General-purpose, long-context autoregressive modeling with Perceiver AR

Curtis Hawthorne [*] [1] Andrew Jaegle [*] [2] Cătălina Cangea [2] Sebastian Borgeaud [2] Charlie Nash [2]
Mateusz Malinowski [2] Sander Dieleman [2] Oriol Vinyals [2] Matthew Botvinick [2] Ian Simon [1] Hannah Sheahan [2]
Neil Zeghidour [1] Jean-Baptiste Alayrac [†] [2] João Carreira [†] [2] Jesse Engel [†] [1]

Abstract

Real-world data is high-dimensional: a book, image, or musical performance can easily contain hundreds of thousands of elements even after compression. However, the most commonly used autoregressive models, Transformers, are prohibitively expensive to scale to the number of inputs and layers needed to capture this long-range structure. We develop Perceiver AR, an **a**uto**r**egressive, modality-agnostic architecture which uses cross-attention to map long-range inputs to a small number of latents while also maintaining end-to-end causal masking. Perceiver AR can directly attend to over a hundred thousand tokens, enabling practical long-context density estimation without the need for hand-crafted sparsity patterns or memory mechanisms. When trained on images or music, Perceiver AR generates outputs with clear long-term coherence and structure. Our architecture also obtains state-of-the-art likelihood on long-sequence benchmarks, including 64×64 ImageNet images and PG-19 books.

1. Introduction

A central goal of artificial intelligence research is the development of systems that can identify structure in the world and use it to effectively perform tasks of interest. In the past few years, autoregressive (AR) modeling with attention architectures (sometimes referred to metonymically as "language modeling") has emerged as a viable path to achieving this goal. In AR modeling, a set of outputs are generated by (i) using a model to map a set of inputs to an output, (ii) appending that output to the set of inputs, and proceeding again from step (i) until the full set of outputs has been produced. This simple recipe can in principle be followed to express any input-output relationship, and breakthrough results have been achieved using Transformers (Vaswani et al., 2017) or related models to learn the input → output mapping (Vinyals et al., 2019; Brown et al., 2020; Jumper et al., 2021; Wu et al., 2021). For this to work, the model must be able to capture patterns in the input that are useful for predicting the next output.

Patterns in real-world data often depend on the details of many inputs (or "tokens," each of which represents a row of an input array), some of which are far away in space, time, or setting from the current output. Many pieces of music, for example, begin by stating a theme with clear melodic and rhythmic elements. Over the piece, these elements are gradually varied with increasingly elaborate restatements designed to draw listeners in. How does the precise timing of a phrase relate to its antecedents? How does a tonal motif persist and develop as it is restated? How does a new harmonization recontextualize a familiar melody? The structure of the piece emerges when each component is considered alongside many others.

There is a tension between this kind of long-form, contextual structure and the computational properties of Transformers. Transformers repeatedly apply a self-attention operation to their inputs: this leads to computational requirements that simultaneously grow quadratically with input length and linearly with model depth. As the input data grows longer, more input tokens are needed to observe it, and as the patterns in the input data grow more subtle and complicated, more depth is needed to model the patterns that result. Computational constraints force users of Transformers to either truncate the inputs to the model (preventing it from observing many kinds of long-range patterns) or restrict the depth of the model (denuding it of the expressive power needed to model complex patterns).

The goal of this work is to design an architecture that retains the well-known benefits of Transformers for autoregressive modeling while enabling long-range pattern recognition without adding extraneous complexity. To do this, we build on the Perceiver family of attention architectures (Jaegle et al., 2021; 2022), which have demonstrated excel-

[*] Equal contribution [†] Equal contribution [1] Google Research, Brain Team [2] DeepMind. Correspondence to: Curtis Hawthorne <fjord@google.com>, Andrew Jaegle <drewjaegle@deepmind.com>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

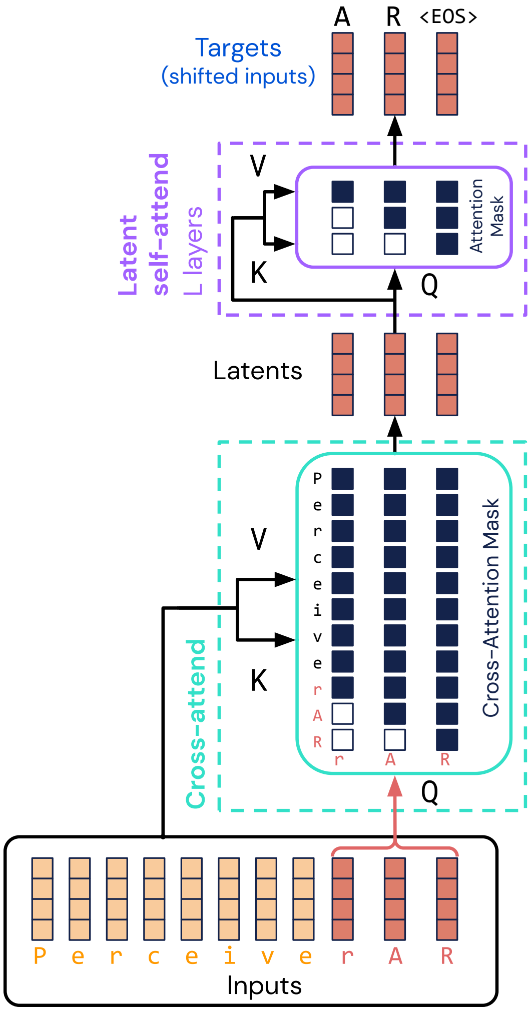


Figure 1. Perceiver AR maps inputs ($X \in \mathcal{R}^{M\times C}$; $M = 11$ shown) to a small latent ($Z \in \mathcal{R}^{N\times C}$; $N = 3$ shown) by cross-attention, querying with the N most recent inputs to produce one latent for each target. Latents subsequently interact via a deep stack of L self-attention layers to produce estimates for each target. Causal masking is used in both cross- and self-attention to maintain end-to-end autoregressive ordering.

lent performance on a wide range of large-context domains. Perceivers use cross-attention to map a full input array into a smaller latent array and perform all subsequent attention operations in the resulting latent space.

This decouples the computational requirements of processing a large input array from those required to make a network very deep, allowing deep Perceivers to be used on large numbers of inputs. However, because each model latent attends to all inputs regardless of position, Perceivers cannot be used directly for autoregressive generation, which requires that each model output depend only on inputs that precede it in sequence.

Perceiver AR solves this problem with three fixes: (i) introducing an ordering to the latents by identifying each latent with a single output element, (ii) using causally masked cross-attention to allow each latent to attend only to input elements that precede it in sequence, and (iii) using causally masked self-attention throughout the latent processing stack to preserve this autoregressive dependency structure end-to-end. These changes allow Perceiver AR's outputs to be decoded in autoregressive sequence while preserving the essential computational and memory benefits of other Perceiver architectures. As each of the architecture's outputs is conditioned on all prior inputs, the architecture is well-positioned to capture long-range dependencies.

We show that this architecture produces excellent results on several real-world domains with long-range context: RGB-level images (Section 5.2), tokenized language (Sections 5.3 to 5.5), and audio or symbolic music (Section 5.6). Input lengths for these tasks are summarized in Table 1. We demonstrate that Perceiver AR can learn to perfectly recognize long-context patterns over distances of at least 100k tokens on a synthetic copy task with known ground-truth structure (Section 5.1.1). We highlight several intriguing properties that result from decoupling input size from compute requirements: by keeping the long context, but changing the number of latents, we can (i) increase or decrease computational load at test time for improved (but slower) or faster (but somewhat worse) results (Section 5.2.1) or (ii) trade off model capacity against batch size at train time with no effect on test-time performance (Section 5.1.2).

We make the following contributions:

- We introduce Perceiver AR, an efficient, domain-agnostic architecture for autoregressive generation that can directly attend to over a hundred thousand tokens.
- We demonstrate the utility of using long contexts for autoregressive generation: Perceiver AR obtains state-of-the-art results on ImageNet and Project Gutenberg density estimation and produces samples with a high degree of coherence and fidelity on several challenging generation tasks (images, symbolic music, and audio).
- We explore the benefits of decoupling the computational requirements of handling long inputs from those of model depth: improved efficiency compared to the widely used decoder-only Transformer and Transformer-XL architectures and the ability to vary the compute used at test time to match a target budget.

Model code is available at `https://github.com/google-research/perceiver-ar`.

2. Autoregression and long-context modeling

Autoregressive models (e.g. Schmidhuber & Heil 1994; Rosenfeld 2000; Bengio et al. 2003; Graves 2013; van den

Task	Input Positions
Copy (Section 5.1)	131,072
ImageNet (Section 5.2)	12,289
PG-19 (Section 5.3)	4,096
Books (Section 5.4)	16,384
Wikitext-103 (Section 5.5)	8,192
Symbolic Music (Section 5.6)	65,536
Music Audio (Section 5.6)	65,536

Table 1. Maximum number of input (key-value) positions used for tasks in this paper. Most models used 1024 query positions.

Oord et al. 2016b; Uria et al. 2016) estimate the density of an example $X \in \mathcal{R}^M$ by decomposing it sequentially using the chain rule of probability:

$$p(X) = \prod_{m=0}^{M-1} p\Big(X_m \Big| X_{<m}\Big). \tag{1}$$

Each X_m is typically a token (an audio sample, an RGB channel, a character, etc.). The chain rule tells us that the density of an input X composed of arbitrarily many tokens can be estimated by sequentially estimating the conditional density of each of the X's tokens. This requires that $p(X_m|\ldots)$ be conditioned on all prior tokens that are useful for predicting the mth token.

Conditioning on all relevant inputs is challenging due to the difficulty of scaling standard models beyond small window lengths. In practice, models must be carefully designed so that the tokens that can be included in the context are as relevant as possible. This usually means that spatially or temporally local tokens are included in the context, but longer-range context is ignored or subsampled. As the context becomes smaller, this leads to worse approximations for signals that contain long-term dependencies.

Perceiver AR is designed to incorporate longer-term context into powerful density estimators, thereby allowing models to contextualize on more of a signal and giving better flexibility on how data is processed, moving us closer to the goal of general-purpose density modeling.

3. Perceiver AR

Perceiver AR follows Perceiver and Perceiver IO in addressing the problem of large inputs using a Transformer-style cross-attention module to map inputs $X \in \mathcal{R}^{M\times C}$ (C is the number of channels) to a smaller number of latents $Z_1 \in \mathcal{R}^{N\times C}$:

$$Z_1 \leftarrow \text{CrossAttend}(X, Z_0) \tag{2}$$

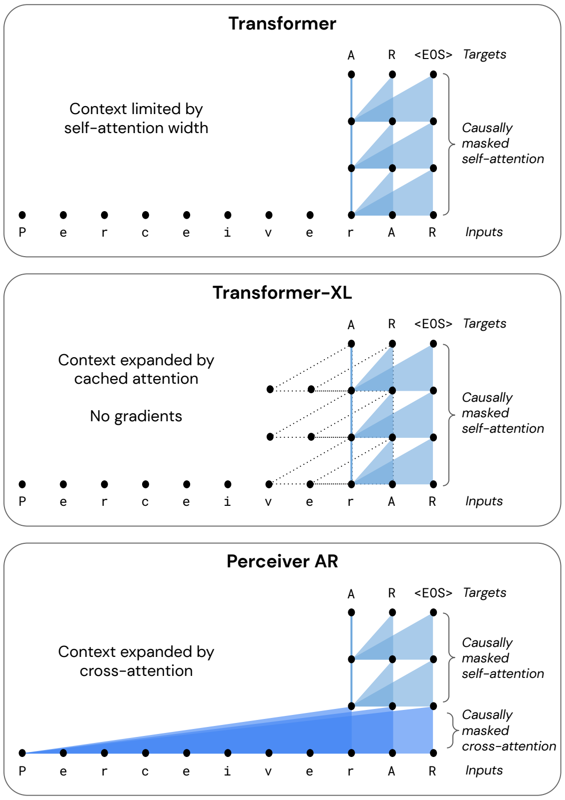

Transformer
A
R
<EOS>
Targets
Context limited by self-attention width
Causally masked self-attention
P e r c e i v e r A R
Inputs
Transformer-XL
A
R
<EOS>
Targets
Context expanded by cached attention
No gradients
Causally masked self-attention
P e r c e i v e r A R
Inputs
Perceiver AR
A
R
<EOS>
Targets
Context expanded by cross-attention
Causally masked self-attention
Causally masked cross-attention
P e r c e i v e r A R
Inputs

Figure 2. Perceiver AR compared to a standard decoder-only Transformer (Liu et al., 2018) and Transformer-XL (train-time configuration) (Dai et al., 2019) during training. Only the subset of configurations that fit in device memory are shown. For a given self-attention stack width ($N = 3$ shown here), the Transformer can process only the same number of input tokens. Transformer-XL incorporates more context while maintaining the width of the processing stack by caching and computing only the forward pass for longer-range inputs. In practice, Transformer-XL can incorporate only a moderate amount of additional context (see Figure 3). Perceiver AR uses a single masked cross-attend to enable training on much longer input contexts without requiring a wider self-attention stack.

