = \mathbf{0}$. We propose to generalize our earlier proposed model prior to a so-called spike-and-slab prior (Ročková & George, 2018), which drastically reduces the costs for the zero-update. More specifically, we redefine the PDF as a (weighted) sum of two Gaussians – a wide (slab) and a more narrow (spike) distribution:

$$\begin{aligned} p(\delta) &= \frac{p_{\text{slab}}(\delta) + \alpha \, p_{\text{spike}}(\delta)}{1 + \alpha}, \quad \text{with} \\ p_{\text{slab}}(\delta) &= \mathcal{N}(\delta | \mathbf{0}, \sigma \boldsymbol{I}) \quad \text{and} \quad p_{\text{spike}}(\delta) = \mathcal{N}(\delta | \mathbf{0}, \frac{t}{6} \boldsymbol{I}), \end{aligned} \tag{3}$$

where $\alpha \in \mathbb{R}_{\geq 0}$ is a hyperparameter determining the height of the spiky Gaussian with respect to the wider slab, t is the the bin width used for quantization (more details in Section 3.3), and $\sigma >> \frac{t}{6}$. By choosing the standard deviation of the spike to be $\frac{t}{6}$, the mass within six standard deviations (i.e. 99.7% of the total mass) is included in the central quantization bin after quantization. Note that the (slab-only) Gaussian prior is a special case of the spike-and-slab Gaussian prior in eq. (3), where $\alpha = 0$. As such, $p(\delta)$ refers to the spike-and-slab prior in the rest of this work.

Adding the spike distribution, not only decreases $\overline{M}_0$, it also more heavily enforces sparsity on the updates via regularizing term M in eq. (2). In fact, a high spike (i.e. large α) can make the bits for signalling a zero-update so small (i.e. almost negligible), that the model effectively learns to make a binary choice; A parameter is either worth updating at the cost of some additional rate, or its not updated and the 'spent' bits are negligible.

3.3 Quantization

In order to quantize a scalar δ_i (denoted by δ in this section to avoid clutter), we use N equispaced bins of width t, and we define the following quantization function:

$$\bar{\delta} = Q_t(\delta) = \text{clip}\left(\left\lfloor \frac{\delta}{t} \right\rceil \cdot t, \ \min = -\frac{(N-1)t}{2}, \ \max = \frac{(N-1)t}{2} \right). \tag{4}$$

As both rounding and clipping are non-differentiable, the gradient of Q_t is approximated using the Straight-Through estimator (STE), proposed by Bengio et al. (2013). That is, we assume $\partial Q_t(\delta)/\partial \delta = 1$.

The bins are intervals $Q_t^{-1}(\bar{\delta}) = [\bar{\delta}-t/2, \bar{\delta}+t/2]$. We view t as a hyperparameter and define N to be the smallest integer such that the region covered by bins (i.e. the interval $[-(N-1)t/2, (N-1)t/2]$), covers at least $1 - 2^{-8}$ of the probability mass of $p(\delta)$. Indeed the number of bins N is proportional to the ratio of the width of $p(\delta)$ and the width of the bins: $N \propto \sigma/t$. The number of bins presents a tradeoff between finetuning flexibility and model rate costs $\overline{M}$, so the σ/t-ratio is an important hyperparameter. The higher N, the higher these costs due to finer quantization, but simultaneously the lower the quantization gap, enabling more flexible finetuning.

Since $\bar{\delta} = Q_t(\delta)$, the discrete model prior $p[\bar{\delta}]$ is the pushforward of $p(\delta)$ through Q_t:

$$p[\bar{\delta}] = \int_{Q^{-1}(\bar{\delta})} p(\delta)d\delta = \int_{\bar{\delta}-t/2}^{\bar{\delta}+t/2} p(\delta)d\delta = P(\delta < \bar{\delta} + t/2) - P(\delta < \bar{\delta} - t/2). \tag{5}$$

That is, $p[\bar{\delta}]$ equals the mass of $p(\delta)$ in the bin of $\bar{\delta}$, which can be computed as the difference of the cumulative density function (CDF) of $p(\delta)$ evaluated at the edges of that bin.

3.4 Entropy coding and decoding

After finetuning of the compression model on instance $\boldsymbol{x}$ by minimizing the loss in eq. (2), both the latents and the model updates are entropy coded (under their respective priors $p_{\bar{\theta}}(\boldsymbol{z})$ and $p[\bar{\delta}]$) into a bitstream $\boldsymbol{b}$. Decoding starts by decoding $\bar{\delta}$ using $p[\bar{\delta}]$, followed by decoding $\boldsymbol{z}$ using $p_{\bar{\theta}}(\boldsymbol{z})$ (where $\bar{\theta} = \bar{\delta} + \theta_{\mathcal{D}}$), and finally reconstructing $\hat{\boldsymbol{x}}$ using $p_{\bar{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. The whole process is shown in Figure 1 and defined formally in Algorithms 1 and 2.

4 Experimental setup

4.1 Datasets

The experiments in this paper use images and videos from the following two datasets:

CLIC19 [1] The CLIC19 dataset contains a collection of natural high resolution images. It is conventionally divided into a professional set, and a set of mobile phone images. Here we merge the existing training folds of both sets and use the resulting dataset $\mathcal{D}$ to train our I-frame model. The corresponding validation folds are used to validate the global model performance.

Xiph-5N 2fps [2] The Xiph dataset contains a variety of videos of different formats. We select a representative sample of five videos (*5N*) from the set of 1080p videos (see Appendix B.3 for details regarding selection of these samples). Each video is temporally subsampled to 2 fps to create a dataset of I-frames, referred to as Xiph-5N 2fps. Frames in all videos contain 1920×1080 pixels, and the set of I-frames after subsampling to 2 fps contain between 20 and 42 frames. The five selected videos are single-scene but multi-shot, and come from a variety of sources. Xiph-5N 2fps is used to validate our instance-adaptive data compression framework.

