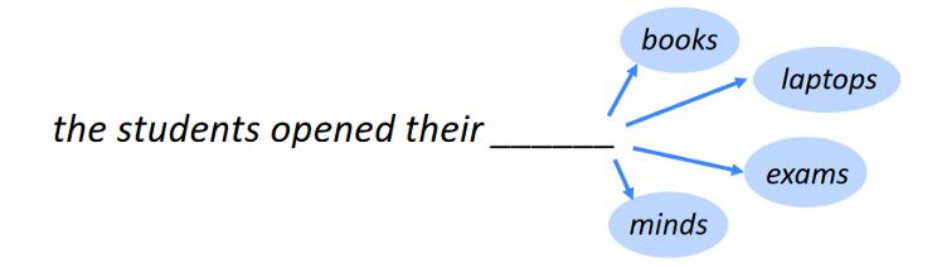
Hierarchical Transformers Are More Efficient Language Models

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, Henryk Michalewski

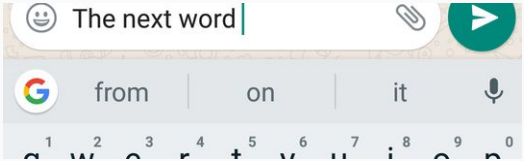
Language Modeling with Transformers - introduction

Language Modeling task

- Predict what word comes next, given some prefix of words



Everyone uses language models!



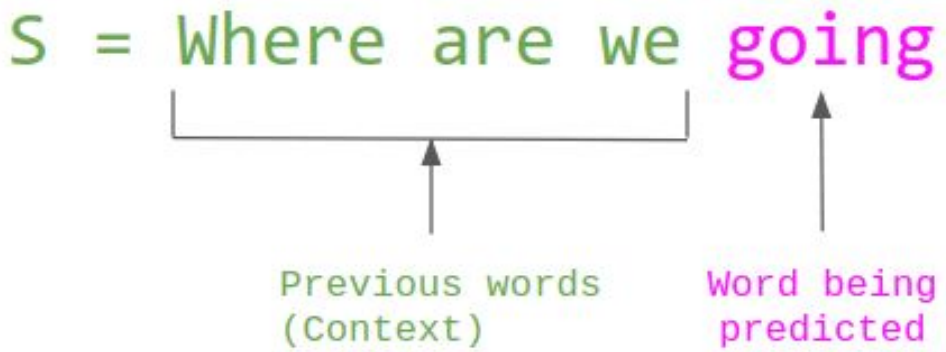
Language Modeling task

- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

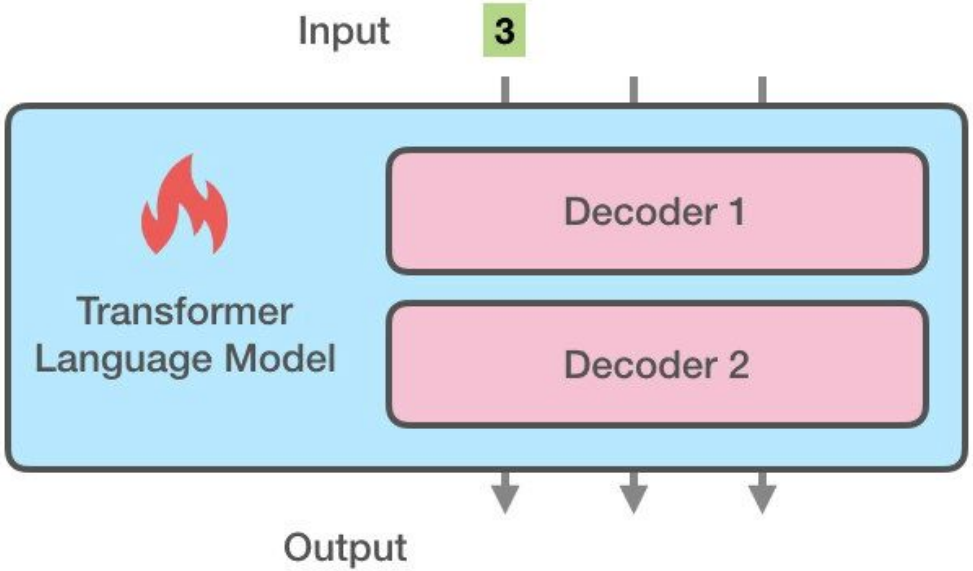
where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