The latent array Z_1 is typically small ($N < M$), so it is amenable to processing by more self-attention modules:

$$Z_{l+1} \leftarrow \text{SelfAttend}(Z_l, Z_l). \tag{3}$$

This operation does not depend on the number of input points M, and N can be chosen so that this operation is affordable and repeated for many layers $l \in [1, L]$. This strategy leads to an architecture (Figure 1) with complexity $\mathcal{O}(MN) + \mathcal{O}(LN^2)$ due to the cross-attention and the latent self-attention stack, respectively (Jaegle et al., 2022). For deep networks, the self-attention stack is where the bulk of compute occurs.

But reducing the number of points from M to N prevents us from establishing the causal dependency between all input and output points used by Transformers for autoregressive modeling. Perceiver AR adapts the latents for autoregressive modeling by introducing causal masking to both the input cross-attention and to the latent self-attention layers (Figure 1) and assigning one latent to each of the final N points of the input (i.e. those with the largest number of antecedents; $N = 3$ in Figure 1). The influence of inputs that come after a given latent are masked at the cross-attention and all self-attention layers.

To see how this works, consider the second (middle) latent in Figure 1: this latent is constructed by querying the input with `A`'s embedding. Because `R` follows `A` in sequence, `R`'s embedding is masked out, both at the cross-attention and at the subsequent self-attention layers.

This procedure allows models to attend to long contexts while also preserving the autoregressive ordering of the targets through the entire network. The same procedure can be applied to any input that can be ordered, as long as masking is applied. For example, an image's RGB channels can be ordered in raster scan order, by decoding the R, G, and B color channels for each pixel in the sequence (Section 5.2) or even under different permutations (Section 5.2.2).

See Appendix C for in-depth mathematical description of Perceivers and the Perceiver AR architecture and Appendix E for additional technical details.

4. Related work

4.1. Relationship to Transformer and Transformer-XL

Perceiver AR decouples the length of the input from the computational requirements of the model, which are primarily controlled by the number of latents. In contrast, standard decoder-only Transformer architectures (Figure 2) maintain a one-to-one correspondence between inputs and outputs throughout the network, leading to $\mathcal{O}(LM^2)$ scaling.

Perceiver AR also scales better to longer context in practice than Transformer-XL, perhaps the most commonly used method for extending context length in practice. Transformer-XL incorporates longer context by allowing attention over long-context positions at every layer in the forward pass and stopping gradient propagation to these positions in the backward pass. Transformer-XL also typically uses fewer positions at train than at test time, which further improves scaling at train time. But even with these modifications, the input size and model depth are still coupled, and as the context and depth increase, the forward pass becomes a compute and memory bottleneck. In practical terms, Perceiver AR scales better when both the context size and model depth increase. In Figure 3, we compare the wall-clock time per step of a decoder-only Transformer, Transformer-XL, and Perceiver AR in our codebase.

4.2. Relationship to other scalable attention architectures

Perceiver AR limits the computational requirements of processing long sequences by avoiding the use of one computational node for each input element. This strategy is also a feature of other architectures, like Set Transformer (Lee et al., 2019) and Luna (Ma et al., 2021), that use differently shaped query and key-value inputs to limit the cost of attention and produce "downsampled" intermediate representations that are smaller than the input's size. Unlike the Perceiver family of architectures, both Set Transformer and Luna alternate downsampling and upsampling attention operations throughout the architecture, which results in an architecture that mitigates some of the cost of standard Transformers, but still has compute requirements multiplicative in input size and model depth. Of this family of architectures, Luna is most similar, as it is compatible with (causally masked) autoregressive modeling.

Several other architectures (Dai et al., 2020; Nawrot et al., 2021; Clark et al., 2021) reduce the processing requirements of Transformers by sequentially compressing the input with attention or convolution, but rely on the use of multiple large, quadratic-complexity attention layers or exploit locality assumptions that limit their generality. This makes them more efficient when applied to input chunks of similar size as normal Transformers (i.e. a few thousand), but reduces their utility for longer contexts.

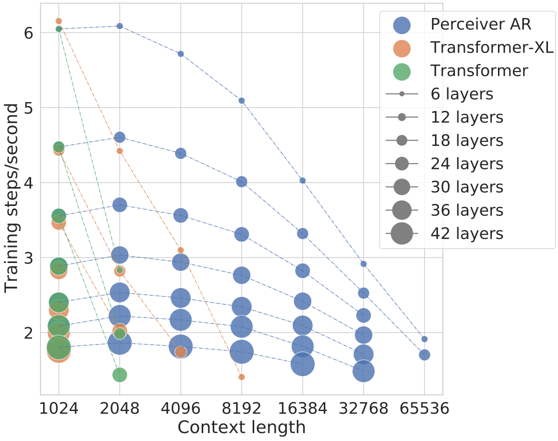


Figure 3. Training speed comparison on TPUv3 for Perceiver AR compared to a standard decoder-only Transformer and Transformer-XL. The Transformer is limited to a context length of 2,048 tokens, even with only 6 layers—larger models and larger context length require too much memory. Using the same 6-layer configuration, we can scale the Transformer-XL memory to a total context length of 8,192. Perceiver AR scales to 65k context length, and can be scaled to over 100k context with further optimization.

4.3. Relationship to encoder-decoder architectures

Perceiver AR bears a resemblance to encoder-decoder Transformers (Vaswani et al., 2017), seq2seq (Sutskever et al., 2014), and other encoder-decoder models. Encoder-decoder models pass long-context inputs into an encoder stack (e.g. `Perceive`, as in Figures 1 and 2) and use the outputs of the encoder to contextualize the immediate antecedents of each output (e.g. `rAR`), which are processed by a separate decoder stack. In contrast, Perceiver AR passes both inputs and targets through a single, shared processing stack. This allows each target to learn how to use long-context and recent input as needed, with minimal architectural restrictions. From this point of view, Perceiver AR can be viewed as an encoder-decoder architecture with 0 encoder layers. By handling all inputs with a single (causally-masked) cross-attention, Perceiver AR sidesteps the need for separate encoder and decoder stacks.

See Appendix D for an extended discussion of Perceiver AR's relationship to other methods for long-context modeling by efficient attention (D.1), architecture design (D.2), and input tokenization (D.3).

5. Results

We evaluate Perceiver AR on a number of different domains to demonstrate its flexibility and evaluate its ability to capture long-range structure on a variety of data. In all experiments except where mentioned, we use pre-layernorm attention modules (Xiong et al., 2020) and squared-ReLU nonlinearities (So et al., 2021).

For each domain, we tuned models against eval perplexity with ad hoc hyperparameter sweeps as our compute permitted. We typically tuned channel size, head dimensions, and model depth. See each domain's section and appendices F and G for more details.

5.1. Copy Task

5.1.1. Long input length

We first verify that the architecture can attend to very long input lengths on a synthetic copy task. Using a model with only 6 layers of 1024 latents, we provide an input with length 2^{17} (131,072). The model is trained on sequences containing a `[BOS]` (Beginning of Sequence) token followed by 65,535 random bytes (encoded as tokens taking one of 256 values). Those random bytes are then mirrored for the second half of the sequence and followed by an `[EOS]` (End of Sequence) token. This results in a maximum copy distance of $2^{17} - 2$ tokens. Train and eval loss are calculated on only the second half of the sequence to avoid training on noise.

After training for 25k steps, the model was able to correctly predict the mirrored tokens plus `[EOS]` of 12 unseen validation sequences (a total of 786,432 tokens) with 100% accuracy. This experiment demonstrates that the model can successfully attend to individual tokens within very long sequences and that a relatively small training signal of 1024 targets per sequence can successfully propagate through the cross-attend bottleneck even when most of the inputs are irrelevant for a given target. As an historical aside, we note that one antecedent of this experiment is the copy task used to validate the Neural Turing Machine (Graves et al., 2014) and Differentiable Neural Computer (Graves et al., 2016). These models showed nearly perfect accuracy when copying up to about length-50 sequences of random 2^6- or 2^8-bit inputs, while Perceiver AR shows perfect accuracy when copying sequences of 2^{16} random 2^8-bit tokenized inputs.

5.1.2. Number of training targets

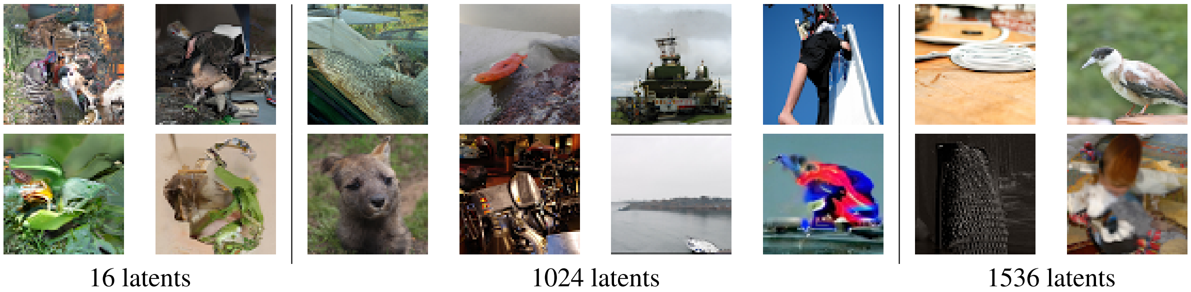
In a standard decoder-only Transformer model, the input length, number of latent processing nodes per layer, and number of training targets are always the same. Perceiver AR allows the flexibility of any ratio of input length to number of latents and training targets. Changing the width of the self-attention stack affects the expressivity of the network and also alters the number of training outputs for which loss can be computed per sequence in a batch.

To illustrate this effect, we trained models on the copy task with a context length of 8,192 tokens using different numbers of latent nodes and batch sizes (Table 2). To reduce the effect of network expressivity, we use only 1 self-attention layer for these experiments. We found that after training for 25k steps with 1024 latents and a batch size of 128, the model converged and predicted the second half of sequences in the test set with 100% accuracy. If we reduced the batch size to 64, the model did not converge and achieved $< 1\%$ accuracy on the test set. However, if we kept the batch size at 64 and increased the number of latents (and therefore training targets per sequence) to 2048, the model successfully converged.

