Hierarchical Transformers Are More Efficient Language Models

Piotr Nawrot[*1], **Szymon Tworkowski**[*1], **Michał Tyrolski**[1], **Łukasz Kaiser**[2], **Yuhuai Wu**[3], **Christian Szegedy**[3], **Henryk Michalewski**[3]
[1]University of Warsaw, [2]OpenAI, [3]Google Research
{p.nawrot99, szy.tworkowski, michal.tyrolski, lukaszkaiser}@gmail.com,
{yuhuai, szegedy, henrykm}@google.com

Abstract

Transformer models yield impressive results on many NLP and sequence modeling tasks. Remarkably, Transformers can handle long sequences, which allows them to produce long coherent outputs: entire paragraphs produced by GPT-3 or well-structured images produced by DALL-E. These large language models are impressive but also very inefficient and costly, which limits their applications and accessibility. We postulate that having an explicit hierarchical architecture is the key to Transformers that efficiently handle long sequences. To verify this claim, we first study different ways to downsample and upsample activations in Transformers so as to make them hierarchical. We use the best performing upsampling and downsampling layers to create Hourglass - a hierarchical Transformer language model. Hourglass improves upon the Transformer baseline given the same amount of computation and can yield the same results as Transformers more efficiently. In particular, Hourglass sets new state-of-the-art for Transformer models on the ImageNet32 generation task and improves language modeling efficiency on the widely studied enwik8 benchmark.

1 Introduction

Transformer models (Vaswani et al., 2017) are capable of solving many sequence modeling tasks, including classical NLP tasks (Devlin et al., 2019), summarization (Zhang et al., 2020), language modeling (Radford et al., 2019; Brown et al., 2020), code generation (Chen et al., 2021), or even music generation (Huang et al., 2018; Dhariwal et al., 2020) and image generation (Parmar et al., 2018; Chen et al., 2020; Ramesh et al., 2021). One compelling feature of Transformers is their ability to handle long contexts given as part of the input. This is particularly visible in tasks where the output depends on parts of the context that may not be close-by in the generated sequence, like in summarization, where the summary may need to refer to information scattered across the context, or in large-scale image generation, where pixels belonging to the same object may be far apart in the generation order. Transformers excel at such tasks thanks to self-attention, and they are used with longer and longer contexts.

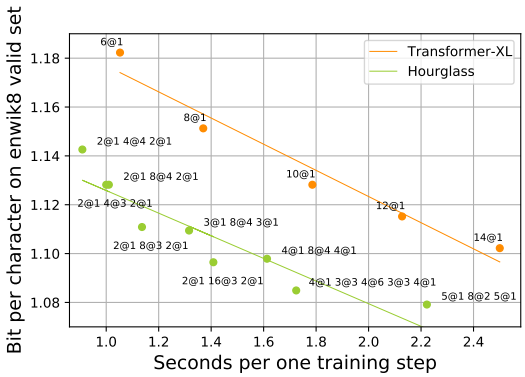

Figure 1: Bits-per-character vs. training cost for baseline (orange) and hierarchical Transformers (green). We observe significant perplexity improvements on enwik8 over the vanilla Transformer–XL baseline, see text for details.

The ability of Transformers to handle long contexts comes at a price: each self-attention layer, at least in its original form, has complexity quadratic in the length of the context. When a stack of n Transformer layers is used, both memory and time complexity is equal to $O(L^2 n)$ where L is a sequence length and n number of decoder blocks. Due to this limitation, vanilla transformers are infeasible to train on tasks with very long input sequences, for instance, on high-resolution images. This issue has been studied extensively, and a number of techniques were introduced that modify attention mechanism without changing overall transformer architecture (Child et al., 2019; Roy et al., 2020; Ren et al., 2021). These sparse attention mechanisms reduce the complexity of self-attention

[*]Equal contribution. Order determined by coin toss.

Findings of the Association for Computational Linguistics: NAACL 2022, pages 1559 - 1571

but still force the model to operate on the sequence of the same length as the input.

For generative Transformer models, operating at the original scale of the input sequence is necessary, at least in the early and final layers, as the input must be processed at first and generated at the end (Section 4.3). But forcing the models to operate at this granularity throughout the layer stack has both fundamental and practical shortcomings:

- Fundamentally, we aim for the models to create high-level representations of words, entities, or even whole events – which occur at a very different granularity than single letters that the model receives on input.

- On the practical side, even layers with linear complexity can be slow and memory-intensive when processing very long sequences.

To alleviate these issues, we propose to change the Transformer architecture to first shorten the internal sequence of activations when going deeper in the layer stack and then expand it back before generation. We merge tokens into groups using a shortening operation (Section 2.1) and so reduce the overall sequence length, and then up-sample them again combining with the sequence from earlier layers (Section 2.3), The first part is analogous to the Funnel-Transformer architecture (Dai et al., 2020), and the whole architecture takes inspiration from U-Nets (Ronneberger et al., 2015). In contrast to both these architectures, the model we present is autoregressive, which is harder to ensure in hierarchical models than in vanilla Transformers.

The resulting model – which we call *Hourglass* – is an autoregressive Transformer language model that operates on shortened sequences. It yields significant performance improvements for different attention types (Fig. 6,7). We tested Hourglass with Transformer-XL (Dai et al., 2019) and Reformer (Kitaev et al., 2020) blocks on enwik8 dataset. In both cases, it is not only better in terms of perplexity, but it is faster and uses less memory during training. We also propose a regularization technique for hierarchical Transformers called *shorten factor dropout* which improves perplexity upon baselines trained with fixed shorten factor (see Section 4.1). Finally, Hourglass achieves the new state-of-the-art among Transformer models for image generation of ImageNet32 (see Tab. 3).

2 Model

Standard self-attention mechanism uses full token-level sequence representations. In the Hourglass, we bring efficiency to the model by utilizing *shortening*, which allows us to use the Transformer layers on inputs with significantly smaller lengths. A high-level overview of our proposed model architecture is shown in figures 2 and 3.