4.2 Global model architecture and training

Model rate can be restricted by (among others) choosing a low-complexity neural compression model, and amortizing the additional model update costs over a large number of pixels. The most natural case for full-model adaptation is therefore to finetune a low-complexity model on a video instance. Typical video compression setups combine an image compression (I-frame) model to compress key frames, and a predict- or between-frame model to reconstruct the remaining frames. Without loss of generality for video-adaptive finetuning, we showcase our full-model finetuning framework for the I-frame compression problem, being a subproblem of video compression. Specifically, we use the (relatively low-complexity) hyperprior-model proposed by Ballé et al. (2018), including the mean-scale prior (without context) from Minnen et al. (2018). Before finetuning, this model is trained on the training fold of the CLIC19 dataset, using the RD objective given in eq. (1). Appendix C provides more details on both its architecture and the adopted training procedure.

[1] `https://www.compression.cc/2019/challenge/`

[2] `https://media.xiph.org/video/derf/`

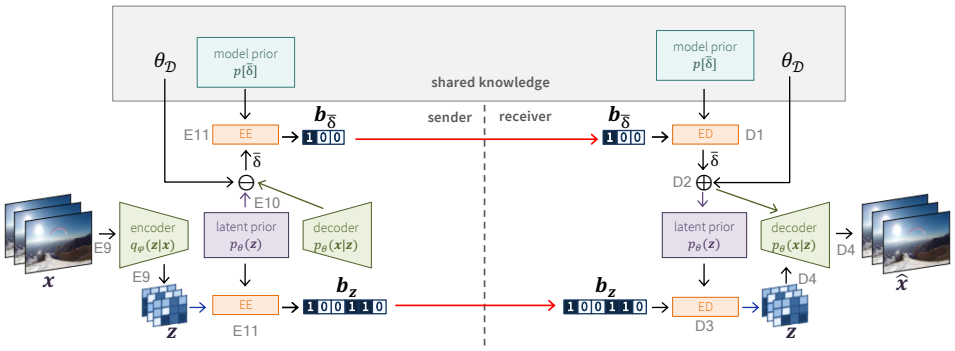


Figure 1: Visualization of encoding (Algorithm 1) and decoding (Algorithm 2) of our full-model instance-adaptive method. Each step is denoted with a code, e.g. *E9*, which refers to line 9 of the encoding algorithm. EE and ED denote entropy encoding and decoding, respectively. Both the compact latent space $\boldsymbol{z}$ and the parameter updates in $\bar{\delta}$ are encoded in their respective bitstreams $\boldsymbol{b_z}$ and $\boldsymbol{b}_{\bar{\delta}}$. Model prior $p[\bar{\delta}]$ entropy decodes $\boldsymbol{b}_{\bar{\delta}}$, after which the latent prior can decode $\boldsymbol{b_z}$.

Algorithm 1 Encoding of $\boldsymbol{x}$

Input: Global model parameters $\{\phi_{\mathcal{D}}, \theta_{\mathcal{D}}\}$ pretrained on training set $\mathcal{D}$, model parameter quantizer Q_t, model prior $p[\bar{\delta}]$, and datapoint to be compressed $\boldsymbol{x}$.
Output: Compressed bitstream $\boldsymbol{b}$.

1: Initialize model parameters: $\phi = \phi_{\mathcal{D}}$, and $\theta = \theta_{\mathcal{D}}$
2: **for** step in MAX STEPS **do**
3: Sample one frame: $\boldsymbol{x}_f \sim \boldsymbol{x}$
4: Quantize transmittable parameters: $\bar{\theta} \leftarrow Q_t(\delta) + \theta_{\mathcal{D}}$, with $\delta = \theta - \theta_{\mathcal{D}}$
5: Forward pass: $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x}_f)$ and evaluate $p_{\bar{\theta}}(\boldsymbol{x}_f \mid \boldsymbol{z})$ and $p_{\bar{\theta}}(\boldsymbol{z})$.
6: Compute loss $\mathcal{L}_{RDM}(\phi, \delta)$ on $\boldsymbol{x}_f$ according to eq. (2).
7: Backpropagate using STE for Q_t, then update ϕ, θ using gradients $\dfrac{\partial \mathcal{L}_{RDM}}{\partial \phi}$ and $\dfrac{\partial \mathcal{L}_{RDM}}{\partial \theta}$.
8: **end for**
9: Compress $\boldsymbol{x}$ to $\boldsymbol{z} \sim q_\phi(\boldsymbol{x})$.
10: Compute quantized model parameters: $\bar{\theta} = \theta_{\mathcal{D}} + \bar{\delta}$, with $\bar{\delta} = Q_t(\theta - \theta_{\mathcal{D}})$.
11: Entropy encode: $\boldsymbol{b}_{\bar{\delta}} = \mathrm{enc}(\bar{\delta}; p[\bar{\delta}])$ and $\boldsymbol{b_z} = \mathrm{enc}(\boldsymbol{z}; p_{\bar{\theta}}(\boldsymbol{z}))$.
12: Concatenate bitstreams: $\boldsymbol{b} = (\boldsymbol{b}_{\bar{\delta}}, \boldsymbol{b_z})$.

Algorithm 2 Decoding of $\boldsymbol{x}$

Input: Model prior $p[\bar{\delta}]$, bitstream $\boldsymbol{b} = (\boldsymbol{b}_{\bar{\delta}}, \boldsymbol{b_z})$, global model parameters $\theta_{\mathcal{D}}$, trained on training set $\mathcal{D}$.
Output: Decoded datapoint $\hat{\boldsymbol{x}}$

1: Entropy decode: $\bar{\delta} = \mathrm{dec}(\boldsymbol{b}_{\bar{\delta}}; p[\bar{\delta}])$.
2: Compute updated parameters: $\bar{\theta} = \theta_{\mathcal{D}} + \bar{\delta}$.
3: Entropy decode latent under finetuned prior: $\boldsymbol{z} = \mathrm{dec}(\boldsymbol{b_z}; p_{\bar{\theta}}(\boldsymbol{z}))$.
4: Decode instance as the mean of the finetuned decoder: $p_{\bar{\theta}}(\boldsymbol{x}|\boldsymbol{z})$

4.3 Instance-adaptive finetuning

Each instance in the Xiph-5N 2fps dataset (i.e. a set of I-frames belonging to one video) is (separately) used for full-model adaptation. The resulting model rate costs are amortized over the set of I-frames, rather than the full video. As a benchmark, we only finetune the encoding procedure, as it does not induce additional bits in the bitstream. Encoder-only finetuning can be implemented by either finetuning the encoding model parameters ϕ, or directly optimizing the latents as in Campos

et al. (2019). We implement both benchmarks, as the former is an ablated version of our proposed full-model finetuning, whereas the latter is expected to perform better due to the amortization gap Cremer et al. (2018).