	1024 latents	2048 latents
64 batch	✗	✓
128 batch	✓	✓

Table 2. Convergence on the copy task with a context length of 8,192 tokens. Either increasing the number of latents (and therefore number of targets) or batch size has a similar effect, as discussed in Section 5.1.2.

Depending on memory, compute, and expressivity requirements, increasing the batch size or number of latents may be better: Perceiver AR enables this flexibility.



16 latents

1024 latents

1536 latents

Figure 4. Representative samples from the ImageNet model. The 4 images on the left were generated using only 16 latents, the middle 8 with 1024 latents (same as train time), and the right 4 with 1536 latents. The same 60-layer model is used for all configurations and all can attend to the full sequence. The full batches from which these images were drawn are shown in Appendix A.

Model	Type	Bits/Dim
PixelCNN	AR	3.57
Sparse Transformer	AR	3.44
Routing Transformer	AR	3.43
Combiner	AR	3.42
VDM	**Diff**	**3.40**
Perceiver AR (ours)	**AR**	**3.40**

Table 3. Results on Downsampled ImageNet (64 × 64) density estimation in bits/dim on the validation set, lower is better. We compare our results against the autoregressive models PixelCNN (van den Oord et al., 2016b), Sparse Transformer (Child et al., 2019), Routing Transformer (Roy et al., 2021), and Combiner (Ren et al., 2021), and the diffusion model Variational Diffusion Model (Kingma et al., 2021).

# Latents	Bits/Dim	Generation (minutes)
16	3.5664	1.99
64	3.4946	2.02
512	3.4139	2.81
1024	3.4025	3.68
1536	3.3986	4.69
2048	3.4018	5.88
4096	3.5569	12.28

Table 4. Results on Downsampled ImageNet (64 × 64) density estimation in bits/dim on the validation set (lower is better) when changing the number of latents used at eval time (1024 used at train time). Generation time is how long a single image takes to infer on a TPUv3 core using activation caching described in Appendix E.3. The same 60-layer model is used for all configurations and all can attend to the full sequence.

5.2. ImageNet 64 × 64

To test this architecture's capabilities in the image modality, we use the downsampled ImageNet dataset (van den Oord et al., 2016b) at the 64 × 64 resolution. Similar to the training procedure used in Sparse Transformer (Child et al., 2019), we flatten the image into a sequence of color channel bytes (256 possible values) for each pixel in raster scan order. After adding a `[BOS]` token to the beginning of the sequence, this results in a length of 12,289. Each input has 3 randomly initialized position embeddings added to it for row (64), column (64), and color channel (3). No data augmentation is used.

We train a model with 1024 latents in 60 self-attention layers after the initial cross-attend. After 750k steps, we achieve 3.40 bits/dim on the validation set, exceeding the performance of previous autoregressive models (Table 3). Generated images samples are in Figure 4 and Appendix A.

5.2.1. Varying compute at test time

Because Perceiver AR decouples input length from the number of latents in the self-attention stack and because no position-specific parameters are learned in this model, we have the intriguing option of evaluating with a different number of latents than were used during training, while still maintaining the ability to attend to the full input sequence (illustrated further in Figure 12).

Input Context	Standard Ordering	$R \to G \to B$
1024	3.55	4.63
12289	3.54	3.53

Table 5. Impact of context length and sequence ordering for ImageNet (64 × 64). Results are bits/dim on the validation set, lower is better. ImageNet examples contain 12,288 (64 × 64 × 3) tokens. For short contexts, the sequence ordering has a large effect on performance. For long contexts, the effect is small.

We found that increasing the number of latents up to 2x the number used during training improves model performance, and decreasing the number latents results in gracefully degrading performance despite dramatic reductions in compute requirements, as seen in Table 4. For example, when using only 16 latents to attend to the full input sequence of 12,289 tokens, the model achieves the same bits/dim as PixelCNN (van den Oord et al., 2016b). This kind of flexibility enables a single model to be used in scenarios with

Model	Context length	# layers	Val ppl.	Test ppl.
Transformer-XL (Rae et al., 2019)	512+1024	36	45.5	36.3
Compressive Transformer (Rae et al., 2019)	512+512+2x512	36	43.4	33.6
Routing Transformer (Roy et al., 2021)	8192	22	-	33.2
Perceiver AR (ours)	2048	60	45.9	**28.9**
Perceiver AR (ours)	4096	60	45.9	29.0

Table 6. Results on PG-19 language modeling. Results are shown in perplexity (ppl.), lower is better. Baseline results are reproduced from the original papers. Routing Transformer does not report validation set performance. The context lengths shown include memory (Transformer-XL and Compressive Transformer) and compressive memory (the latter only).

varying compute, latency, and quality requirements, without requiring additional training or a distillation process.

We suspect these models could be made even more flexible to the number of latents used during evaluation or inference if variable latent access is incorporated into training, but we leave a full exploration of these ideas to future work.

5.2.2. Impact of sequence ordering and length

We test Perceiver AR's ability to utilize context beyond the width of its self-attention stack by training on image sequences with strong long-range dependencies. Typically autoregressive models of image data use a raster scan ordering, where RGB subpixels in each image location are generated in sequence before moving on to the next location. We re-order ImageNet image data so that all the red subpixels are predicted in sequence, then the green, then the blue. This induces strong long range dependencies between subpixels in the same spatial location.

We train 16-layer versions of the models in Section 5.2 for 30k steps (due to compute constraints), and compare short context models (1024 inputs) to full context models (12,289 inputs). As a baseline we also train both model variants on sequences with the standard ordering. The results are shown in Table 5. We find that for the standard ordering, both small and long context models perform similarly. However, on the re-ordered image data, the short context model has significantly worse performance. Whereas the long context model has comparable performance to the standard ordering baselines. This indicates that our long context model is able to access and process information at distant timesteps.

For general domains, we often do not know which long-range dependencies are important, and this experiment shows that the performance impact of missing existing dependencies can be severe. Perceiver AR provides a solution by extending Transformer context size in a scalable way.

5.3. Project Gutenberg (PG-19)

We next test the architecture for language modeling on the Project Gutenberg (PG-19) dataset (Rae et al., 2019), a collection of English-language books published before 1919 and scraped from the Project Gutenberg eBook depository. We use PG-19 as it is publicly available and contains a large number of words (1.97B train, 3.01M validation, 6.97M test) drawn from a reasonably large number of books (28k train, 50 validation, 10 test). We evaluate Perceiver AR on PG-19 using Subword-tokenized inputs as in prior work (Rae et al., 2019; Roy et al., 2021). We use the Subword tokenization settings reported in section 4.2 of Rae et al. (2019), and we trained models until the loss converged on a subset of the validation set (after training on about 200k steps at batch size 2048, or about 420B total tokens).

We compare Perceiver AR to state-of-the-art numbers from the literature (Table 6). The best models reported by these papers use 36 (Compressive Transformer; Rae et al. 2019) and 22 layers (Routing Transformer; Roy et al. 2021). We find that Perceiver AR outperforms state-of-the-art methods on this dataset when using the same tokenization. However, consistent with previous work, we saw no evidence of improvement beyond 2k tokens on PG-19 (Sun et al., 2021).

Motivated by early experiments, both models use large input embeddings (4096), large numbers of cross-attend heads (128), and high cross-attend dropout (0.96875 for the 4096-context model and 0.875 for the 2048-context model; see Appendix E.2). The results presented in Table 6 suggest that simply scaling Perceiver AR can produce excellent results even with very high levels of cross-attend dropout.

Consistent with prior work, we notice that models exhibit a consistent drop in performance between PG-19 validation and test sets. This is likely due to the relatively small number of books contained in the PG-19 validation (50 books) and test (100 books) sets. Despite the large number of tokens, the small number of books on the validation set will lead many of these tokens to exhibit shared content and style, limiting how representative the validation set can be of the task as a whole. As noted by Sun et al. (2021), the PG-19 validation set contains at least one book (out of 50) that is arguably out of distribution for the train set.

To better understand the effect of context in language modeling, we next evaluate on a large internal book dataset where the number of documents is not a concern (Section 5.4).

Model	Context	Depth	Steps/Sec	Eval ppl.
AR	1024	36	2.19	14.006
T-XL	1024	23	2.17	14.822
AR	4096	36	2.09	13.806
T-XL	1024	24	2.06	14.719
AR	8192	36	1.95	13.791
T-XL	1024	25	1.97	14.593
AR	16384	36	1.75	13.749
T-XL	1024	28	1.76	14.276

Table 8. Books results on the test set, shown in perplexity (ppl.), lower is better, from 36-layer Perceiver AR models with varying contexts and Transformer-XL models with depth matched for compute (steps/sec), trained for 500k steps.

5.4. Books

Here, we study the usefulness of longer input context by training on an internal dataset containing 4 million books published between 1500 and 2008. This dataset was previously used in Rae et al. 2021 as part of the MassiveTest dataset. We train models with 1024 latents, 36 layers and $\{1024, 4096, 8192, 16384\}$ input context tokens.

These results (Table 7) show a clear trend: better results are obtained with contexts longer than 1024.

Model	Context	Eval ppl.	Train Steps/sec
Perceiver AR	1024	14.88	2.19
Perceiver AR	4096	14.60	2.09
Perceiver AR	8192	14.57	1.95
Perceiver AR	16384	14.56	1.75

Table 7. Books results, shown in perplexity (ppl.), lower is better.

We also run experiments where we compare our model against a Transformer-XL baseline, with compute (measured by steps per second) matched as closely as possible. Models are trained with a batch size of 256 for 500k steps.

First, we evaluate 36-layer Perceiver AR models with $\{1024, 4096, 8192, 16384\}$ context lengths, respectively. Each of them is matched with a Transformer-XL trained on sequences of length 1024 and $\{23, 24, 25, 28\}$ layers, respectively. Table 8 shows that our model improves consistently over the Transformer-XL in this controlled scenario.

In Table 9, we then compare the deepest Transformer-XL model (42 layers) that fits in memory against 4 Perceiver AR models with the same context lengths and respective numbers of self-attention layers $\{62, 61, 60, 56\}$. Remarkably, we were not able to increase the number of layers for some AR models—and achieve a closer match in compute—without running out of memory. Instead, we ran the deepest possible models for a specific context length. Even in this scenario, all Perceiver AR models performed better than the deepest Transformer-XL.

Model	Context	Depth	Steps/Sec	Eval ppl.
T-XL	1024	42-layer	1.17	13.253
AR	1024	62-layer	1.19	12.849
AR	4096	61-layer	1.21	12.680
AR	8192	60-layer	1.25	12.660
AR	16384	56-layer	1.26	12.816

Table 9. Books results on the test set, shown in perplexity (ppl.), lower is better, from a 42-layer Transformer-XL model and Perceiver AR models with varying contexts and depth matched for compute (steps/sec), trained for 500k steps.

5.5. Wikitext-103

We further evaluate on the Wikitext-103 (Merity et al., 2017) dataset, a commonly used word-level language modeling benchmark. The dataset consists of 28,475 Wikipedia articles containing between 68 and 26,993 words, averaging at 3.6k words. Wikitext-103 is a small dataset where strong regularization is required to prevent severe overfitting and to obtain good performance. Nonetheless, we obtain competitive results, suggesting Perceiver AR's utility even for relatively small datasets.

Model	Valid ppl.	Test ppl.
Adaptive inputs	18.0	18.7
Transformer-XL	18.3	18.2
Shortformer	17.5	18.15
Compressive Transformer	16.0	17.1
Routing Transformer	-	15.8
Transformer-XL (ours)	17.58	18.42
Perceiver AR (1024)	17.86	18.52
Perceiver AR (2048)	17.60	18.35
Perceiver AR (4096)	17.66	18.25
Perceiver AR (8192)	17.58	18.37

Table 10. Results on the Wikitext-103 language modeling benchmark. Baseline results are reproduced from the original papers: Adaptive inputs (Baevski & Auli, 2018), Transformer-XL (Dai et al., 2019), Shortformer (Press et al., 2021), Compressive Transformer (Rae et al., 2019), and Routing Transformer (Roy et al., 2021). Transformer-XL (ours) is a reimplementation in our codebase. We train Perceiver AR models with varying context lengths.