Attention type in the vanilla layers and shortened layers is a configurable parameter. By default we use relative attention defined in Transformer-XL (Dai et al., 2019). Any attention module can be used - we show significant efficiency gains when applying Hourglass also for LSH (Kitaev et al., 2020) attention (see Section 3.2 and Fig. 7).

2.1 Methods of shortening the input sequence

Shortening can be defined as any function S that accepts a tensor x of shape (l, d) and returns a tensor x' of shape $(\frac{l}{k}, d)$, where k is a hyperparameter called *shorten factor*.

A simple shortening method is 1D average pooling with stride k and pool size k, applied along the sequence dimension l. Another way of shortening is what we will further call *linear pooling* (l and d denote sequence length and d_{model}):

Algorithm 2 LinearPooling

$x' \leftarrow Reshape(x, (\frac{l}{k}, k \cdot d))$
$x' \leftarrow LinearProjection(x')$

Shortening can be also performed by attention, as was introduced in (Dai et al., 2020): $x' = S(x) + Attention(Q = S(x), K = V = x)$ where S is shortening function, originally $S = AvgPool$. Directly after this attention operation, a position-wise feed-forward with a residual is performed, so that these two layers form a Transformer block (Vaswani et al., 2017). In this work we also try $S = LinearPool$ and find it more effective on image tasks (see Tab. 8).

2.2 Shortening and autoregressive property

Information leaks Shortening interferes with the standard causal masking used in Transformer decoders. Namely, in any shortened representation by a factor of k each shortened token contributes to predicting up to the next k tokens in the finest scale, that is if e is the shortened sequence and x is the sequence on the finest scale, e_0 is not only used

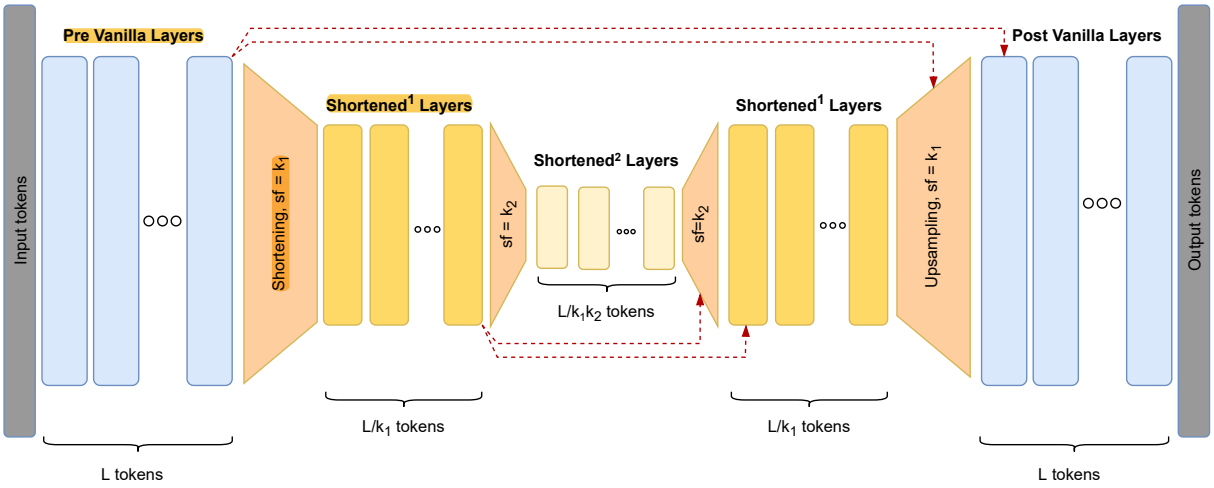

Figure 2: Hourglass - a high-level architecture overview. The arrows denote residual connections.

to generate x_0; in fact, the same embedding is used to generate tokens $x_0, ..., x_{k-1}$.

Therefore, we need to guarantee that e_0 and any other e_i cannot access information about tokens they will implicitly predict. To ensure that, we apply another shift right by $k-1$ tokens, directly before any shortening by a factor of k (Fig. 4). The shift is the smallest that does not cause an information leak (see Fig. 5 for an example of a shifting that leads to a leak). We included a more detailed analysis of this fact in the Appendix (Section A.2).

Reduced expressivity Let us consider an Hourglass model with shortening by a factor of k and no transformer blocks operating on the finest scale (that is, a model without vanilla layers).

In this situation
$P(x) = \prod_{i=0}^{n-1} P(x_i|e_0, ..., e_{\lfloor \frac{i}{k} \rfloor}) = \prod_{i=0}^{n-1} P(x_i|x_0, ..., x_{\lfloor \frac{i}{k} \rfloor \cdot k-1})$
because for predicting x_i we combine the processing done on shortened representations e with token-independent operations. This means token x_i is generated independently from the tokens $x_{\lfloor \frac{i}{k} \rfloor \cdot k}, ..., x_{i-1}$. This situation is detrimental to the model's capabilities, though including at least one vanilla layer solves this issue. In the Appendix we provide a detailed example illustrating this problem (Section A.1).

2.3 Upsampling methods

Upsampling is a crucial part of the Hourglass architecture since we need to convert shortened representations back to the full token-level sequence in order to perform language modeling.

A method proposed in (Dai et al., 2020) is repeating each shortened vector *shorten factor* times. This method is computationally efficient, but it does not distinguish tokens with respect to position inside the group.

Another method is *linear upsampling* which works analogously to linear pooling – it projects vectors of shape $(\frac{l}{k}, d)$ to $(\frac{l}{k}, k \cdot d)$ and then reshapes to l vectors, each of dimension d. This method is fast and allows to project shortened embeddings differently for each position in the group. This happens because the $(k \cdot d) \times d$ projection matrix can be thought of as k separate $d \times d$ matrices, one per each position.