Global models trained with RD tradeoff parameter $\beta \in \{3e{-}3, 1e{-}3, 2.5e{-}4, 1e{-}4\}$ are used to initialize the models that are then being finetuned with the corresonding value for β. Both encoder-only and direct-latent finetuning minimize the RD loss as given in eq. (1). For encoder-only tuning we use a constant learning rate of $1e{-}6$ whereas for latent optimization, a learning rate of $5e{-}4$ is used for the low bit rate region (i.e. two highest β values), and $1e{-}3$ for the high rate region. In case of direct latent optimization, the pre-quantized latents are finetuned, which are initialized using a forward pass of the encoder.

Our instance-adaptive *full-model* finetuning framework extends encoder-only finetuning, by jointly updating ϕ and θ using the RDM loss from eq. (2). In this case we finetune the global model that is trained with $\beta = 0.001$, independent of the value of β during finetuning. Empirically, this resulted in negligible difference in performance compared to using the global model of the corresponding finetuning β, while it alleviated memory constraints thanks to the smaller size of this low bit rate model architecture (see Appendix C). The training objective in eq. (2) is expressed in bits per pixel and optimized using a fixed learning rate of $1e{-}4$. We set up the model prior as follows: quantization bin width $t = 0.005$, standard deviation $\sigma = 0.05$, and the multiplicative factor of the spike $\alpha = 1000$. We empirically found that sensitivity to changing α in the range 50-5000 was low. Realize that, instead of empirically setting α, its value could also be solved for a target initial cost $\overline{M}_0$, given the number of decoding parameters and pixels in the instance.

All finetuning experiments (both for encoding-only and *full-model*) ran for 100k steps, each containing one minibatch of a single, full resolution I-frame[3], we select the best model based on the loss over all I-frames. We used the Adam optimizer with its default settings (Kingma & Ba, 2014).

5 RESULTS

5.1 RATE-DISTORTION GAINS

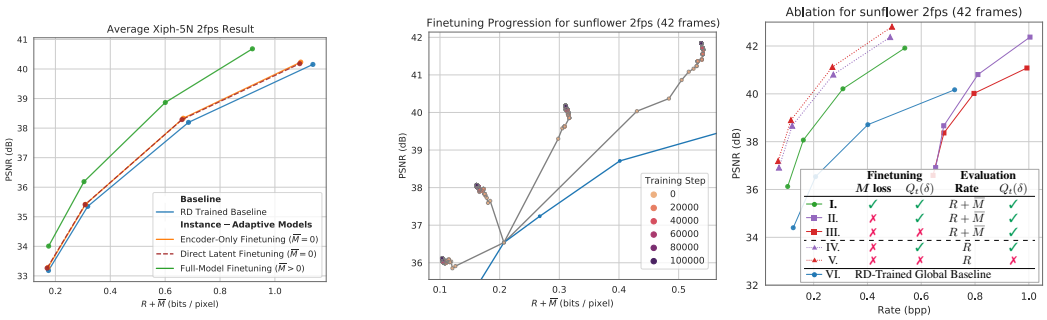
Figure 2a shows the compression performance for encoder-only finetuning, direct latent optimization, and full-model finetuning, averaged over all videos in the Xiph-5N 2fps dataset for different rate-distortion tradeoffs. Finetuning of the entire parameter-space results in much higher RD gains (on average approximately 1 dB for the same bit rate) compared to only finetuning the encoding parameters ϕ or the latents directly. Figure 7 in Appendix D shows this plot for each video separately. Note that encoder-only finetuning performance is on par with direct latent optimization, implying that the amortization gap (Cremer et al., 2018) is close to zero when finetuning the encoder model on a moderate number of I-frames.

Table 1 provides insight in the distribution of bits over latent rate R and model rate $\overline{M}$, which both increase for lower values of β. However, the relative contribution of the model rate increases for the higher bit rate regime, which could be explained by the fact that in this regime the latent rate can more heavily be reduced by finetuning. Figure 2a indeed confirms that the the total rate reduction is higher for the high bit rate regime, which thus fully originates from the reduction in latent rate after finetuning. Table 1 also shows that the static intial costs $\overline{M}_0$ only marginally contribute to the total model rate. Figure 2b shows for one video how finetuning progressed over training steps, confirming that the compression gains already at the beginning of finetuning cover these initial costs. This effect was visible for all tested videos (see Appendix D).

5.2 ABLATIONS

Figure 2c shows several ablation results for one video. Case I is our proposed full-model finetuning, optimizing the RDM loss, while simultaneously quantizing the updates to compute distortion (denoted with $Q_t(\delta)$). One can see that both removing the (continuous) model rate penalty from

[3] All frames in the videos in Xiph-5N 2fps are of spatial resolution 1920×1080. In order to make this shape compatible with the strided convolutions in our model, we pad each frame to 1920×1088 before encoding. After reconstructing $\hat{\boldsymbol{x}}$, it is cropped back to its original resolution for evaluation.



(a) (b) (c)

Figure 2: RD performance for four different rate-distortion tradeoffs with $\beta = 3e{-}3$, $1e{-}3$, $2.5e{-}4$, and $1e{-}4$ (from left to right). (a) Averaged performance over all videos of the Xiph-5N 2fps dataset. Our *full-model* finetuning outperforms *encoder-only* and *direct latent* optimization with approximately 1 dB gain for the same rate. (b) Finetuning progression of the `sunflower` video over time. Between each dot, 500 training steps are taken, showing that already at the start of finetuning large RD gains are achieved and the RD performance continues to improve during finetuning. (c) Ablation where we show the effect of both quantization- ($Q_t(\delta)$) and model rate aware (M Loss) finetuning. Case V shows the upper bound on achievable finetuning performance when (naively) not taking into account quantization and model update rate.

the finetuning procedure (case II), and additionally removing quantization-aware training (case III) heavily deteriorates the compression gains. The difference between II and III quantifies the quantization gap, which we cover by quantization-aware finetuning. In both case II and III the additional model rate costs were not penalized during finetuning, resulting in a total rate even higher than the baseline (non-finetuned) model. Analyzing these runs while (naively) not taking into account the additional model update costs (cases IV and V) provides upper bounds on compression improvement in the RDM framework. The most naive bound is given by RD finetuning without parameter quantzation (V), and a tighter bound is given by RD finetuning with quantization (IV). We see that the gap is small, suggesting we don't lose much performance in our quantization. The gap between this bound (IV) and our method (I) is larger, and might be decreased by improving the model prior.