We show results in Table 10. Perceiver AR performs on par with Transformer-XL Large (Dai et al., 2019) and Shortformer (Press et al., 2021). We also include the state of the art results on Wikitext-103 (trained on Wikitext-103 only) (Rae et al., 2019; Roy et al., 2021). Increasing the context length from 1,024 to 2,048 for Perceiver AR does improve perplexity but further increasing to 4,096 or 8,192 is harm-

Model	MAESTRO	Test	Validation
Music Transformer	v1	-	1.84
Perceiver AR	v1	1.82	1.82
Perceiver AR	v3	1.91	1.90

Table 11. Results on MAESTRO symbolic music generation on Test and Validation datasets. Results are shown in negative log-likelihood, lower is better. Baseline result is reproduced from (Hawthorne et al., 2019).

ful. This effect has been noted in previous work (Press et al., 2021; Sun et al., 2021) and further provides evidence that current language model performance on small benchmarks is not bottlenecked by their limited range context.

5.6. MAESTRO

We also evaluate Perceiver AR in the music domain, with samples available in the online supplement (`https://bit.ly/3uF5LJg`). Here, modeling long-range dependencies is essential for obtaining good representations (Huang et al., 2019). We use the MAESTRO dataset (Hawthorne et al., 2019), which contains approximately 200 hours of music in both symbolic (MIDI) and audio modalities. For symbolic data, we experiment with both v1 and v3 versions of MAESTRO. The former allows us to compare with the existing state-of-the-art; the latter is an improved set with approximately 10% more performances. We use the MIDI tokenization described in Section A.2 of Huang et al. (2019).

After training for 1M steps with an input context of 4096 tokens and 2048 latents in 12 self-attention layers, our model obtains a lower negative log-likelihood (Table 11) than Music Transformer (Huang et al., 2019; Hawthorne et al., 2019). That model had 6 layers and was trained on random crops of 2048 tokens, but benefited from data augmentation, which ours did not. We also report results on MAESTRO v3.

For audio data, we generate vector-quantized embeddings using the SoundStream neural audio codec (Zeghidour et al., 2021) at several bitrates. Lower-bitrate codecs model coarser structure and enable training on a longer time window for a fixed context length, but at the expense of lower audio fidelity. We show results in Table 12.

SoundStream bitrate	Context	Test	Validation
12kbps	54.4s	2.49	2.34
18kbps	36.8s	2.60	2.57
22kbps	29.6s	2.65	2.62

Table 12. Perceiver AR negative log-likelihood results on SoundStream audio generation, for a fixed context length of 65536.

5.7. Music samples

To showcase the long-term coherence of these models, we introduce another symbolic task involving a larger, private dataset of piano performances (Simon et al., 2019). These were transcribed from 10,000+ hours of audio, using a variation of the Onsets and Frames model (Hawthorne et al., 2018). From this dataset, we only used pieces resulting in 1024–32,768 tokens. Samples shorter than the lower limit are unlikely to contain song content; at the other end, there are only about 200 pieces longer than 32,768 tokens. The same tokenization as for MAESTRO is used. We train a model with 1024 latents and 24 self-attention layers on input contexts of 32,768 tokens, achieving a negative log-likelihood of 1.24 on the test set.

We generate samples from the models trained on this new task, as well samples from MAESTRO v3 symbolic and SoundStream models. The samples obtained from the large transcription dataset exhibit stylistic and structural coherence that spans several minutes, containing repeating musical themes, chord patterns, arpeggios and even ritardandos. The audio domain samples exhibit the same tradeoff of audio fidelity and long-term structure seen in the previous section, but demonstrate strong correspondence to the original domain. Rendered audio samples are available in the online supplement: `https://bit.ly/3uF5LJg`.

6. Conclusion

We introduce Perceiver AR, an architecture designed for long-context autoregressive modeling. Perceiver AR scales to longer input sizes than the architectures typically used in practice (Transformers and Transformer-XL), while scaling to the depths needed for density estimation on real-world data. Perceiver AR decouples the computational requirements of processing many inputs from those of building deep networks. This gives us more control over how much compute is used for a given model at test time and allows us to smoothly trade off speed against performance. Perceiver AR produces good results on a number of domains. Our experiments suggest that the larger context used by Perceiver AR may open the door for more flexible autoregressive ordering strategies. Perceiver AR is a good candidate for a general-purpose, long-context autoregressive model.

7. Acknowledgements

Thanks to Chitwan Saharia for help with SR3 upsampling, Christos Kaplanis for Books guidance, and Jordan Hoffmann and Richard Tanburn for help with RoPE. Thanks to Daniel Toyama, Douglas Eck, Adam Roberts, Nando de Freitas, Dani Yogatama, Josh Gardner, Catalin Ionescu, Skanda Koppula, Rabeeh Karimi Mahabadi, Ethan Manilow, Viorica Patraucean, Oleh Rybkin, Nikolay Savinov, and others at DeepMind and Google Research for suggestions.

References

Baevski, A. and Auli, M. Adaptive input representations for neural language modeling. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of Machine Learning Research (JMLR)*, 2003.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse Transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with Performers. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Clark, J. H., Garrette, D., Turc, I., and Wieting, J. CANINE: pre-training an efficient tokenization-free encoder for language representation. *arXiv preprint arXiv:2103.06874*, 2021.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, 2019.

Dai, Z., Lai, G., Yang, Y., and Le, Q. Funnel-Transformer: Filtering out sequential redundancy for efficient language processing. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2020.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Graves, A., Wayne, G., and Danihelka, I. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenestette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538:471–476, 2016.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I., Raffel, C., Engel, J., Oore, S., and Eck, D. Onsets and frames: Dual-objective piano transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2018.

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., Elsen, E., Engel, J., and Eck, D. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Hessel, M., Budden, D., Viola, F., Rosca, M., Sezener, E., and Hennigan, T. Optax: composable gradient transformation and optimisation, in JAX!, 2020. URL http://github.com/deepmind/optax.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. Music Transformer: Generating music with long-term structure. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver: General perception with iterative attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., Henaff, O., Botvinick, M. M., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver IO: A general architecture for structured inputs & outputs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Zídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596:583–589, 2021.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Kingma, D. P., Salimans, T., Poole, B., and Ho, J. On density estimation with diffusion models. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2021.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Kudo, T. and Richardson, J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, 2018.

Lakhotia, K., Kharitonov, E., Hsu, W.-N., Adi, Y., Polyak, A., Bolte, B., Nguyen, T.-A., Copet, J., Baevski, A., Mohamed, A., and Dupoux, E. Generative spoken language modeling from raw audio. *arXiv preprint arXiv:2102.01192*, 2021.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set Transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., and Shazeer, N. Generating Wikipedia by summarizing long sequences. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

Ma, X., Kong, X., Wang, S., Zhou, C., May, J., Ma, H., and Zettlemoyer, L. LUNA: Linear unified nested attention. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2021.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. SampleRNN: An unconditional end-to-end neural audio generation model. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Nash, C., Menick, J., Dieleman, S., and Battaglia, P. W. Generating images with sparse representations. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

Nawrot, P., Tworkowski, S., Tyrolski, M., Kaiser, L., Wu, Y., Szegedy, C., and Michalewski, H. Hierarchical Transformers are more efficient language models. *arXiv preprint arXiv:2110.13711*, 2021.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N., and Kong, L. Random feature attention. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Polyak, A., Adi, Y., Copet, J., Kharitonov, E., Lakhotia, K., Hsu, W.-N., Mohamed, A., and Dupoux, E. Speech resynthesis from discrete disentangled self-supervised representations. *arXiv preprint arXiv:2104.00355*, 2021.

Press, O., Smith, N. A., and Lewis, M. Shortformer: Better language modeling using shorter inputs. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, 2021.

Rabe, M. N. and Staats, C. Self-attention does not need $O(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive Transformers for long-range sequence modelling. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., Rutherford, E., Hennigan, T., Menick, J., Cassirer, A., Powell, R., van den Driessche, G., Hendricks, L. A., Rauh, M., Huang, P.-S., Glaese, A., Welbl, J., Dathathri, S., Huang, S., Uesato, J., Mellor, J., Higgins, I., Creswell, A., McAleese, N., Wu, A., Elsen, E., Jayakumar, S., Buchatskaya, E., Budden, D., Sutherland, E., Simonyan, K., Paganini, M., Sifre, L., Martens, L., Li, X. L., Kuncoro, A., Nematzadeh, A., Gribovskaya, E., Donato, D., Lazaridou, A., Mensch, A., Lespiau, J.-B., Tsimpoukelli, M., Grigorev, N., Fritz, D., Sottiaux, T., Pajarskas, M., Pohlen, T., Gong, Z., Toyama, D., de Masson d'Autume, C., Li, Y., Terzi, T., Mikulik, V., Babuschkin, I., Clark, A., de Las Casas, D., Guy, A., Jones, C., Bradbury, J., Johnson, M., Hechtman, B., Weidinger, L., Gabriel, I., Isaac, W., Lockhart, E., Osindero, S., Rimell, L., Dyer, C., Vinyals, O., Ayoub, K., Stanway, J., Bennett, L., Hassabis, D., Kavukcuoglu, K., and Irving, G. Scaling language models: Methods, analysis & insights from training Gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research (JMLR)*, 2020.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. ZeRO: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

Ren, H., Dai, H., Dai, Z., Yang, M., Leskovec, J., Schuurmans, D., and Dai, B. Combiner: Full attention Transformer with sparse computation cost. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2021.

Rosenfeld, R. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88 (8):1270–1278, 2000.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with Routing Transformers. *Transactions of the Association for Computational Linguistics (TACL)*, 9, 2021.

Saharia, C., Ho, J., Chan, W., Salimans, T., Fleet, D. J., and Norouzi, M. Image super-resolution via iterative refinement. *arXiv preprint arXiv:2104.07636*, 2021.

Schmidhuber, J. and Heil, S. Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1): 142–146, 1994.

Shazeer, N. Fast Transformer decoding: one write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Simon, I., Huang, C.-Z. A., Engel, J., Hawthorne, C., and Dinculescu, M. Generating piano music with transformer. 2019. URL https://magenta.tensorflow.org/piano-transformer.

So, D. R., Mańke, W., Liu, H., Dai, Z., Shazeer, N., and Le, Q. V. Primer: Searching for efficient Transformers for language modeling. *arXiv preprint arXiv:2109.08668*, 2021.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. RoFormer: Enhanced Transformer with rotary position embedding. *arXiv preprint arxiv:2104.09864*, 2021.

Sun, S., Krishna, K., Mattarella-Micke, A., and Iyyer, M. Do long-range language models actually use long-range context? In *Proceedings of the Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2014.

Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. Neural autoregressive distribution estimation. *Journal of Machine Learning Research (JMLR)*, 2016.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016b.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2017.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., Jones, L., Kaiser, L., Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N., and Uszkoreit,

J. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of Neural Information Processing Systems (NeurIPS)*, 2017.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, Dani Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Chris, A., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, 2019.

Wang, B. Mesh transformer Jax, 2021. URL https://github.com/kingoflolz/mesh-transformer-jax.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Weston, J., Chopra, S., and Bordes, A. Memory networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Wu, C., Liang, J., Ji, L., Yang, F., Fang, Y., Jiang, D., and Duan, N. NÜWA: Visual synthesis pre-training for neural visual world creation. *arXiv preprint arXiv:2111.12417*, 2021.