We also investigated a method which we further call *attention upsampling*. It is similar to attention pooling (Dai et al., 2020) and to the aggregation layer from (Subramanian et al., 2020). It works as follows: $x = U(x, x') + Attention(Q = U(x, x'), K = V = x')$ where x are embeddings from just before the shortening, x' are final shortened embeddings and U is an arbitrary upsampling function. After the attention operation there is also a residual with a feed-forward layer.

Linear upsampling learns a fixed pattern that is the same for each shortened token. Attention upsampling has the advantage of being content-based – each token can extract relevant information from the shortened embeddings. We set $U(x, x') = x + LinearUpsampling(x')$ which allows to explicitly inject group-level information into the attention queries. We experimentally show that variants of attention upsampling lead to the best results for our model across different datasets (see Tab. 7).

Algorithm 1 HourglassLM

procedure HOURGLASS(x, $[k, ...s\_factors]$)
  $x \leftarrow PreVanillaLayers(x)$
  $x' \leftarrow Shortening(ShiftRight(x, k-1), k)$

  if EMPTY($s\_factors$) **then**
    $x' \leftarrow ShortenedLayers(x')$
  else
    $x' \leftarrow$ HOURGLASS($x', s\_factors$)
  end if
  $x \leftarrow x + Upsampling(x, x', k)$
  $x \leftarrow PostVanillaLayers(x)$
  return x

Figure 3: The architecture starts with *pre vanilla layers* – a stack of Transformer blocks operating on the full token-level sequence. After them we insert *shortening layer* where k is the *shorten factor* parameter (Fig. 4). The sequence is shifted right before shortening to prevent information leak (Fig. 5). Then we recursively insert another Hourglass block operating on k times smaller scale. On the final level of shortening, we apply *shortened layers* – Transformer blocks operating on the smallest scale. *Upsampling layer* brings the resulting activations x' back to the original resolution. After upsampling and residual, the activations are processed by token-level *post vanilla layers*.

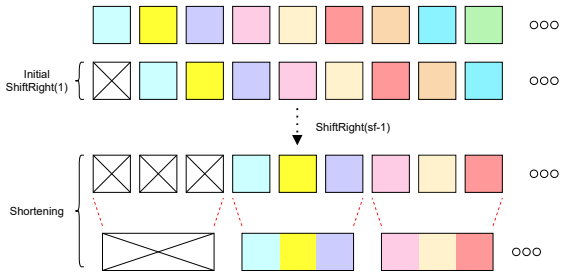

Figure 4: An overview of our shortening approach. Different colors denote token positions. Initially, we shift right by one, which is a standard step in TransformerLM. Then, just before performing shortening, we additionally shift the tokens right by *shorten factor* -1 to preserve the autoregressive property of the model.

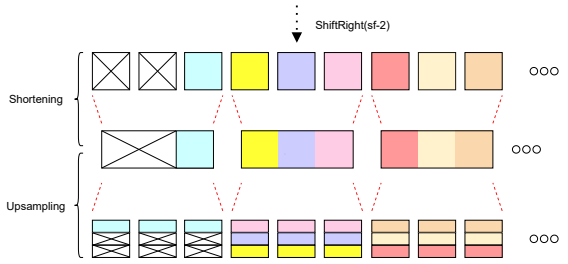

Figure 5: An example of information leak. If the shift right factor is too small, after upsampling the knowledge from the next tokens leaks to previous ones violating autoregressiveness and making decoding impossible.

3 Experiments

In this section, we present experimental results of Hourglass. We start with a quick analysis of time and memory complexity of the approach (Section 3.1). Then we investigate the efficiency gains of applying Hourglass to Transformers with different attention types (Section 3.2). Finally, we use Hourglass with relative attention parametrization from Transformer-XL (Dai et al., 2019), evaluate it on three language modeling tasks, and compare the results with other models. (Sections 3.3, 3.4)

To show cross-domain generalization of our method, we train our model on one dataset related to Natural Language Processing and two from the Computer Vision field.

To ensure consistency in presenting configurations of our model, we introduce a notation describing hierarchy of our architecture: $(N_1@f_1, \ldots, N_k@f_k)$ where each entry $(N_j@f_j)$ means N_j layers shortened by factor f_j.

Our model implementation is open source.[1]

[1] github.com/google/trax/blob/master/trax/models/research/hourglass.py

3.1 Computational cost analysis

In vanilla Transformers, the number of parameters can indicate the computation required to train the model. This is not true for Hourglass – for instance, it can have 128 layers operating on a sequence shortened by 32 and still fit into the memory of a single GPU. A weak correlation between true Hourglass' computational cost and its number of parameters can be observed in Table 1.

Hourglass achieves the biggest speedup with the standard $\mathcal{O}(l^2)$ attention. In that case, a single shortening by a shorten factor k reduces the complexity to $\mathcal{O}(\frac{l^2}{k^2})$ so by a factor of k^2. For more recent linear-time attention mechanisms (Katharopoulos et al., 2020; Choromanski et al., 2021) the reduction would be smaller – but still by a factor of k. Feed-forward layers also have linear complexity so shortening reduces it by a factor of k.

In Table 1 we show an empirical efficiency comparison between Hourglass and Transformer-XL.

Hierarchy	BPC	GB	Speed	#Param
6@1 (Baseline)	1.182	4.53	0.95	21M
2@1 1@3 2@1	1.163	4.41	1.11	24M
2@1 4@4 2@1	**1.143**	4.41	1.10	34M
8@1 (Baseline)	1.151	5.75	0.73	28M
2@1 4@3 2@1	1.128	4.88	1.00	34M
2@1 8@4 2@1	1.128	4.98	0.99	48M
2@1 1@2 4@4 1@2 2@1	1.115	4.69	0.86	48M
2@1 8@3 2@1	**1.111**	5.50	0.88	48M
10@1 (Baseline)	1.128	6.99	0.56	34M
3@1 8@4 3@1	**1.109**	6.14	0.76	55M
12@1 (Baseline)	1.115	8.12	0.47	41M
4@1 8@4 4@1	1.098	7.20	0.62	62M
2@1 16@3 2@1	**1.096**	5.89	0.71	75M
14@1 (Baseline)	1.102	9.35	0.40	48M
5@1 8@2 5@1	**1.079**	9.57	0.45	69M

Table 1: Efficiency comparison between Hourglass variants and Transformer-XL baseline on enwik8 – we report validation set perplexity (BPC), running memory (GB) and number of training steps per second (Speed). We observe significant perplexity gains over the baseline for a matching computation cost. It is also visible that for Hourglass the number of model parameters (#Param) correlates poorly with true computational cost.