5.3 Distribution of bits

Figure 3 shows for different (mutually exclusive) parameter groups the distribution of the model updates δ (top) and their corresponding bits (bottom). Interestingly one can see how for (almost) all groups, the updates are being clipped by our earlier defined quantizer Q_t. This suggests the need for large, expensive updates in these parameter groups, for which the additional RD gain thus appears to outweigh the extra costs. At the same time, all groups show an elevated center bin, thanks to training with the spike-and-slab prior. By design of this prior, the bits paid for this zero-update are extremely low, which can best be seen in the bits histogram (Fig. 3-bottom) of the *Codec Decoder Weight* and *Biases*. The parameter updates of the *Codec Decoder IGND* group are the only ones that are non-symmetrically distributed across the zero-updated, which can be explained by the fact

β	PSNR (dB)	$R + \overline{M}$	R (bits/pixel)	$\overline{M}$ (bits/pixel)	$\overline{M}_0$	$\overline{M}$ (bits/parameter)	$\overline{M}$ (kB/frame)
3.0e-3	34.0	0.175	0.174	0.001	0.00033	0.025	0.38
1.0e-3	36.2	0.304	0.302	0.002	0.00033	0.041	0.63
2.5e-4	39.0	0.551	0.545	0.006	0.00033	0.106	1.65
1.0e-4	40.8	0.866	0.852	0.014	0.00033	0.229	3.57

Table 1: Distribution of bit rate for different rate-distortion tradeoffs β, averaged over the videos in the Xiph-5N 2fps dataset. The number of bits are distributed over the latent rate R and the model rate $\overline{M}$, which is computed using the quantized model prior $p[\bar{\delta}]$.

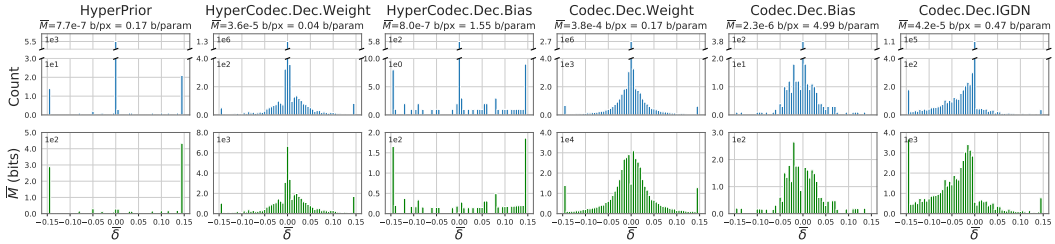


Figure 3: Empirical distribution of parameter updates for the model finetuned on the `sunflower` video with $\beta = 2.5e{-}4$. Columns denote parameter groups. Top: Histograms of the model updates $\overline{\delta}$. Bottom: Histogram of bit allocation for $\overline{\delta}$. Subtitles indicate the total number of bits for each parameter group, both expressed in bits per pixel (b/px) and bits per parameter (b/param).

that IGDN (Ballé et al., 2016) is an (inverse) normalization layer. The *Codec Decoder Weights* were found to contribute most to the total model rate M.

6 Discussion

In this work we presented instance-adaptive neural compression, the first method that enables finetuning of a full compression model on a set of I-frames from a single video, while restricting the additional bits for encoding the (quantized) model updates. To this end, we proposed an extension of the typical rate-distortion loss by incorporating both model quantization and the additional model rate costs during finetunig. This loss guarantees pure RD performance gains after overcoming a small initial cost for encoding the zero-update. We showed improved RD performance on all five tested videos from the Xiph dataset, with an average distortion improvement of approximately 1 dB for the same bit rate.

Among videos, we found a difference in achieved finetuning RD gain (see Appendix D). Possible causes can be three-fold. First, the performance of the global model for each video differs, therewith influencing the maximum gains to be achieved by finetuning. Second, video characteristics such as (non-)stationarity greatly influence the diversity of the set of I-frames, thereby affecting the ease of model-adaption. Thirdly, the number of I-frames differs per video and thus trades off model update costs (which are amortized over the set of I-frames), with diversity (and therewith ease of finetuning).

The results of the ablation study in Fig. 2c show that the quantization gap (IV vs V) is considerably smaller than the performance deterioration due to (additionally) regularizing the finetuning using the model prior (I vs IV). From this, we can conclude that most improvement in future work can be gained by leveraging more flexible model priors, for example by learning the model prior's parameters and/or modelling dependencies among updates.

We showed how instance-adaptive full-model finetuning introduced considerable gains for the subproblem of I-frame compression in video compression. Equivalently, one can exploit full-model finetuning to enhance compression performance of the remaining (non-key) frames of a video, improving the total video compression performance even further. Also, neural video compression models that exploit temporal redundancy directly, could benefit from full-model RDM finetuning, as long as the model's complexity is low enough to restrict model rate. Leveraging such a low-complexity video model moves computational complexity of data compression from the receiver side to the sender side by heavily finetuning this small model on each datapoint. We foresee the convenience of this computational shift in applications where bandwidth and receiver compute power is scarce, but encoding compute time is less important. This use case in practice happens, for example, for (non-live) video streaming to low-power edge devices. Finding such low-complexity video compression models is however not trivial as it's capacity must still be enough to compress an entire video. We foresee great opportunities here, and will therefore investigate this use case in future work.

REFERENCES

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

Caglar Aytekin, Xingyang Ni, Francesco Cricri, Jani Lainema, Emre Aksu, and Miska M Hannuksela. Block-optimized variable bit rate neural image compression. In *CVPR Workshops*, pp. 2551–2554, 2018.

Johannes Ballé, Valero Laparra, and Eero Simoncelli. Density modeling of images using a generalized normalization transformation. In *4th International Conference on Learning Representations, ICLR 2016*, 2016.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

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

Joaquim Campos, Simon Meierhans, Abdelaziz Djelouah, and Christopher Schroers. Content adaptive optimization for neural image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*, pp. 1078–1086, 2018.