Wu, F., Fan, A., Baevski, A., Dauphin, Y. N., and Auli, M. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T.-Y. On layer normalization in the Transformer architecture. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big Bird: Transformers for longer sequences. *Proceedings of Neural Information Processing Systems (NeurIPS)*, 33, 2020.

Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., and Tagliasacchi, M. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (TASLP)*, 2021.

A. ImageNet Samples

Full batches of the generated images used to populate Figure 4 can be seen in Figures 5 to 8. All images were generated with a temperature of 1.0. We also show upsampled versions of the images that were generated with 1536 latents by using SR3 (Saharia et al., 2021) to achieve a resolution of 256 × 256 in Figure 9.

B. Books

As discussed in Section 5.4, we train all models on the Books dataset with 1024 latents, 36 layers and $\{1024, 4096, 8192, 16384\}$ input context tokens. In addition to the results shown in the main paper at stride 512, we also evaluate performance on our test set of 100 books as a function of stride (Appendix G), looking at 5 values: $\{16, 64, 128, 512, 1024\}$ (Figure 10).

We draw the reader's attention to two effects here. **First**, while the perplexity at a given context length is relatively stable for strides ≤ 128, perplexity consistently increases with stride. When using strided evaluation, the first tokens in a model's context window see a relatively small number of preceding tokens. With an evaluation stride of 1024, the first token in the context window of a 1024-context model sees only one preceding token. This property is likely responsible for the increasing gap between 1024-context models and larger-context models as the stride increases: the perplexity gain moving from the perplexity gain of the 16384- over the 1024- model is 0.8 at stride 1024, but only 0.26 at stride 16.

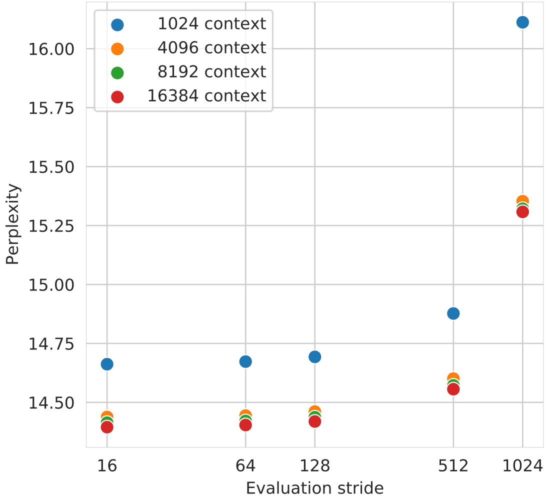


Figure 10. Perplexity results on the Books test set, from 4 different 36-layer Perceiver AR models with 1024-, 4096-, 8192- and 16384-contexts, respectively. The evaluation is done for 5 stride values.

Second, regardless of the evaluation stride, each twofold increase in context improves perplexity, but with diminishing returns: the gap between 1024 and 4096 context is consistently larger than that between 4096 and 8192 or 8192 and 16384 contexts. Perceiver AR models with the same depth and different context sizes use the same number of parameters (with the possible exception of differences in the parameters of the position encoding). We believe this effect points to the need for larger capacity models to exploit the increased information in longer contexts. Nonetheless, the overall trend suggests that larger context leads to improved results, even when using essentially the same model capacity.

C. A More Detailed Look at Perceiver AR's Internals

Like Perceiver and Perceiver IO, Perceiver AR is built on Transformer-style attention blocks. Perceivers use two types of attention: cross- and self-attention. These two types of attention share an interface—both take in two arrays and return a third—but differ in what they pass to that interface.

Zooming in: QKV attention takes in a key-value input $X_{KV} \in \mathbb{R}^{M \times C}$ and a query input $X_Q \in \mathbb{R}^{N \times D}$ (where C and D indicate number of channels). The output of QKV attention is an array with the same index (first) dimension as the query input and a channel (second) dimension determined by an output projection:

$$Q = f_Q(X_Q); K = f_K(X_{KV}); V = f_V(X_{KV}) \quad (4)$$

$$X_{QK}^{\text{pre}} = QK^T \quad (5)$$

$$X_{QK} = \text{softmax}(X_{QK}^{\text{pre}}/\sqrt{F}) \quad (6)$$

$$\text{Attn}(X_Q, X_{KV}) = X_{QKV} = f_O(X_{QK}V), \quad (7)$$

where X_{QK}^{pre} and X_{QK} are the array of pre- and post-softmax attention maps $\in \mathbb{R}^{N \times M}$, and X_{QKV} is an array $\in \mathbb{R}^{N \times D}$. The functions $f_{\{Q,K,V\}}$ are linear layers mapping each input to a shared feature dimension F and f_O is a linear layer projecting the output to a target channel dimension D, which is often the same size as X_Q's. All linear layers are applied convolutionally over the index dimension. We have omitted batch and head dimensions (in the case of multi-headed attention) for readability.

In Perceiver AR attention blocks, QKV attention is followed by a two-layer MLP with a squared ReLU nonlinearity following the first layer. The full module has the following structure:

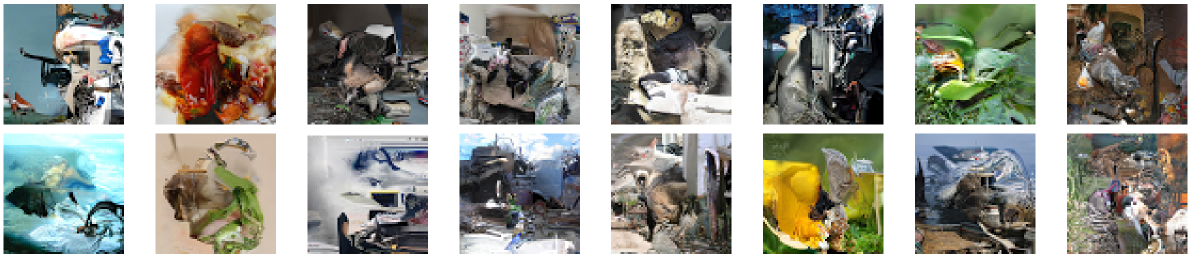


Figure 5. Full batch of generated samples from the model trained on ImageNet using **16 latents** during inference.

$$X_{QKV} = \text{Attn}(\text{layerNorm}(X_Q), \text{layerNorm}(X_{KV})) \quad (8)$$

$$X_{QKV} = X_{QKV} + X_Q \quad (9)$$

$$X_{QKV} = X_{QKV} + \text{MLP}(\text{layerNorm}(X_{QKV})), \quad (10)$$

slightly abusing notation for simplicity and to emphasize the residual structure. “Attn” refers to QKV as described above.

Zooming out: When discussed in the main text, the operations CrossAttend : $X_{KV} \times X_Q \to X_Q$ and SelfAttend : $X_{KV} \times X_Q \to X_Q$ refer to the full system of equations given in Equations (8) to (10).

These two operations differ only in that $X_Q \neq X_{KV}$ for cross-attention (with $N < M$ for the “encoder” cross-attention considered here) and $X_Q = X_{KV}$ for self-attention. Cross-attention is used to reduce the shape of an input array and self-attention to keep it the same shape. In Perceiver and Perceiver IO, the initial cross-attention’s query input X_Q is typically learned (its elements are “learned latents”), while in Perceiver AR, it is typically constructed[1] by taking the last N elements of the input array: $X_Q = X_{KV}[-N{:}, :]$, using NumPy-style indexing notation (Harris et al., 2020). In either case, using fewer latents than inputs is essential to controlling compute and memory costs while keeping long-context inputs.

Perciever AR also differs from Perceiver and Perceiver IO in the use of causally masked cross- and self-attention. In self-attention masks, all elements of X_{QK}^{pre} at indices (m', m) (queries × keys), where $m', m \in [0, M)$ and $m > m'$ are masked. In the cross-attention mask, to compensate for the fact that latents are placed at the trailing index locations, all elements of X_{QK}^{pre} at indices (n, m) (queries × keys), where $n \in [0, N), m \in [0, M)$ and $m > n + M - N - 1$ are masked. This prevents “earlier” queries from attending to “later” keys. Causal masking is implemented in attention by multiplying all masked connections in the pre-softmax attention map X_{QK}^{pre} by $-\infty$.

If we denote causally masked self- and cross-attention by CrossAttend$_{\text{cm}}$ and SelfAttend$_{\text{cm}}$, respectively, Perceiver AR (without learned latents[2]) is given by the following:

$$Z_0 \leftarrow \text{CrossAttend}_{\text{cm}}(X, X[-N{:}, :]) \quad (11)$$

$$Z_{l+1} \leftarrow \text{SelfAttend}_{\text{cm}}(Z_l, Z_l), \quad (12)$$

where Equation (12) is applied once per self-attend layer. The final output is obtained by layer-norming and projecting Z_L to the vocabulary size, followed by a softmax to produce output logits.

D. Further Related Work

In this section we describe additional background with the goal of elucidating Perceiver AR’s problem setting and method.

D.1. Efficient attention

Many recent Transformer variants seek to avoid the $O(N^2)$ memory requirements of self-attention. This is often done by introducing sparsity when computing the attention matrix – as in Sparse Transformer (Child et al., 2019), Big Bird (Zaheer et al., 2020), and Combiner (Ren et al., 2021) – or by approximating this computation at lower cost – e.g. as in Linear Transformer (Katharopoulos et al., 2020), Linformer (Wang et al., 2020), Reformer (Kitaev et al., 2020), Random-Feature Attention (Peng et al., 2021), and Performer (Choromanski et al.).

The downside of methods that use sparsity is that this sparsity must be hand-tuned or created with heuristics that are often domain specific and can be hard to tune. In contrast, our work does not force a hand-crafted sparsity pattern on attention layers, but rather allows the network to learn which long-context inputs to attend to and propagate through the

[1] With the exception of experiments on Wikitext-103, where learned latents are used.

[2] For latents, replace the second (query) input to the RHS of Equation (11) with Z_0 (or Z_{-1} if you like), which is learned.