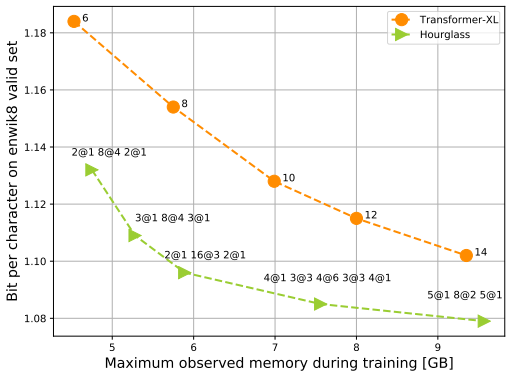

Figure 6: Comparison between Transformer-XL baseline and Hourglass on Enwik8 valid set w.r.t. maximum memory used during training. All models are trained for 200k steps with the same hyperparameters.

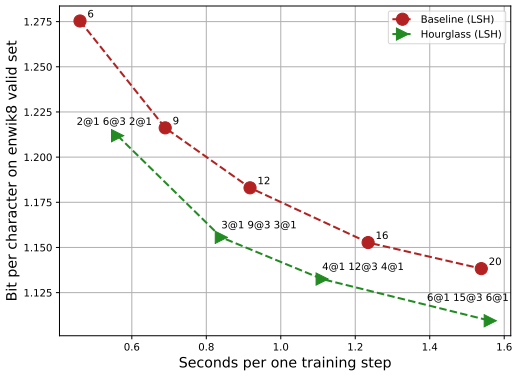

Figure 7: Comparison between Reformer baseline and Hourglass, both with LSH attention, on Enwik8 valid set w.r.t. cost of one training step in seconds.

3.2 Impact of Hourglass

To demonstrate the efficiency of Hourglass, we measured how computational cost decreases and perplexity improves, purely adding the technique to Transformer-XL (Dai et al., 2019) and Reformer (Kitaev et al., 2020) backbones (results depicted in Figures 6 and 7, respectively).

In both cases, models are implemented under the same codebase and the only difference between Hourglass and its corresponding baseline is the usage of shortening and upsampling layers. We show that by incorporating a single shortening of the input, we can train larger models with the same memory requirements and training speed and achieve better perplexity than baselines.

3.3 Enwik8

Enwik8 (Mahoney, 2011) is a byte-level language modeling benchmark containing the first 100M bytes of unprocessed English Wikipedia text, split into 90M train, 5M valid, and 5M test sets.

Similarly to (Dai et al., 2019) and (Beltagy et al., 2020), we evaluate our model on the test set, splitting it into overlapping sequences of size $l = 4096$ with a step size of 128 and calculate the test loss only over the last 128 tokens. With a $(4@1, 8@3, 4@1)$ hierarchy, $d_{model} = 768$, $d_{ff} = 3072$ and 8 heads, we reach **0.98** test bits-per-character.

3.4 Image Generation

We use datasets introduced in (van den Oord et al., 2016a) which are downsampled versions of the popular ImageNet. In the autoregressive image generation setup, they consist of respectively $\mathbf{32 \times 32 \times 3}$ and $\mathbf{64 \times 64 \times 3}$ tokens, corresponding to RGB channels, per image. As the only preprocessing step we flatten the images.

3.4.1 ImageNet32

For our main result the following hierarchy is used: $(3@1, 24@3, 3@1)$. We use $d_{\text{model}} = 512$, $d_{\text{ff}} = 2048$, 8 attention heads and 0.01 dropout rate. With this configuration we achieve **3.741** bits/dim, yielding the new state-of-the-art among autoregressive (Transformer-based) models on this dataset, compared to the previous state-of-the-art of 3.758 bpd by (Ho et al., 2019).

Enwik8	#Param	BPC
Transformer-XL (2019) 24L	277M	0.99
Hourglass	146M	0.98
Adaptive-Span (2019) 24L	209M	0.98
Transformer-LS (2021)	110M	0.97
Feedback Transformer (2021)	77M	0.96
Expire-Span (2021) 24L	277M	**0.95**

Table 2: **Enwik8 Results.** We report bits-per-character (BPC) on the test set and number of model parameters. Hourglass applied to Transformer-XL significantly outperforms its baseline. Our technique could be also used with other more performant attention methods which we leave for future work.

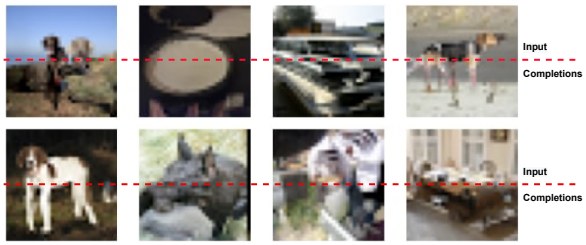

Figure 8: Examples of our model completions, where bottom half of each image was generated by our model, prompted by the upper half.