Abdelaziz Djelouah, Joaquim Campos, Simone Schaub-Meyer, and Christopher Schroers. Neural inter-frame compression for video coding. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6421–6429, 2019.

Tiansheng Guo, Jing Wang, Ze Cui, Yihui Feng, Yunying Ge, and Bo Bai. Variable rate image compression with content adaptive optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 122–123, 2020.

Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7033–7042. openaccess.thecvf.com, 2019.

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*, 2016.

Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. In *International Conference on Learning Representations*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2014.

Diederik P Kingma and Max Welling. Auto-Encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Jan P Klopp, Liang-Gee Chen, and Shao-Yi Chien. Utilising low complexity cnns to lift non-local redundancies in video coding. *IEEE Transactions on Image Processing*, 2020.

Andrey Kuzmin, Markus Nagel, Saurabh Pitre, Sandeep Pendyam, Tijmen Blankevoort, and Max Welling. Taxonomy and evaluation of structured compression of convolutional neural networks. *arXiv preprint arXiv:1912.09802*, 2019.

Yat-Hong Lam, Alireza Zare, Caglar Aytekin, Francesco Cricri, Jani Lainema, Emre Aksu, and Miska Hannuksela. Compressing weight-updates for image artifacts removal neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Yat-Hong Lam, Alireza Zare, Francesco Cricri, Jani Lainema, and Miska M Hannuksela. Efficient adaptation of neural network filter for video compression. In *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 358–366, 2020.

Haojie Liu, Han Shen, Lichao Huang, Ming Lu, Tong Chen, and Zhan Ma. Learned video compression via joint spatial-temporal correlation exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 11580–11587, 2020.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in neural information processing systems*, pp. 3288–3298, 2017.

Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Guo Lu, Chunlei Cai, Xiaoyun Zhang, Li Chen, Wanli Ouyang, Dong Xu, and Zhiyong Gao. Content adaptive and error propagation aware deep video compression. *arXiv preprint arXiv:2003.11282*, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems*, pp. 10771–10780, 2018.

Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G Anderson, and Lubomir Bourdev. Learned video compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3454–3463, 2019.

Veronika Ročková and Edward I George. The spike-and-slab lasso. *Journal of the American Statistical Association*, 113(521):431–444, 2018.

L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/pdf?id=rJiNwv9gg`.

Chao-Yuan Wu, Nayan Singhal, and Philipp Krahenbuhl. Video compression through image interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 416–431, 2018.

Yang Yang, Guillaume Sautière, J Jon Ryu, and Taco S Cohen. Feedback recurrent autoencoder. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3347–3351. IEEE, 2020a.

Yibo Yang, Robert Bamler, and Stephan Mandt. Improving inference for neural image compression. *Advances in Neural Information Processing Systems*, 33, 2020b.

Nannan Zou, Honglei Zhang, Francesco Cricri, Hamed R Tavakoli, Jani Lainema, Miska Hannuksela, Emre Aksu, and Esa Rahtu. L^2C–learning to learn to compress. *arXiv preprint arXiv:2007.16054*, 2020.

A MODEL RATE LOSS

A.1 GRADIENT DEFINITION

The proof below shows that $\nabla_\delta \overline{M}$ is an unbiased first-order approximation of $\nabla_\delta M$, validating its use during finetuning.

The gradient of the continuous model rate loss M towards δ is defined as:

$$\nabla_\delta M = -\nabla_\delta \log p(\delta) = -\frac{\nabla_\delta p(\delta)}{p(\delta)} \tag{6}$$

The gradient of the non-differentiable discrete model rate loss $\bar{M}$ towards δ can be defined by exploiting the Straight-Through gradient estimator (Bengio et al., 2013), i.e. $\nabla_\delta \bar{\delta} = 1$.

As such[4]:

$$\begin{aligned}\nabla_\delta \overline{M} = -\nabla_\delta \log p[\bar{\delta}] &= -\nabla_\delta \log[P(\delta < \bar{\delta} + \frac{t}{2}) - P(\delta < \bar{\delta} - \frac{t}{2})] \\ &= -\frac{p(\bar{\delta} + \frac{t}{2}) - p(\bar{\delta} - \frac{t}{2})}{P(\delta < \bar{\delta} + \frac{t}{2}) - P(\delta < \bar{\delta} - \frac{t}{2})}.\end{aligned} \tag{7}$$

By first-order approximation we can write:

$$\nabla_\delta p(\bar{\delta}) \approx \frac{p(\bar{\delta} + \frac{t}{2}) - p(\bar{\delta} - \frac{t}{2})}{t} \tag{8}$$

and

$$P(\delta < \bar{\delta} + \frac{t}{2}) - P(\delta < \bar{\delta} - \frac{t}{2}) \approx p(\bar{\delta})t. \tag{9}$$

Using eq. (8) and eq. (9), we can express the gradient of the discrete model rate costs $\nabla_\delta \bar{M}$ as:

$$\nabla_\delta \overline{M} \approx -\frac{\nabla_\delta p(\bar{\delta})t}{p(\bar{\delta})t} = -\frac{\nabla_\delta p(\bar{\delta})}{p(\bar{\delta})}, \tag{10}$$

which is thus a first-order approximation of $\nabla_\delta M$.

A.2 CONTINUOUS VS DISCRETE MODEL RATE PENALTY

The proof provided in the previous section is not restricted to specific designs for $p(\delta)$, and thus holds both for the spike-and-slab prior including a spike ($\alpha > 0$), or the special case where no spike is used ($\alpha = 0$).

Figure 4 shows quantization of the model updates (top-left) and its corresponding gradient (left-bottom), exploiting the Straight-Through gradient estimator Bengio et al. (2013). Also the discrete (true) model rate $\overline{M}$ and its continuous analogy M (middle-top) with their corresponding gradients (middle-bottom) are shown for the special case of not using a spike, i.e. $\alpha = 0$. One can see that the continuous model rate proxy is a shifted version of the discrete costs. Since such a translation does not influence the gradient (and neither gradient-based optimization), the continuous model rate loss can be used during finetuning, preventing instable training thanks to its smooth gradient.