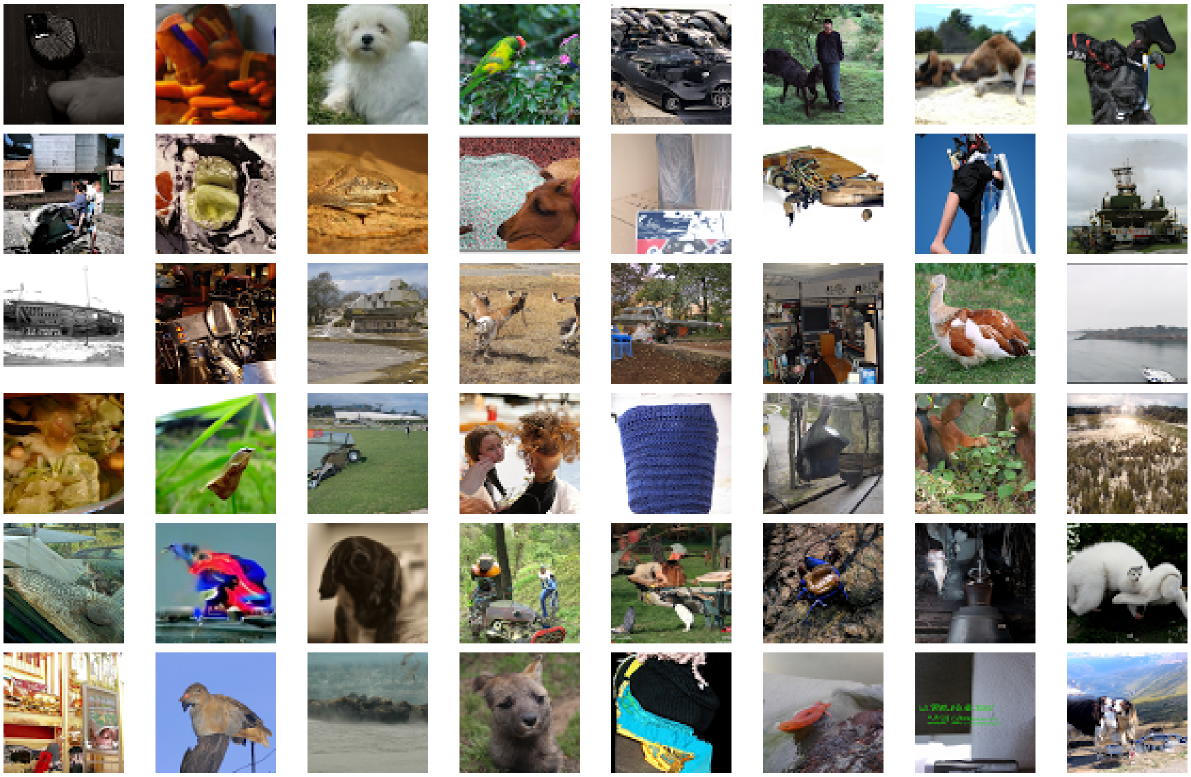


Figure 6. Full batch of generated samples from the model trained on ImageNet using **1024 latents** during inference, the same as the number of latents used during model training.

network. The initial cross-attend operation, which reduces the number of positions in the sequence, can be viewed as a form of learned sparsity.

Because Perceiver AR does not depend on hand-tuned sparsity patterns or e.g. structured dilation patterns like those used in WaveNet (van den Oord et al., 2016a), it can model arbitrarily complex dependency patterns between any of its inputs immediately after the cross-attend. Models with fixed sparsity patterns, on the other hand, typically require several layers (which depends logarithmically on the distance between points in the input array in the case of WaveNet) or precise and fragile conjunctions of input receptive fields (in the case of hand-tuned sparsity patterns) to allow a given set of points to interact. The net effect of this situation is that the effective processing used to process a given set of features is much smaller in these architectures than densely connected architectures like Perceiver AR.

D.2. Other efficient architectures

Memory-based models. Longer effective context size can also be achieved by reducing the compute requirement on tokens that are far in the past using stop gradients (Dai et al., 2019), recurrence (Mehri et al., 2017), memory (Weston et al., 2015; Rae et al., 2019), or other forms of compression (Wu et al., 2022) to process long-term structure. These strategies typically impose bottlenecks with local structure, which limits the flexibility with which context can be exploited by a given target. Although Perceiver AR performs processing using latents that are fewer in number than the inputs, each latent is given direct access to all inputs, rather than communicating with the past through a narrow or precomputed mechanism.

Alternatives to attention. Efficiency can also be obtained using domain-tuned architectures such as lightweight or dynamic convolutions (Wu et al., 2019). The recently introduced S4 (Gu et al., 2021) is a very efficient model that avoids attention altogether while producing interesting results, but it is not yet competitive on standard datasets like WikiText-103.

Many of the insights presented in this prior work are complementary to Perceiver AR's architectural design, and we expect they can be hybridized to produce even more efficient models suitable for long-scale, general purpose model design in future work.

D.3. Input Tokenization

Another approach to reducing the memory and compute requirements of self-attention is to directly reduce the length

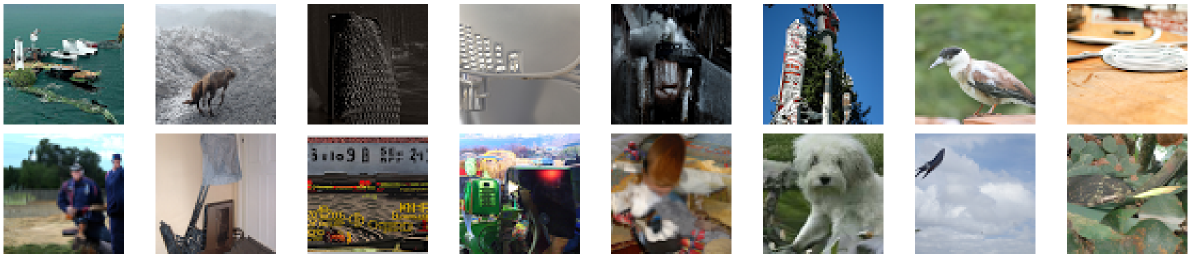


Figure 7. Full batch of generated samples from the model trained on ImageNet using **1536 latents** during inference. The same random seed was used as when generating the images with 1024 latents, which explains how the image of the white dog appears in both batches.

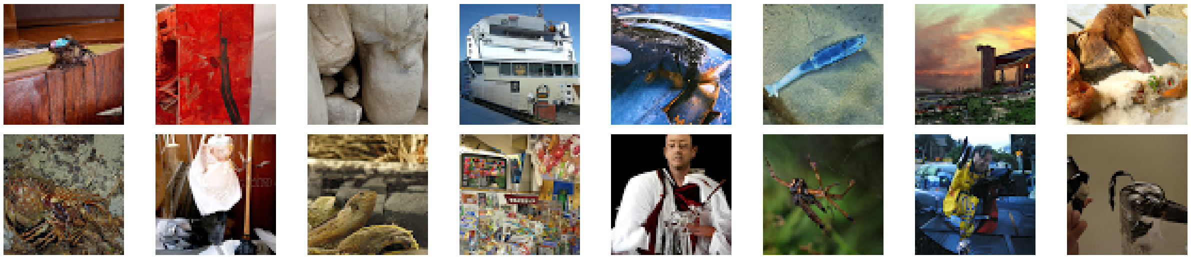


Figure 8. Full batch of generated samples from the model trained on ImageNet using **2048 latents** during inference.

of the input data by using tokenization to group multiple inputs into single tokens. Such approaches have led to excellent results on a variety of domains of interest and are often an implicit feature of the data arrays in standard datasets.

In NLP, subword chunks are often grouped together (Kudo & Richardson, 2018) based on their frequency in a training text corpus. In vision, previous work has explored using K-means to group RGB values into a single token (Chen et al., 2020) or to group individual pixels into patches (Dosovitskiy et al., 2021), sometimes followed by quantization (Ramesh et al., 2021). Others have also leveraged DCT coefficients (Nash et al., 2021) as used in JPEG to convert fixed-size images to variable-length sequences. Arguably the most widely applied tokenization method is neural vector quantization, where an encoder network is trained to map images to a spatially downsampled collection of discrete codes (van den Oord et al., 2017; Ramesh et al., 2021). Similar techniques have been developed for audio, where vector-quantized encodings of raw waveforms have effectively been used as vocabularies for end-to-end speech synthesis (Lakhotia et al., 2021; Polyak et al., 2021) and music generation (Dhariwal et al., 2020). The SoundStream codec we use in this paper builds upon these techniques by using residual vector quantization and adversarial reconstruction losses to achieve high audio fidelity with fewer discrete tokens (Zeghidour et al., 2021).

Tokenization is a broadly useful strategy, but it has its downsides. For many domains, effective tokenization schemes rely on lossy compression. Data is discarded in the tokenization process, and inputs can not be recovered exactly. Care is required to ensure that the data can be reconstructed at a fidelity adequate for the required application. Neural compression schemes such as VQ-VAE require users to train and maintain additional encoder and decoder networks, which can hinder ready application to new datasets and domains. And, perhaps most tellingly, effective tokenization is typically designed and used in a domain-specific fashion, which limits the ease of adaption and scaling to new domains. By effectively modeling long sequences, PerceiverAR can in some cases eliminate the need for tokenization and in others can reduce the need for heavy lossy compression. But in the near term, we anticipate that tokenization — of one form or another — will remain a necessary tool for incorporating more context.

E. Additional Details of the Methods

E.1. Memory Usage

The single largest source of memory usage in a Perceiver AR model is typically the attention map in the initial cross-attend layer, which results in a matrix of size $[\mathrm{heads}, \mathrm{input\_length}, \mathrm{self\_attention\_length}]$. For experiments in this paper where this matrix caused out of memory errors (Section 5.1.1), we found that processing attention heads in subgroups rather than all at once was sufficient to reduce our memory usage. Using the chunked approach in (Kitaev et al., 2020; Rabe & Staats, 2021; Jumper et al., 2021) for the cross-attend layer allows scaling input length beyond even those limits without requiring any architecture

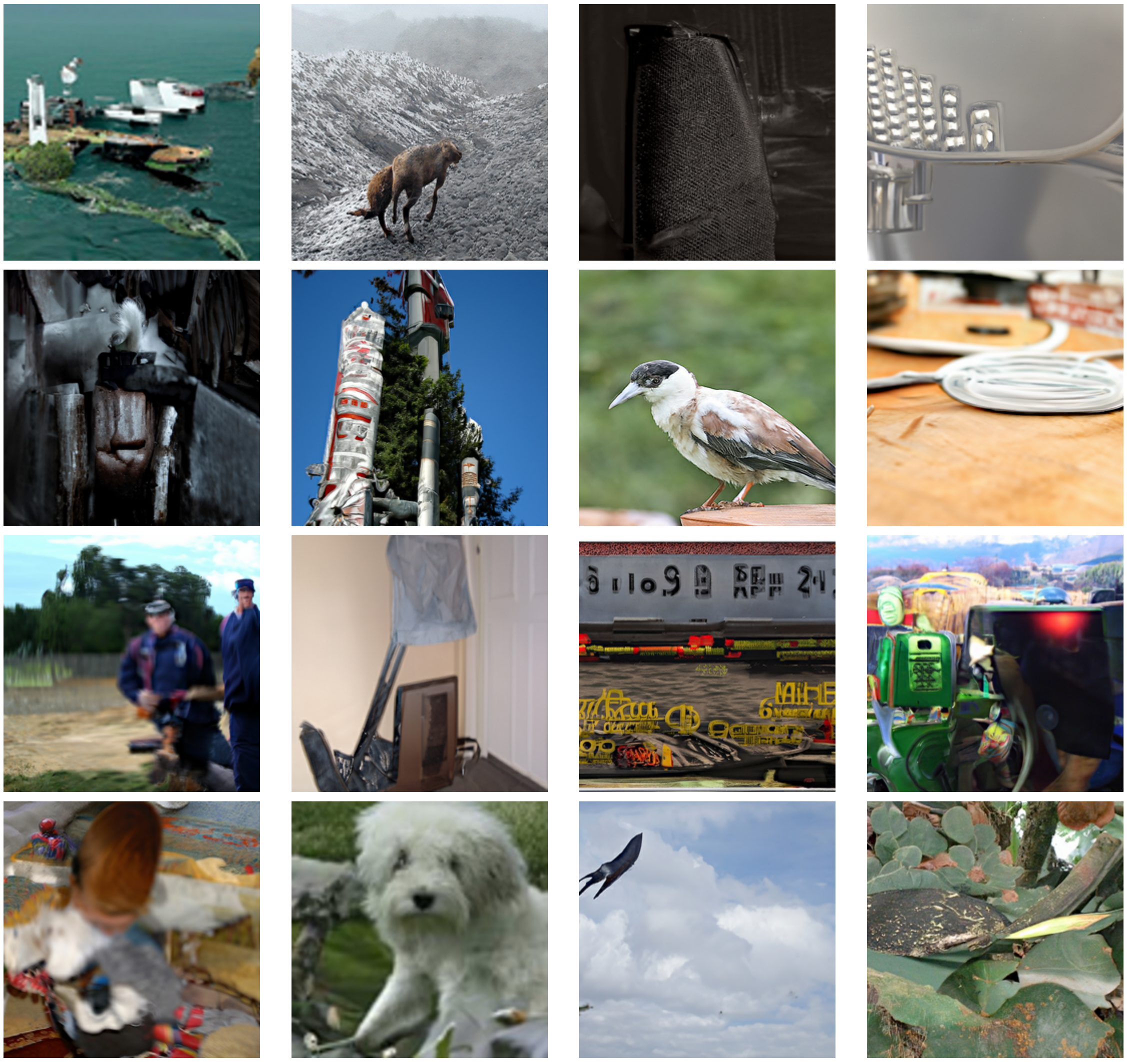


Figure 9. Images from the batch generated with 1536 latents (Figure 7) upsampled to 256 × 256 pixels using SR3 (Saharia et al., 2021).

changes or adding compute requirements for layers other than the cross-attend. These memory-saving tricks do result in reductions to training throughput (in steps per second), so we avoid them where possible.