Language Modeling = Text Generation



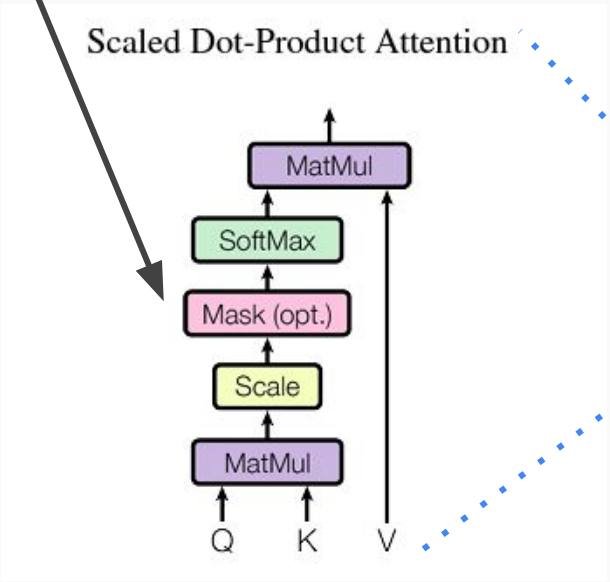
$$P(S) = P(\text{Where}) \times P(\text{are} \mid \text{Where}) \times P(\text{we} \mid \text{Where are}) \times P(\text{going} \mid \text{Where are we})$$

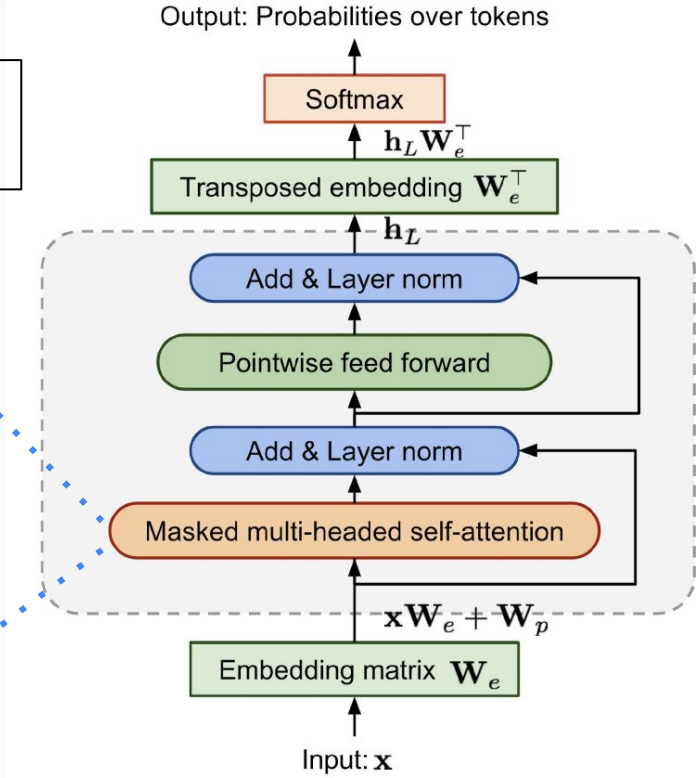
Language Modeling using Transformers - high-level overview



Transformer Decoder Architecture

Masking is required for autoregressive transformers (language models)





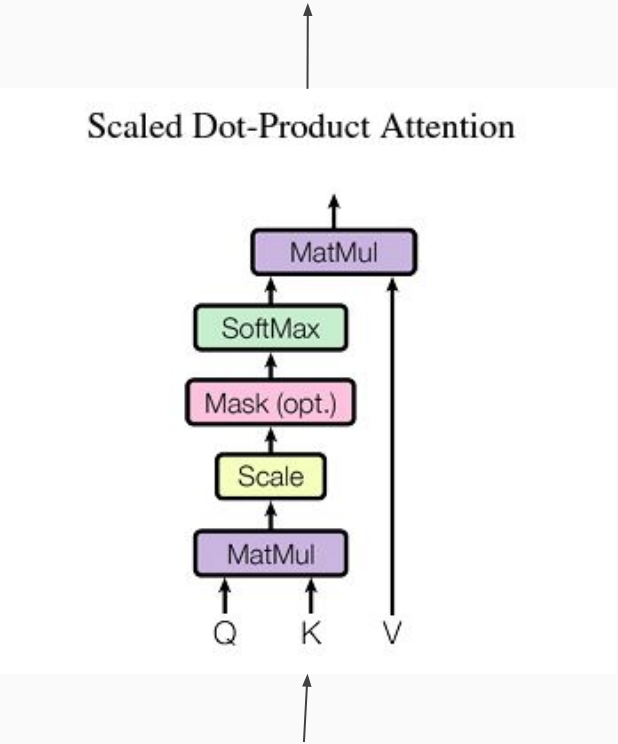
Transformer Block
Repeat x L=12

$$\mathbf{h}_\ell = \text{transformer\_block}(\mathbf{h}_{\ell-1})$$
$$\ell = 1, \ldots, L$$