[4]Due to cutting (a maximum) of 2^{-8} mass from the tails of $p(\delta)$ to enable quantization, the difference in cumulative masses in eq. (7) should be renormalized by $1 - 2^{-8}$. As this is a constant division inside a $\log$, it results in a subtraction of $\nabla_\delta \log 1 - 2^{-8} = 0$. Since this normalization does not influence the gradient, we omitted it for the sake of clarity.

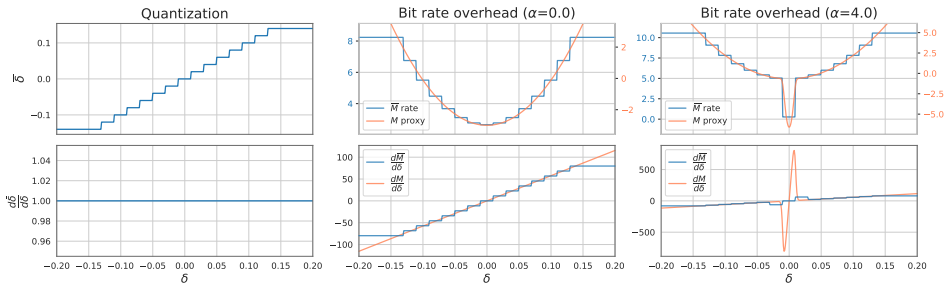


Figure 4: (Left) Illustrative example of the quantization effect and the corresponding gradient (using the Straight-Through estimator) for parameter update δ. (Middle) The true bit rate overhead (blue) and its continuous proxy (orange) of a (slab-only) Gaussian prior ($\alpha = 0$) and their gradients with respect to unquantized δ. (Right) The true bit rate overhead (blue) and its continuous proxy (orange) of a spike-and-slab prior ($\alpha = 4$) and their gradients with respect to unquantized δ. One can see how the effect of the spike (almost) fully disappears in the gradient of the quantized bit rate overhead, as the largest mass of the spike distribution is part of the center quantization bin.

Figure 4-right shows the same figure but for a model prior that includes a spike. Comparing true (discrete) model rate costs of the slab-only prior (middle-top), to these costs using the spike-and-slab prior (right-top), shows how the introduced spike reduces the number of bits to encode the zero-update (i.e. the center bin), at the cost of making larger updates more expensive in number of bits. Another interesting phenomena is visible when comparing the gradients of the discrete and continuous model rates for slab-only (middle-bottom) versus spike-and-slab prior (right-bottom). The effect of the spike almost fully disappears in the gradient of the discrete model rate. This is caused by the fact that most of the spike's mass is (by design) positioned inside the center quantization bin after quantization. Mathematically, this can be seen from filling in eq. (7) for the spike-and-slab prior:

$$\nabla_\delta \overline{M} = \quad -\frac{p_{\text{slab}}(\bar{\delta} + \frac{t}{2}) - p_{\text{slab}}(\bar{\delta} - \frac{t}{2}) + p_{\text{spike}}(\bar{\delta} + \frac{t}{2}) - p_{\text{spike}}(\bar{\delta} - \frac{t}{2})}{P_{\text{slab}}(\delta < \bar{\delta} + \frac{t}{2}) - P_{\text{slab}}(\delta < \bar{\delta} - \frac{t}{2}) + P_{\text{spike}}(\delta < \bar{\delta} + \frac{t}{2}) - P_{\text{spike}}(\delta < \bar{\delta} - \frac{t}{2})}. \quad (11)$$

We can distinguish different behavior of eq. (11) for two distinct ranges of $\bar{\delta}$:

- $\bar{\delta} = 0$
  Symmetry around zero-update: $p_{\text{slab}}(\bar{\delta}+\frac{t}{2}) = p_{\text{slab}}(\bar{\delta}-\frac{t}{2})$ and $p_{\text{spike}}(\bar{\delta}+\frac{t}{2}) = p_{\text{spike}}(\bar{\delta}-\frac{t}{2})$.
  $\implies \nabla_\delta \overline{M} = 0$

- $\bar{\delta} \neq 0$
  Since $t = 6\sigma_{\text{spike}}$:
  $p_{\text{spike}}(\bar{\delta}+\frac{t}{2}) \approx 0, \quad p_{\text{spike}}(\bar{\delta}-\frac{t}{2}) \approx 0, \quad P_{\text{spike}}(\delta < \bar{\delta}+\frac{t}{2}) \approx 0, \quad P_{\text{spike}}(\delta < \bar{\delta}-\frac{t}{2}) \approx 0.$
  $\implies \nabla_\delta \overline{M} = -\frac{p_{\text{slab}}(\bar{\delta}+\frac{t}{2}) - p_{\text{slab}}(\bar{\delta}-\frac{t}{2})}{P_{\text{slab}}(\delta<\bar{\delta}+\frac{t}{2}) - P_{\text{slab}}(\delta<\bar{\delta}-\frac{t}{2})}.$

From the $\bar{\delta} = 0$ case, one can see that inclusion of the spike leaves the gradient unbiased. The $\bar{\delta} \neq 0$ case shows that the spike does not influence the gradient in the limit when the spike's mass is entirely positioned within the center quantization bin. As the standard deviation of the spiky Gaussian was chosen to be $\frac{t}{6}$, a total of 0.3% of it's mass is in practice being quantized in an off-center quantization bin. This explains the slight increase of the off-center bins in the gradient of the discrete model rate costs in Fig. 4 (right-bottom). Comparing the gradient of the discrete versus continuous model rate costs for the spike-and-slab prior in Fig. 4 (right-bottom), we can see that the first order approximation between the two introduces a larger error than in the slab-only case (Fig. 4(middle-bottom)). This can be explained by the fact that the introduced spiky Gaussian has a higher tangent due to its support being more narrow than that of the Gaussian slab. Nevertheless, the use of the continuous model rate loss is preferred for finetuning as it (much more) strictly enforces zero-updates (thanks to the present gradient peaks around the center bin) than its discrete counterpart.