E.2. Cross-attend Dropout

Dropout (Srivastava et al., 2014) is used by default in many Transformer implementations and is an essential tool for mitigating overfitting on small datasets. Dropout is typically imposed on linear layers after the attention softmax or within the Transformer MLP block. Perceiver AR supports this kind of dropout, but the cross-attend layer also enables interesting possibilities for dropout before the attention softmax.

We find that for some tasks, masking out positions in the initial cross-attend is an effective way of preventing overfitting. This can also be used to save memory by setting a budget for a certain number of inputs and then selecting that number of inputs randomly from the maximum input context. Because no position-specific parameters are learned for the cross-attend layer, a smaller number of inputs can be used at train time than during evaluation or inference.

Imposing cross-attend dropout can be interpreted as enforcing high sparsity at training time, but allowing less extreme sparsity at evaluation time. Because the attention layer itself is scale invariant, the uniform scaling normally imposed by dropout at train time is unnecessary.

E.3. Activation Caching for Inference

Naively sampling from a Transformer for inference can be very slow because activations for all positions must be calculated at every step. Caching the key/value activations

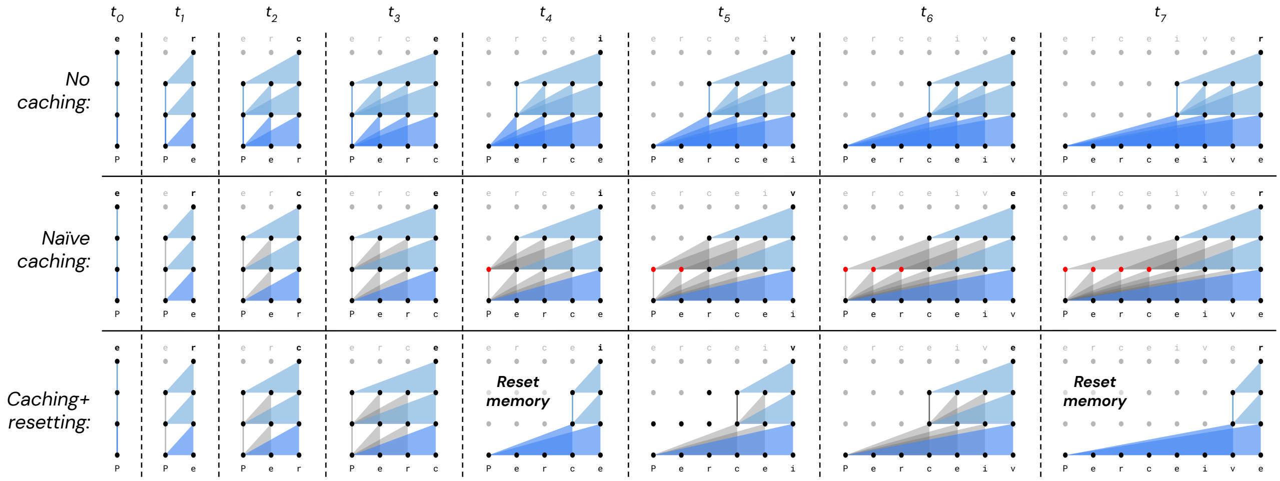


Figure 11. **Figure best viewed on a screen.** Caching at generation time allows previously computed states to be reused but introduces long-term dependencies not seen at training time when the input size and latent size differ. We illustrate this effect here for a Perceiver AR with $N = 4$ latents and an input context $M = 8$. Here, blue triangles indicate which attention operations are performed in the current step, while gray triangles indicate reused (cached) computations. The top row (no caching) matches what happens at training time. If caching is applied naively (middle row), the amount of computation per step can be greatly reduced. However, as the model is run out for more steps than there are latents, caching introduces dependencies on latents that are no longer active but have already been used to compute latents that are active. These latents are shown in red. In other words, caching previous latents allows longer-distance latents (which are no longer active) to influence the current generation, introducing long-term dependencies that were not encountered at train time. We find in practice that these long-term dependencies lead to degraded performance when caching is run out for too many steps. To avoid this problem, we cache by periodically resetting memory, allowing some computation to be reused but avoiding the introduction of long-term dependencies not encountered at training time. This strategy is described in detail in Appendix E.3.

for previously inferred positions is a common technique for improving generation speed (Vaswani et al.; Shazeer, 2019). Perceiver AR can use a similar approach, but the exact technique cannot be applied directly because the number of output positions is smaller than the number of input positions, so preserving all previous activations would result in an effective self-attention stack as wide as the number of inputs.

Transformer-XL (Dai et al., 2019) solves this problem by keeping a buffer of activations for only the last N positions. This works because the model is presented with activations for that number of previous positions during training, which is not the case for Perceiver AR. We also cannot simply restrict the buffer size to be the same as the number of targets because even when activations for a given position are expired, they have already influenced other positions within the buffer (Figure 11).

Instead, we apply a simple trick to ensure that no cached activations are influenced by positions beyond what was seen at training time. We use a fixed activation buffer the same width as the self-attention stack at train time. When the buffer is full, we do a full forward pass without any cached activations for the next position, but using half the number of latents. The activations from that pass are saved in the buffer, leaving it half full. Inference then proceeds until the buffer is full again. The activations from the cross-attend layer do not require this trick unless inference will extend beyond the length of the inputs used at train time.

We find that these occasional full forward passes add minimal overhead, and the speed gains from using cached activations are still significant. We performed a test inferring a single image with a sequence length of 12,289 using a model with 1024 latents (see Section 5.2 for details) on a single TPUv3 core to compare speeds. Inference without any caching took 7.93 minutes. With caching the same task took 3.68 minutes, less than half the time.

E.4. Varying Compute at Test Time

Figure 12 illustrates how the number of latents can be changed at test time without changing either the trained parameters of the model or the model's input context length. In Section 5.2.1 we discuss how this possibility can be used to scale up or down compute requirements and output quality.

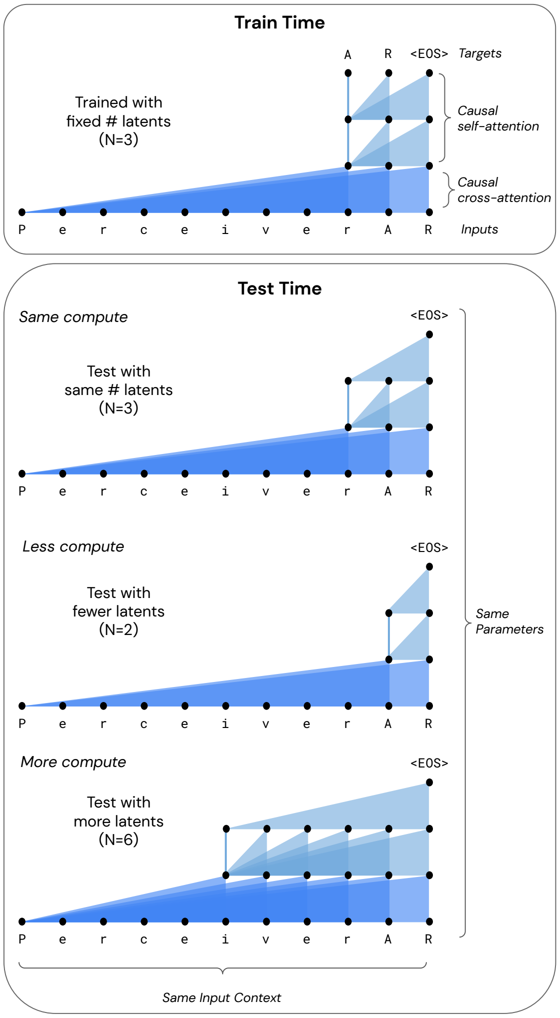


Figure 12. Because Perceiver AR decouples input length from the width of the self-attention stack, the number of latents can be different at train time and test time. This does not require additional training because no per-position parameters are learned in the self-attention stack. Section 5.2.1 discusses how this possibility can be used to scale up or down compute requirements and output quality.

F. Training Details

F.1. Common

Unless otherwise stated, models were trained with the following configuration.

We use the Adam optimizer (Kingma & Ba, 2015) as implemented in the Optax framework (Hessel et al., 2020) with $b1 = 0.1$, $b2 = 0.999$, $eps = 1\mathrm{e}{-}8$, a base learning rate of $3\mathrm{e}{-}4$, and a 10k step linear warmup. To reduce memory usage, we used ZeRO Stage 1 optimizer state partitioning (Rajbhandari et al., 2020).

For training stability, we used a global max norm of 1.0. We also added an additional loss term $z\_loss * log(z)^2$ with $z\_loss = 1\mathrm{e}{-}4$, as used in training the T5 family of models (Raffel et al., 2020).

Both the input embedding and latent vectors were size 1024, and 16 attention heads were used for the initial cross-attend and within the self-attention stack. Within the MLP layers of the cross-attend and self-attention stack, the input dimensionality is projected to 4x its size and Squared ReLU activations (So et al., 2021) were used. We used a cross-attend dropout probability of 0.1.

Training and evaluation were done on either TPUv2 or TPUv3 clusters.

F.2. Rotary Position Encodings

We encode the position of tokens using rotary position encoding (Su et al., 2021). With this method, rotation matrices (built from sine and cosine functions, just as with sinusoidal/Fourier position encodings) rotate pairs of dimensions in each of the key and query heads to reflect the absolute position of the key or query in the sequence. When used to compare a given key and query pair, these rotations produce attention weights that reflect only the relative distance between tokens. The result is a memory-efficient relative position mechanism.

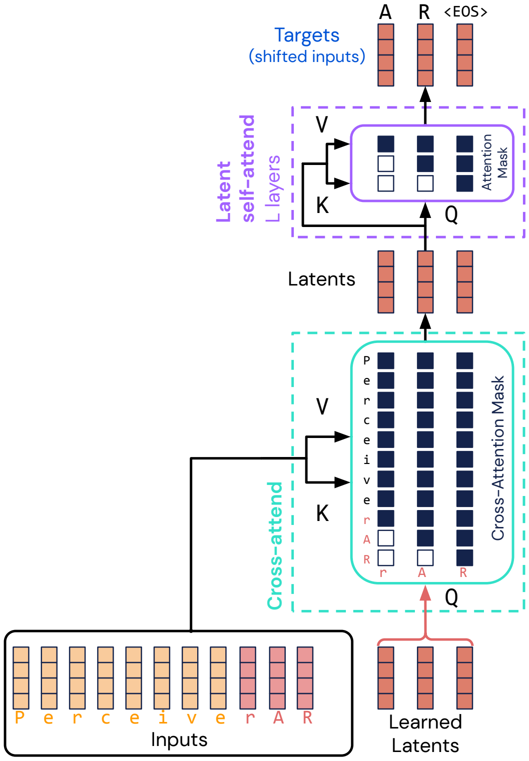


Figure 13. Variant of Perceiver AR where we feed learned input latents instead of the input embeddings directly. This is used for our Wikitext-103 experiments as we noticed it helped reduce overfitting.