3.4.2 ImageNet64

The sequence length that our model can handle is limited mainly by the computational complexity of used attention module. We replace relative attention in vanilla layers by LSH attention (Kitaev et al., 2020), which allows us to handle **12288**-long sequences. To achieve relative attention parametrization, the LSH attention is combined with rotary positional embeddings (Su et al., 2021). In shortened layers, standard relative attention is used. For LSH attention, we set chunk length to 128 and use 2 hashes, which results in small memory complexity in our full-size layers. In this setup, we reach a score of **3.443** bpd with a $(3@1, 12@3, 3@1)$ architecture. All attention layers had $d_{model} = 768$, $d_{ff} = 3072$ and 8 heads. No dropout was used.

3.4.3 CIFAR-10

CIFAR-10 (Krizhevsky, 2009) is an image dataset consisting of 60000 images of size 32x32. We use this dataset primarily for our ablations (Section 4). Due to the relatively small number of examples compared to ImageNet, models reach convergence after 100k steps.

4 Ablations

In this section, we start by introducing a training technique called *shorten factor dropout* (Section 4.1), and then analyze Hourglass's components described above. We show that shortened layers behave similarly to full token-level layers in terms of scalability (Section 4.2). Then we study the effect of different distributions of $(pre, post)$ vanilla layers on Hourglass' accuracy (Section 4.3). We further analyze the performance of various upsampling and downsampling methods (Sections 4.4 and 4.5). Finally, we discuss different shorten factors and multi-stage shortening in Section 4.6.

ImageNet32	BPD
PixelCNN (van den Oord et al., 2016b)	3.83
Image Transformer (Parmar et al., 2018)	3.77
Axial Transformer (Ho et al., 2019)	3.76
Hourglass	3.74
VDM (Kingma et al., 2021)	3.72
DenseFlow (Grcić et al., 2021)	**3.63**
ImageNet64	BPD
Reformer (Kitaev et al., 2020)	3.65
Performer (Choromanski et al., 2021)	3.64
Hourglass	3.44
Sparse Transformer (Child et al., 2019)	3.44
Routing Transformer (Roy et al., 2020)	3.43
Combiner (Ren et al., 2021)	3.42
VDM (2021)	3.40
DenseFlow (2021)	**3.35**

Table 3: Bits per Dimension (BPD) on downsampled imagenet. Autoregressive models are separated by a horizontal line from non-autoregressive ones. On ImageNet32, our model yields new state-of-the-art for autoregressive models.

We conduct the ablations on both text and image generation to show applicability across different domains. We report bits per character (BPC) on the enwik8 validation (dev) set evaluated without context (sequence length 2048) and bits per dim (BPD) on the CIFAR-10 test set. For the exact hyperparameter setup refer to the Appendix.

4.1 Shorten factor dropout

Different shorten factors can be used for the same model when using parameterless pooling methods. We propose a training procedure where the shorten factor is randomly sampled with uniform distribution from a predefined set in each step. We observe that such a training regime improves validation loss compared to a baseline trained with a single, fixed shorten factor. For example, a model trained with shorten factor randomly sampled from $\{2, 3\}$ performs better when evaluated with any of these shorten factors, compared to models trained with a corresponding fixed shorten factor (Tab. 4).

We hypothesise that such a technique promotes a more uniform distribution of information over the sequence of tokens. It may be essential for fixed-size pooling techniques as they do not ac-

count for variable length constituents like words. By spreading information uniformly, we prevent a situation where we lose content by shortening three information-dense tokens or lose available capacity by merging three low information ones.

Shorten factor dropout is not limited to our architecture and can be applied to any model that utilizes shortening, particularly (Dai et al., 2020).

Hierarchy	Train k	Val $k = 2$	Val $k = 3$
2@1 8@k 2@1	$\{2, 3\}$	**1.104**	**1.116**
	2	1.116	
	3		1.124
4@1 12@k 4@1	$\{2, 3\}$	**1.086**	**1.094**
	2	1.098	
	3		1.101
5@1 10@k 5@1	$\{2, 3\}$	**1.082**	**1.087**
	2	1.096	
	3		1.095

Table 4: Comparison between models trained with shorten factor dropout (Train $k = \{2, 3\}$, Section 4.1) and fixed shorten factor baselines on enwik8.

4.2 Scaling shortened layers

In this study, we show that layers operating on the shortened sequence contribute significantly to Hourglass's accuracy. In Table 5 we measure the impact of scaling the depth of the shortened part of the model with a fixed number of vanilla layers.

We also check if scaling laws of Transformers, described in (Kaplan et al., 2020), hold by comparing a regression line fitted to various Hourglass configurations and one fitted to Transformer-XL baseline. We observe in Figure 1 that the slopes are very similar, which indicates that the laws hold.

Number of shortened layers	enwik8	CIFAR-10
Baseline ($n = 1$)	1.164	3.28
$n = 4$	1.134	3.16
$n = 8$	1.111	3.07
$n = 16$	**1.096**	**3.03**

Table 5: Impact of increasing the number of shortened layers on perplexity. Vanilla layers: $(1, 1)$ for CIFAR-10 and $(2, 2)$ for enwik8, shorten factor 3 used in both.

4.3 Impact of vanilla layers

We observe a significant contribution to Hourglass' performance with increasing the number of vanilla layers. One reason is that we perform more computations as in vanilla layers we process the sequence in token-level - no shortening is applied. We also see that the distribution of vanilla layers before shortening and after shortening does impact the training (see Tab. 6), and equal distribution leads to the best perplexity.

Vanilla layers	enwik8	CIFAR-10
(0, 0)	1.460	3.429
(0, 2)	1.176	3.108
(2, 0)	1.189	3.035
(1, 1)	**1.171**	**3.012**
(2, 2)	1.128	2.966

Table 6: Impact of the distribution of vanilla layers on enwik8 (BPC) and CIFAR-10 score (BPD). We see that equal distribution of layers before and after shortening leads to better results on both datasets.

4.4 Upsampling method

In Table 7 we investigate different possibilities of choosing the upsampling method. For attention-free methods, linear upsampling performs better on images, while repeat upsampling works well for text. Attention upsampling works well regardless of the function U and has the lowest perplexity.