B SAMPLE SELECTION FROM XIPH DATASET

B.1 XIPH DATASET

The Xiph test videos can be found at `https://media.xiph.org/video/derf/`. Like Rippel et al. (2019) we select all 1080p videos, and exclude computer-generated videos and videos with inappropriate licences, which leaves us with the following videos:

`aspen_1080p`	`red_kayak_1080p`
`blue_sky_1080p25`	`riverbed_1080p25`
`controlled_burn_1080p`	`rush_field_cuts_1080p`
`crowd_run_1080p50`	`rush_hour_1080p25`
`dinner_1080p30`	`snow_mnt_1080p`
`ducks_take_off_1080p50`	`speed_bag_1080p`
`in_to_tree_1080p50`	`station2_1080p25`
`old_town_cross_1080p50`	`sunflower_1080p25`
`park_joy_1080p50`	`tractor_1080p25`
`pedestrian_area_1080p25`	`west_wind_easy_1080p`

B.2 XIPH-5N DATASET

Due to computational limits, we draw a sample of five videos, which we refer to as the Xiph-5N dataset. When drawing such a small sample randomly, a high probability arises of drawing an unrepresentative sample, including for example too many videos with either low or high finetuning potential. To alleviate this problem, we use the following heuristic to select $N(= 5)$ videos:

1. Evaluate the global model's RD performance on all videos for all values of rate-distortion tradeoff parameter β.

2. Per β value, rank all videos based on their respective $\mathcal{L}_{RD}$ loss.

3. For each video, average the $\mathcal{L}_{RD}$ rank over all values of β.

4. Order all videos according to their average rank, and select N videos based on $N+1$ evenly spaced percentiles.

The global model's RD performance for all videos from the Xiph dataset is shown in Figure 5. The five videos part of Xiph-5N are indicated with colors, and Table 2 provides more details about these five videos. The column titled *RD-tank percentile* shows the actual percentile at which the selected videos are ranked in the distribution. The computed (target) percentiles for $N = 5$ are 1/6, 2/6, ..., 5/6. For each percentile the video was selected that was closest to these computed percentiles. The last column denoting *Nr. of selected I-frames* denotes the number of I-frames after subsampling to 2 fps. Videos of which the original sampling frequency was not integer divisible by a factor 2, were subsampled with a factor resulting in the I-frame sampling frequency closest to 2 fps.

B.3 XIPH-5N 2FPS DATASET

We finally create an I-frame compression dataset by subsampling all videos in Xiph-5N to 2 fps. We refer to this dataset as Xiph-5N 2fps.

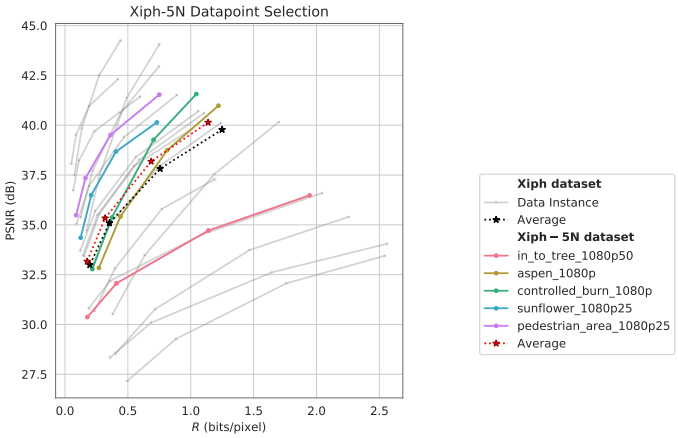


Figure 5: RD performance of global baseline for all datapoints in Xiph. The five selected datapoints in Xiph-5N are indicated with colours.

Video	RD-rank percentile	Target percentile	width × height × frames	Original fps	Duration (s)	Nr. of I-frames at 2 fps
`in_to_tree`	0.22	0.167	1920 × 1080 × 500	50	10.0	20
`aspen`	0.37	0.333	1920 × 1080 × 570	30	19.0	38
`controlled`	0.50	0.500	1920 × 1080 × 570	30	19.0	38
`sunflower`	0.69	0.667	1920 × 1080 × 500	25	20.0	42
`pedestrian_area`	0.82	0.833	1920 × 1080 × 375	25	15.0	32
AVERAGE	**0.52**	**0.500**	**1920 × 1080 × 503**	**32**	**16.6**	**34**

Table 2: Characteristics of the five selected videos in Xiph-5N 2fps.

C GLOBAL MODEL ARCHITECTURE AND TRAINING

For our neural compression model, we adopt the architecture proposed by Ballé et al. (2018), including the mean-scale prior from Minnen et al. (2018). We use a shared hyperdecoder to predict the mean and scale parameters. Like (Ballé et al., 2018) we use a model architecture with fewer parameters for the low bit rate regime ($\beta \geq 1e{-}3$). Table 3 indicates both the model architecture and the number of parameters, grouped per sub-model. The upper row in this table links the terminology proposed by Ballé et al. (2018) to conventional VAE terminology which we follow in this work.

Figure 6 provides a visual overview of the model architecture, where we use $\boldsymbol{z}_2$ and $\boldsymbol{z}_1$ to indicate the latent and hyper-latent space respectively (referred to as $\boldsymbol{y}$ and $\boldsymbol{z}$ in the original paper of Ballé et al. (2018)). Note that even though we adopt a hierarchical latent variable model, we simplify the notation by defining a single latent space $\boldsymbol{z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ throughout this work.

During training we adopt a mixed quantization strategy, where the quantized latent $\boldsymbol{z}_2$ are used to calculate the distortion loss (with their gradients estimated using the Straight-Trough estimator from Bengio et al. (2013)) , while we use noisy samples for $\boldsymbol{z}_1, \boldsymbol{z}_2$ when computing the rate loss.

Optimization of eq. (1) on the training fold of the **CLIC19** dataset, was done using the Adam optimizer with default settings (Kingma & Ba, 2014), and took 1.5M steps for the low bit rate models, and 2.0M steps for the high bit rate models. Each step contained 8 random crops of 256×256 pixels, and the initial learning rate was set to $1e{-}4$ and lowered to $1e-5$ at 90% of training.

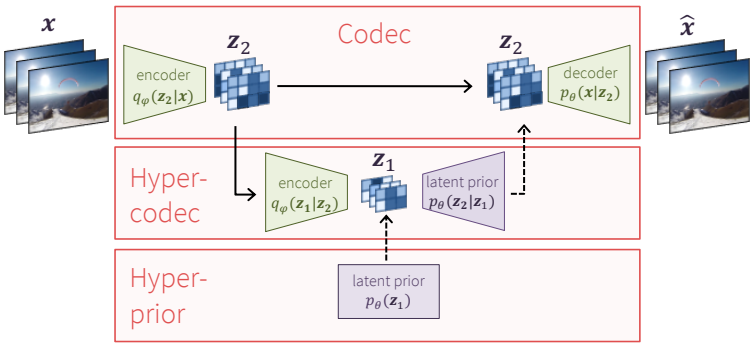


Figure 6: The mean-scale hyperprior model architecture visualized using the VAE framework.