Intuitively, this mechanism represents position using a strategy somewhat like that used by analogue clock hands. Like a clock hand, each pair of dimensions (sine/cosine) in the rotary position encoding is responsible for indicating position at some frequency (second, minutes, hours, etc.). When we have multiple clock hands (multiple dimensions within each token vector), each of which moves at a different frequency, we can precisely tell both the current time (q/k position) and we can resolve it at different resolutions, depending on what's needed for a given task.

We found in early experiments that rotating a fraction of channel dimensions led to better results, which we discovered has also been noticed (the first 50%) (Wang, 2021). Fractional rotation of this kind reduces the fidelity with which we encode position information (because it results in fewer frequency bands). The result of fractional rotation is that only some channel dimensions are modulated by the relative position between a query and key. This may encourage the network to exploit both position-dependent and position-agnostic relationships between queries and keys when computing attention weights. Prior work characterizing the effect of position on attention weights suggests that many common position encoding strategies bias the network towards attending to recent tokens and fractional rotation may mitigate this effect. Fractional rotation also but makes the rotation computation significantly cheaper, as it requires fewer matrix multiplies.

F.3. Copy Task

The copy task was trained with 1024 latents in a 6-layer self-attention stack. For position encoding, we used fixed sinusoidal embeddings to denote absolute position, as described in the original Transformer paper (Vaswani et al., 2017). There were 4096 sequences per batch.

To reduce the instantaneous memory requirements created by attending to the input sequence of length 131,072, the 16 cross-attention heads were split into 4 groups of 4 and computed separately (see Appendix E.1).

The first 1K steps were a linear learning rate warmup and the remaining followed a cosine learning rate decay schedule.

F.4. ImageNet 64 × 64

The model trained on ImageNet 64 × 64 has 770.1M parameters. Training proceeded for a total of 750k steps. After an initial 10k step linear warmup, a constant learning rate of 3e−4 was used until the final 50k steps, which used a cosine decay to 0.

F.5. Wikitext-103

We train 18-layer Perceiver AR models with 1,024 latents, adaptive inputs embeddings (Baevski & Auli, 2018), and with increasing context length from 1,024 to 8,192 tokens. One key difference to other experiments is that we use learned input latents rather than the input embeddings as illustrated in Figure 13. We observe this variant to help reduce overfitting. We also apply cross-attend dropout (Appendix E.2) with p=0.15 to the context tokens for which we

SoundStream bitrate	Context (32k)	Context (8k)	Test (32k)	Test (8k)	Validation (32k)	Validation (8k)
12kbps	27.2s	6.8s	2.31	2.25	2.28	2.27
18kbps	18.4s	4.6s	2.52	2.53	2.51	2.52
22kbps	14.8s	3.7s	2.55	2.60	2.55	2.60

Table 13. Perceiver AR negative log-likelihood results on SoundStream audio generation, from two models with context lengths of 8192 (8k) and 32768 (32k), respectively.

predict the next token.

The Transformer-XL baseline used a memory length of 384 at train time and 1600 at eval time, for a full effective context length of 2624. We used a memory dropout rate of 0.25.

Perceiver AR models trained on Wikitext-103 have the following parameter counts: 356.5M (1024 context), 357.7M (2048 context), 359.8M (4096 context), 364.0 (8192 context). Note that the parameter count increases slightly with increasing context length because of the use of absolute position encodings on Wikitext-103. The baseline Transformer-XL has 285.2M parameters.

F.6. PG-19

Both Perceiver AR models trained on PG-19 use 974.6M parameters.

F.7. Books

All Books models reported in Table 7 have 498.9M parameters.

Perceiver AR models reported in Table 8 all have 498.9M parameters. Compute matched Transformer-XL models have the following parameter counts, by number of layers: 346.6M (23), 360.3M (24), 373.9M (25), 414.8M (28).

Perceiver AR models reported in Table 9 have the following parameter counts, by context length and number of layers: 826.4M (1024, 62L), 813.8M (4096, 61L), 801.2M (8192, 60L), 750.8M (16384, 56L). The 42-layer Transformer-XL with matched compute has 605.8M parameters.

We use the same memory settings for Transformer-XL baselines as on Wikitext-103: all Transformer-XL models have a full effective context length of $1600 + 1024 = 2624$.

F.8. Music Generation Tasks

All models use a value of 0.7 for the cross-attend dropout described in Appendix E.2. All but the MAESTRO SoundStream-task models apply rotation to 25% of the attention dimensions. Additionally, the models trained on the piano dataset and MAESTRO SoundStream tasks use a learning rate of $2\mathrm{e}{-4}$. The ones trained on the symbolic MAESTRO data use a learning rate of $1\mathrm{e}{-4}$, 1 initial cross-attend head and 4 heads within the self-attention stack. The model trained on the piano transcription dataset uses 0.1 for cross-attend dropout and a 0.25 dropout rate on the latents. Finally, this model pre-embeds the query inputs using a 3-layer MLP with GELU (Hendrycks & Gimpel, 2016) activations.

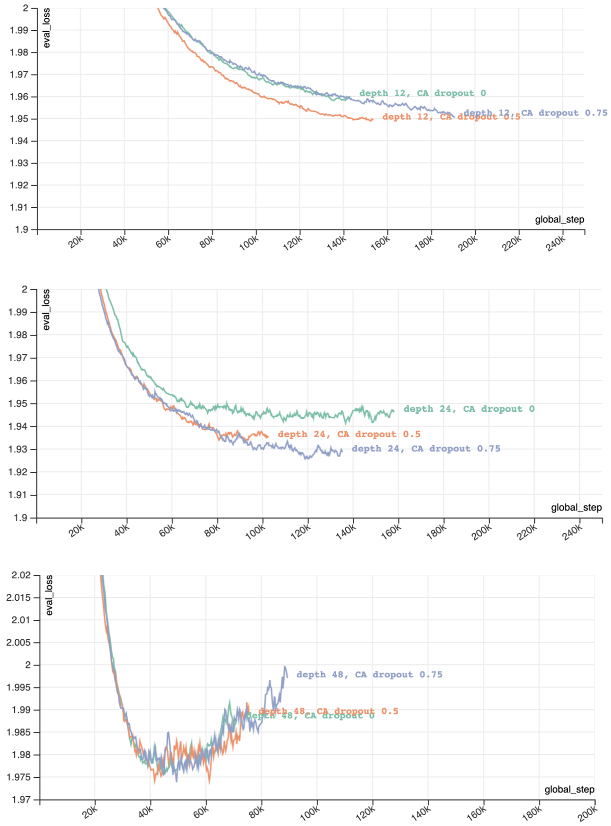


Figure 14. Validation loss on MAESTRO v3 as a function of the cross-attend dropout applied to the model. Each value $\{0, 0.5, 0.75\}$ is applied at 3 different model depths $\{12, 24, 48\}$.

Audio from MAESTRO v3 was first processed using the SoundStream 12kbps codec. This bitrate of 12kbps corresponds to a vocabulary of 1024 tokens (10 bits), 24 tokens/frame, and 50 frames/second. This reconstructs audio while reasonable quality, compressing each second of audio (16k wave points) into 1.2k sequential tokens. Our trained model has an input context of length 8192 (∼6.8 seconds) and uses 1024 latents in 12 self-attention layers. We also train and evaluate this configuration using higher-bitrate codecs—18kbps and 22kbps, with 36 and 44 tokens/frame respectively.

G. Evaluation Details

As discussed in Shortformer (Press et al., 2021), there is a quality vs. speed tradeoff when evaluating long sequences with regard to stride. A stride of 1 (maximum overlap) is the slowest but highest quality, and a stride equal to the input length (no overlap) is the fastest but lowest quality. Because Perceiver AR decouples the number of targets from the number of inputs, our stride options range from 1 to the number of latents. We found that a stride of half the number of latents gave a good balance between speed and quality, and that is what we used for all the evaluations in this paper, unless otherwise mentioned. For a related set of issues for inference, see the discussion in Appendix E.3.

H. Dropout Ablations

We study the effects of applying different forms of dropout to Perceiver AR, at several model depths when training on the MAESTRO v3 symbolic dataset.

Figure 14 illustrates the behaviour of the model when varying the amount of cross-attend dropout applied to it. The model has an input context of 4,096 and a post-attention dropout value of 0.5—both hyperparameters are kept constant across all runs. This type of dropout appears less useful when increasing model depth, up to having no effect at all during training for the 48-layer model.

For the second experiment, we vary the amount of (post-attention) dropout in the same settings as previously described, while keeping a constant value of 0.7 for the cross-attend dropout. Figure 15 shows that higher dropout rates become more useful at bigger depths—while at depth 12, applying 0.75 dropout actually hurts model training, this becomes beneficial at depth 48, where applying no dropout leads to quick overfitting.

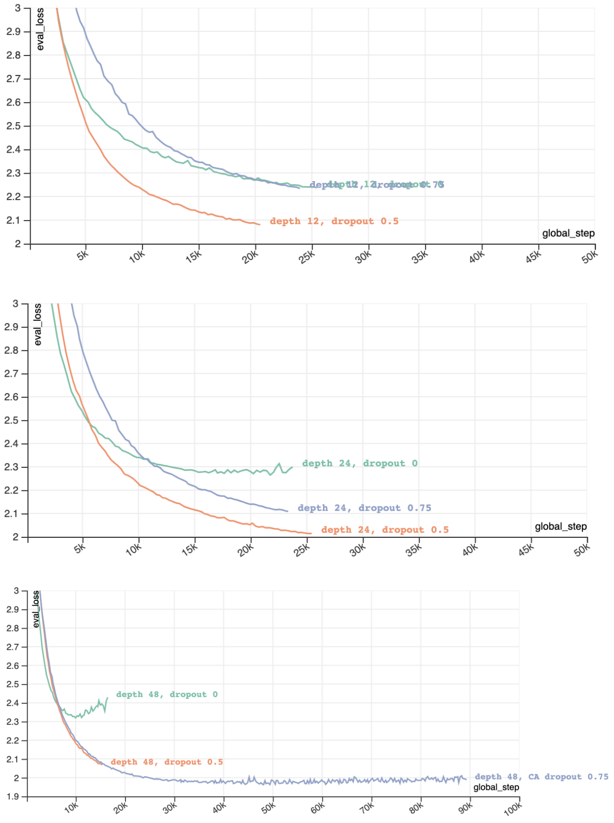


Figure 15. Validation loss on MAESTRO v3 as a function of the post-attention dropout applied to the model. Each value $\{0, 0.5, 0.75\}$ is applied at 3 different model depths $\{12, 24, 48\}$.

I. MAESTRO SoundStream

In Table 13, we compare results on all MAESTRO SoundStream tasks obtained from models that were trained on contexts of 8192 and 32768 tokens, respectively. While a smaller context length yields better results at the lowest bitrate, the 32k-context model obtains a significantly lower negative log-likelihood than the 8k- one on SoundStream 22kbps. Furthermore, the gap between corresponding NLLs widens as the bitrate increases. This result suggests that a shorter context is less and less useful as we attempt to produce higher-fidelity audio. The 32k context models reported here have the following parameter counts: 366.3M (12kbps), 391.5M (18kbps), 408.3M (22kbps). The 8k context models reported here have the following parameter counts: 215.2M (12kbps), 240.4M (18kbps), 257.1M (22kbps).

65536-context SoundStream models reported in Table 12 have the following parameter counts: 668.6M (12kbps), 693.8M (18kbps), 710.6M (22kbps).