Upsampling method	enwik8	CIFAR-10
Repeat	1.148	3.062
Linear	1.163	3.020
$U(x, x') = x$	1.145	**2.967**
$U(x, x') = x + Linear(x')$	**1.132**	3.012

Table 7: Upsampling method ablation - baseline configurations are $(2@1, 24@4, 2@1)$ and $(1@1, 8@3, 1@1)$ for enwik8 and CIFAR-10, respectively.

4.5 Pooling method

Table 8 presents impact of pooling method on both enwik8 (BPC) and CIFAR-10 (BPD). Attention pooling reaches the lowest perplexity for both datasets. Average pooling performs well on text among attention-free methods, while linear pooling works better for images. Both of these methods perform significantly worse for the other modality. Attention pooling demonstrates small differences with respect to chosen shortening function S (Section 2.1), still preserving the preference towards linear pooling on images and average pooling on text.

Pooling method	enwik8	CIFAR-10
AvgPool	1.129	3.116
Attention, $S = AvgPool$	**1.124**	3.012
Attention, $S = LinearPool$	1.142	**2.998**
LinearPool	1.159	**2.998**

Table 8: Ablation of pooling methods. Attention pooling achieves the best perplexity on both datasets.

4.6 Shortening strategies

While the analysis above gives a clear indication of what methods to choose for shortening and upsampling, we are still left with the question of which shorten factors to use and whether to do single-stage or multi-stage shortening.

Consistently, it is beneficial to do at least one shortening and by a factor of at least 3, while keeping 2-3 vanilla layers. Beyond that, a number of different configurations can yield similar results. In Table 1 we present the different hierarchical configurations that we tested on enwik8 and plotted in Figure 1. It can be seen that configurations with similar computation costs perform similarly. The sequence length used in these experiments is 2048 – we hypothesise that more hierarchy may be beneficial with even longer sequences.

5 Related Work

Shortening in Transformers Shortening in our work is inspired by Funnel-Transformer (Dai et al., 2020). The key difference is that they train an encoder model for text classification, where our work is entirely focused on language modeling, which provides additional challenges we had to solve regarding shortening in the autoregressive setup (Section 2.2). Another difference is that they use repeat upsampling method while we use attention. There are also a few works related to character-level modeling which use shortening, namely (Clark et al., 2021) and (Tay et al., 2021). However, the authors of these works focused mainly on shortening sequence in encoder part of the transformer, whereas we focused on applying shortening in decoder.

The idea of shortening is also discussed in (Subramanian et al., 2020). However, proposed architectures either focus on downsampling or upsampling, while Hourglass is a U-Net-like architecture and is symmetric in these terms. Their models use transformer layers on the finest scales when post-processing final representations. We do these also, in the beginning, to preprocess tokens on the finest scale, and we have found it essential to the score (Section 4.3). Our attention upsampling method is similar to their *aggregation layer* in the bottom-up model, however we use one upsampling for each scale change while they combine different scales to create one global upsampling.

Relative positional encoding Our work is primarily built on the backbone of Transformer-XL (Dai et al., 2019) - we use the same relative attention parametrization. Instead of the segment-level recurrence mechanism, we use shortening to make our model more efficient and feasible to train on longer sequences. Another relative attention parametrization is RoFormer (Su et al., 2021) where rotary positional embeddings are introduced. We find this work particularly relevant because the method is compatible with any attention type, including efficient attention, and can be combined with our model (Section 3.4.2).

Sparse Attention A well-known approach addressing the memory bottleneck is utilizing sparsity patterns in the attention matrix - Routing (Roy et al., 2020) and Sparse Transformer (Child et al., 2019) are examples of such methods. Our solution is different in the sense that it uses full attention - just with shortened sequence length. Combiner (Ren et al., 2021) makes a step further and provides full attention capabilities with similar computational complexity to Routing and Sparse transformers by leveraging structured factorization. This work, similarly to papers mentioned above on efficient transformers, concentrates on speeding up the attention component, while the most important feature of the Hourglass architecture is that it can use any attention module as a drop-in.

Image generation on downsampled ImageNet VDM (Kingma et al., 2021) and DenseFlow (Grcić et al., 2021) are recently proposed state-of-the-art methods for density estimation on this dataset. The difference between these methods and Transformer-based methods (Parmar et al., 2018; Ho et al., 2019) including this work is that the former, unlike Transformers, are non-autoregressive.

6 Conclusion

In this paper, we show how hierarchy can improve the efficiency of Transformers in a language modeling setup. Our proposed architecture, Hourglass, significantly outperforms the baseline both in terms of perplexity reached at a given computation cost (Figure 1) and empirical metrics like running memory (Figure 6). Hourglass achieves state-of-the-art results among autoregressive models on the ImageNet32 generation task and competitive results on other image generation and language modeling tasks.

Hourglass can be used with any attention type, which opens many directions for future research related to Transformers capable of processing longer sequences. Another line of future work might be related to advances in the shortening mechanism itself, for example, involving a dynamic pooling operation that could explicitly handle the problem of fixed-size groups in multi-stage shortening. We also leave open the problem of choosing the best hi-

erarchy for a task. We conjecture that experiments with much longer contexts will provide better guidance for this choice and will benefit even more from the Hourglass architecture.

7 Acknowledgments

Some experiments were performed using the Entropy cluster funded by NVIDIA, Intel, the Polish National Science Center grant UMO-2017/26/E/ST6/00622, and ERC Starting Grant TOTAL. The work of Henryk Michalewski was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959. The authors would like to thank Marek Cygan and Kamil Wilczek for their help with cluster setup, and Grzegorz Grudziński, Dawid Jamka and Sebastian Jaszczur for helpful discussions. This article describes a Team Programming Project completed at the University of Warsaw in the academic year 20/21. We are grateful to Janusz Jabłonowski, the head of Team Programming Projects, for his support and open-mindedness.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020. Generative pretraining from pixels. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1691–1703. PMLR.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2021. Rethinking attention with performers.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2021. Canine: Pre-training an efficient tokenization-free encoder for language representation.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. 2020. Jukebox: A generative model for music.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. 2021. Addressing some limitations of transformers with feedback memory.