Table 3: Model architecture and parameter count for the adopted hyperprior model. As suggested by Ballé et al. (2018), a distinct architecture is used for the low and high bit rate regime. We report the number of output channels per layer, for the exact model architecture we refer to Ballé et al. (2018). Note that the last column refers to the total number of parameters that needs to be known at the receiver side.

	Transmitter		**Receiver**			
	Encoder $q(\boldsymbol{z}_2\|\boldsymbol{x})$	Hyper Encoder $q(\boldsymbol{z}_1\|\boldsymbol{z}_2)$	Hyperprior $p(\boldsymbol{z}_1)$	Hyper Decoder $p(\boldsymbol{z}_2\|\boldsymbol{z}_1)$	Decoder $p(\boldsymbol{x}\|\boldsymbol{z}_2)$	Nr. of receive parameters
Low bit rate model						
Layers x Output channels	4x192	3x128	3x3	2x128 + 1x256	3x192 + 1x3	-
Parameter count	2.89M	1.04M	2.89M	5.50k	1.26M	4.16M
High bit rate model						
Layers x Output channels	4x320	3x192	3x3	2x192 + 1x384	3x320 + 1x3	-
Parameter count	8.01M	2.40M	8.01M	8.26k	2.95M	10.97M

D RATE-DISTORTION FINETUNING PERFORMANCE PER VIDEO

Figure 7 shows the RD plots for the different videos after finetuning for 100k steps. Note that the blue lines, indicating global model performance, differ per video. Full-model finetuning outperforms the global model, *encoder-only* and direct latent optimization for all videos. Note however that some videos seem more difficult to be finetuned than others, e.g. `controlled_burn` versus `sunflower`.

Figure 8 shows for each of these videos the finetuning progression over training steps. Differences in performance are visible among videos. The videos that most easily finetune, e.g. `sunflower`, show quicker performance improvement after the start of finetuning, and also continue to improve more over time.

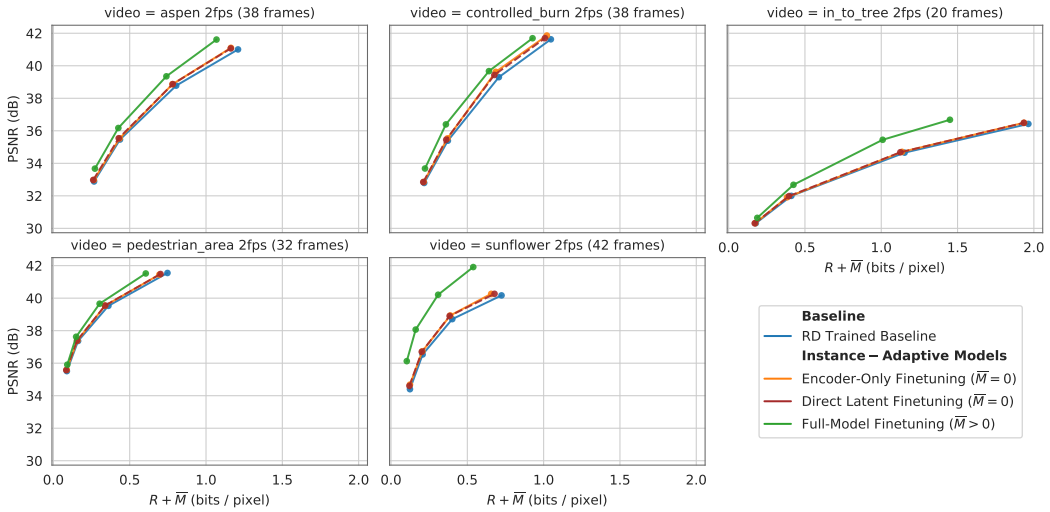


Figure 7: RD performance for instance-adaptive encoder-only and full-model finetuning, compared with the performance of the global model. Each plot is evaluated on a different video, and the instance-adaptive models used to create these graphs are finetuned on this corresponding single video.

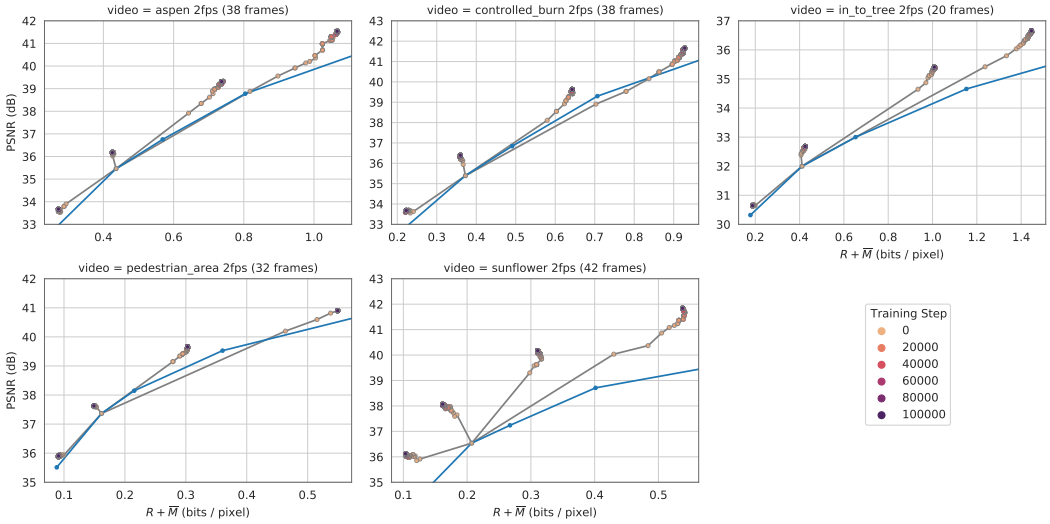


Figure 8: Progression of finetuning over time for all videos in Xiph-5N 2fps.