Matej Grcić, Ivan Grubišić, and Siniša Šegvić. 2021. Densely connected normalizing flows.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. 2019. Axial attention in multidimensional transformers.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2018. Music transformer.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention.

Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. 2021. Variational diffusion models.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer.

Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images.

Matt Mahoney. 2011. Large text compression benchmark.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation.

Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang, Jure Leskovec, Dale Schuurmans, and Bo Dai. 2021. Combiner: Full attention transformer with sparse computation cost.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient content-based sparse attention with routing transformers.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding.

Sandeep Subramanian, Ronan Collobert, Marc'Aurelio Ranzato, and Y-Lan Boureau. 2020. Multi-scale transformer language models.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. 2021. Not all memories are created equal: Learning to forget by expiring.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast character transformers via gradient-based subword tokenization.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016a. Pixel recurrent neural networks. *CoRR*, abs/1601.06759.

Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. 2016b. Conditional image generation with pixelcnn decoders.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization.

Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. 2021. Long-short transformer: Efficient transformers for language and vision.

A Autoregressive shortening

In Section 2.2 we address two problems of shortening in an autoregressive setup: information leaks and reduced expressivity. Here we study these issues in more detail.

A.1 Motivation behind using vanilla layers

At first sight, it may be tempting to create hierarchical models that directly shorten the input to maximize the efficiency gains. In this section, we explain why vanilla layers are crucial for modeling at least some sequences, especially due to autoregressivity.

Consider a sequence modeling task where the input is a random sequence with repeats, such as `A#AC#CD#DB#B`. The sequence consists of chunks `L#L` where `L` is a random uniform letter and `#` is a special symbol. A vanilla Transformer language model can achieve 66% sequence accuracy on this task – it cannot predict the token at the beginning of the chunk, but it can predict the last token of the chunk by simply copying the token at 2 positions earlier, which is possible using a vanilla self-attention layer.

It is however not easy to learn this task in a shortening setup when there are no vanilla layers operating on the finest scale – this is the situation defined in *Reduced expressivity* subsection of Section 2.2. Assume shorten factor is $k = 3$ and the input is `A#AB#BC#C`. To avoid information leak, we shift the input sequence right by 1, and then by $k-1 = 2$ directly before shortening. Then the sequence is `000A#AB#B`. Our shortened embeddings are as follows: $e_0 = S(emb_0, emb_0, emb_0), e_1 = S(emb_A, emb_\#, emb_A)$ where emb is input embedding matrix and S is a shortening function.

Shortened embeddings	`[000]`	`[A#A]`	`[B#B]`
Shifted input embeddings	`0A#`	`AB#`	`BC#`
Target sequence	`A#A`	`B#B`	`C#C`
Positions	`123`	`456`	`789`

Table 9: Example input sequence which is difficult to model without vanilla layers. The model can use only input embeddings shifted by one from the residual and shortened embeddings (shorten factor is 3) to predict the target sequence. Note that it is impossible to predict tokens at positions divisible by 3 using only that information.

Because no vanilla layers are used, for prediction we can use only shortened embeddings and input token embeddings shifted right by 1 from the residual connection. Note that to predict the `A` token at position 3 we can use only embedding of $emb_\#$ and e_0 - both of these contain no information so we are unable to predict this token better than randomly (see Table 9). An analogous situation occurs for prediction of any tokens at positions divisible by 3, which makes the model unable to achieve more than 50% accuracy when the task has vocabulary size of at least 2.

This issue can be solved by adding at least one vanilla layer to the model, so that it can attend within the neighborhood of k previous tokens. For this particular problem, it is sufficient to use local attention with context size k in vanilla layers which is significantly more efficient than full attention.

A.2 Information leaks – analysis

A.2.1 Definition of autoregressive model

Formally, given a target sequence, $x = x_1, ..., x_n$, an autoregressive model (e.g. transformer decoder) models the sequence as $P(x) = \prod_{i=1}^{n} P(x_i|x_1, ..., x_{i-1})$ and

$$\forall_i P(x_i|x_1, ..., x_n) = P(x_i|x_1, ..., x_{i-1})$$

namely x_i token depends only on previous tokens, never on itself nor next ones.

A.2.2 Definition of information leak

We say that a leak was caused by function $F_n \colon A^n \longrightarrow A^n$ transforming sequence of input tokens $(x_1, x_2, ..., x_n)$ into another sequence $(u_1, ..., u_n) = F((x_1, ..., x_n))$ when $\exists_{i<j<n} P(x_i|x_1, ..., x_{i-1}, x_j) \neq P(x_i|x_1, ..., x_{i-1})$, namely there exists $j \geq i$ that token x_i depends on x_j which violates the autoregressive property.

A.2.3 Model representation

Let $R_k \colon A^n \longrightarrow A^n$ be a shift right function which reindexes tokens by shifting each of them right by k positions:

$$R_k((x_1, x_2, ...x_n)) = (\underbrace{0, ..., 0}_{k}, x_1, ..., x_{n-k})$$

$S_k : A^* \longrightarrow A^*$ shortening function with factor k which takes on input $x_1, ..., x_n$ sequence and returns $s_1, ..., s_m$ where $m = \frac{n}{k}$, U_k upsampling function which works in similar way but upsamples $U_k((u_1, ..., u_m)) = u_1, ..., u_n$.

Between them there is also applied D decoder function, $D = D_1 \circ \cdots \circ D_l$, where each D_i is a

function representing decoder block. Due to causal attention masking in the decoder block, there is no risk of information leak caused by function D.

A.2.4 Leak description

Because of mentioned attention mask, we will omit the flow of information between tokens caused by the influence of attention mechanism because this mask keeps the autoregressive property.

Now, let $(x_1, ..., x_n)$ be an input sequence and $(u_1, ..., u_n) = U(D(S_k(T_s((x_1, ..., x_n))))) = F$. In order to preserve autoregressive property, it is obligatory that no leak occurs.

We will show that shift by any value $0 < s < k - 1$ where k is the shorten factor will cause a leak.

To start with, consider input sequence $(x_1, ..., x_n)$ and perform operation F. $R_s((x_1, ..., x_n)) = (\underbrace{0, ..., 0}_{s}, x_1, ..., x_{n-s}) = r$. Assuming that n is divisible by s, we have $S_k(r) = (v_1, ..., v_{\frac{n}{k}}) = v$ where each v_i consists of information obtained in $(r_{(i-1)\cdot k+1}, ..., r_{ik})$. Now let see that operation D preserves autoregressive property, let $d = D(t)$. Now, $U(d) = (u_1, ..., u_n)$ and each u_i depends on $d_{\lfloor \frac{i-1}{k} \rfloor + 1}$.

Now consider $s \leq k - 2$ and let $(u_1, ..., u_n) = F((x_1, ..., x_n))$ will be a result of our Transformer part. Let take u_1 which depends on d_1 and d_1 depends on $(r_1, ..., r_k) = (0, ..., 0, x_1, ..., x_{k-s})$. For that reason d_1 depends on $x_1, x_2, ..., x_{k-s}$, so we have

$$P(x_1|x_{k-s}) \neq P(x_1)$$

which violates the autoregressive property.

B Experimental setup

B.1 Common parameters

Here we list common hyperparameters used for all experiments mentioned in the paper. We use Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 1\text{e}{-9}$. Weight decay and gradient clipping is not used.

In terms of model details, we decided to use a Pre-Norm architecture and FastGelu activation in feed-forward layers.

B.2 Enwik8

We use $d_{model} = 512$, $d_{ff} = 2048$ and 8 attention heads. Models in ablation study are trained for 200k steps with cosine learning rate schedule, setting cycle length for 200k steps and linear warmup of 4000 steps.

For the main result achieving **0.98** bpc with $4@1, 8@3, 4@1$ hierarchy, we set $d_{model} = 768$, $d_{ff} = 3072$ and $n_{heads} = 8$ which results in 146M parameters. It is trained for a total number of 350k steps with one cycle of cosine schedule. Linear warmup of 20k steps is used.

At the beginning of our work on this paper, we have performed grid search over following hyperparameters for enwik8:

- batch size: $\{8, 16, 32\}$, finally chosen 8
- dropout: $\{0.05, 0.1, 0.15, 0.20\}$, finally chosen 0.15
- learning rate: $\{1\text{e}{-4}, 2\text{e}{-4}, 3\text{e}{-4}, 4\text{e}{-4}, 5\text{e}{-4}\}$, finally chosen $4\text{e}{-4}$

All next experiments were conducted using these parameters without additional searching.

B.3 Downsampled ImageNet - common parameters

For ImageNet32 and ImageNet64 experiments we use inverse square root learning rate decay from (Vaswani et al., 2017), setting warmup steps to 8000 in both experiments. Total batch size is 64.

B.4 ImageNet32

In this dataset, we operate on input sequence length of 3072. We use $d_{\text{model}} = 512$, $d_{\text{ff}} = 2048$, 8 attention heads and 0.01 dropout rate. We perform 400k training steps with linear warmup and inverse square root decay and then we train for additional 70k steps with cosine learning rate decay, starting from the learning rate from the previous schedule at 400k and decreasing it to 0 at 470k steps.

B.5 ImageNet64

As an input we receive a sequence of 12288 tokens representing $64 \times 64 \times 3$ images. We set $d_{\text{model}} = 768$, $d_{\text{ff}} = 3072$, 8 attention heads and dropout equal to 0. We perform 300k steps with linear warmup and inverse square root decay.

B.6 CIFAR-10

All the ablation studies are run for 100k training steps, unless otherwise specified. Input sequence has length 3072 and model parameters are as follows: $d_{\text{model}} = 512$, $d_{\text{ff}} = 2048$, 8 attention heads and dropout equal to 0. Total batch size is 8. Cosine learning rate decay with linear warmup of 5000 steps and 100k cycle length is used.

C Environment setup

C.1 Hardware

Experiments are conducted on several setups.

- Ablation Study and short training sessions were computed on nodes consisting of 4x *Titan V* with 12GB memory each, 64GB RAM, *Intel Xeon E5-2660 v4* CPU
- longer trainings were completed on 8x *RTX 2080 Ti* with 11GB memory each, 128GB RAM and *Intel Xeon E5-2660 v4* CPU.
- Few longest trainings were conducted on 8×8 TPUv3 units, each with 16GB memory.

C.2 Software

All experiments were performed on Linux operating system using Trax library version 1.3.9 along with all its dependencies from this particular release date. Additionally, to run shorten factor dropout experiments we modified the Transformer-XL codebase in PyTorch.

D Reproducibility

To ensure the reproducibility of this work and to support open science principles, we made our code publicly available at `github.com/google/trax`. In this repository, we also provide Google Colab notebooks where the evaluation of our main Enwik8 and ImageNet32/64 results can be reproduced.[2][3]

D.1 Randomness

Seeds in all experiments were chosen randomly, however each experiment contains history which allows retrieving all randomly set parameters for reproductions.

For each ablation described in the ablation study section, we rerun the baseline 3 times to calculate standard deviation. All other experiments are run only once due to costs and since the variance we noticed was minimal.

D.2 Experiment representation

Each experiment is represented by a configuration file that unambiguously determines the whole setup – all hyperparameters and training details like specific optimizers, data preprocessing functions, or batch size per device.

[2] https://github.com/google/trax/blob/master/trax/models/research/examples/hourglass_enwik8.ipynb

[3] https://github.com/google/trax/blob/master/trax/models/research/examples/hourglass_downsampled_imagenet.ipynb