Input: sequence of tokens

Attention layer in transformers - high-level overview

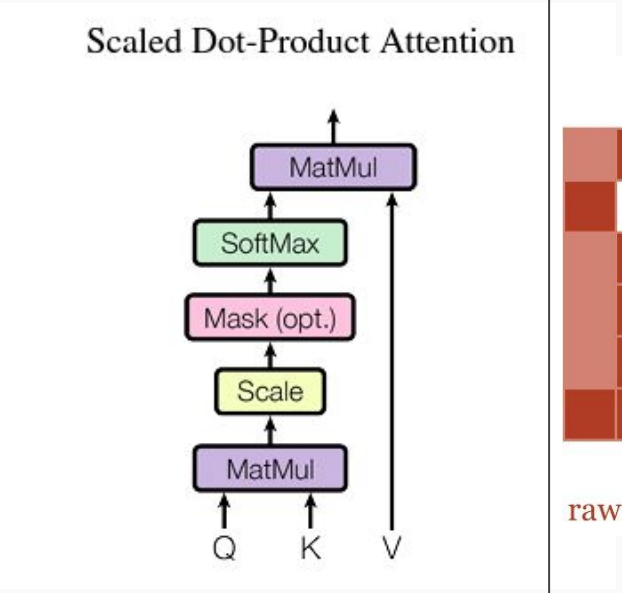
Output embeddings

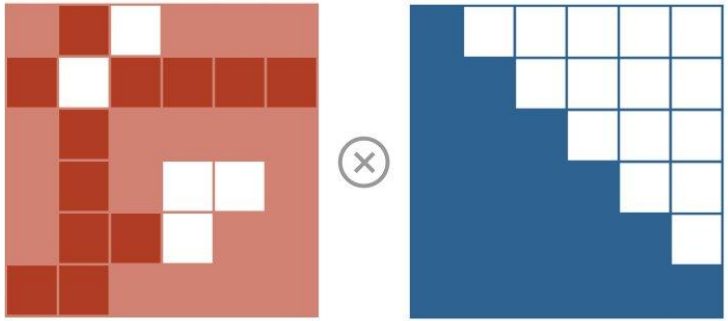


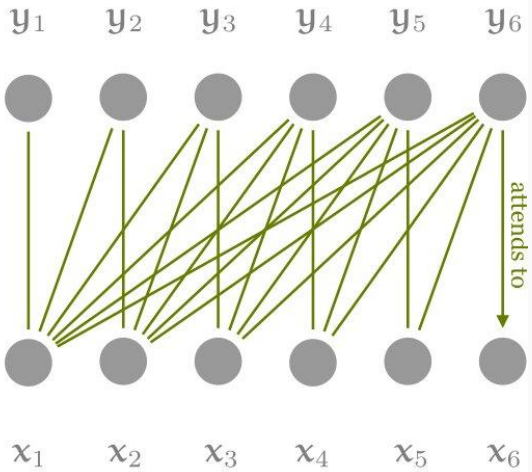
Input embeddings

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

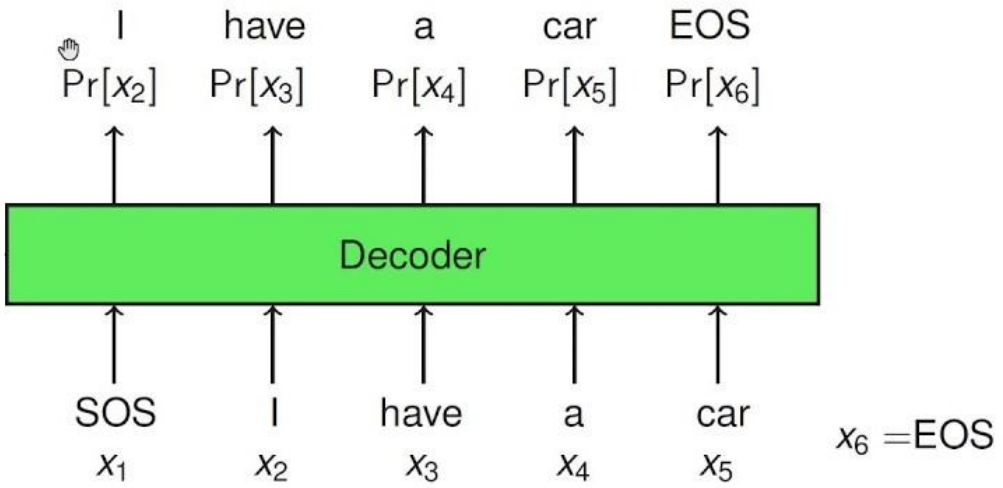
Attention masking - autoregressive transformers







Why Transformers are better than RNNs? Training parallelization



- Compute all next word probabilities in parallel?
  - Feed entire target sequence into decoder.
  - Only possible during training.
  - Predictions may not use future input.

Definition of the Autoregressive Property (a.k.a. “no feedback” property??)

- Formally, given a target sequence, x = x_1, ..., x_n, an autoregressive model (e.g. transformer decoder) models the sequence as

$$P(x) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1})$$

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

Hierarchical Transformers

Overview

- Hourglass is a Transformer-based language model

Overview

- Hourglass is a Transformer-based language model
- Builds on Funnel Transformer (Dai et al. 2020) and U-Nets (Ronneberger, Fischer, and Brox 2015).
  - *In contrast to these architectures, the model we present is autoregressive*

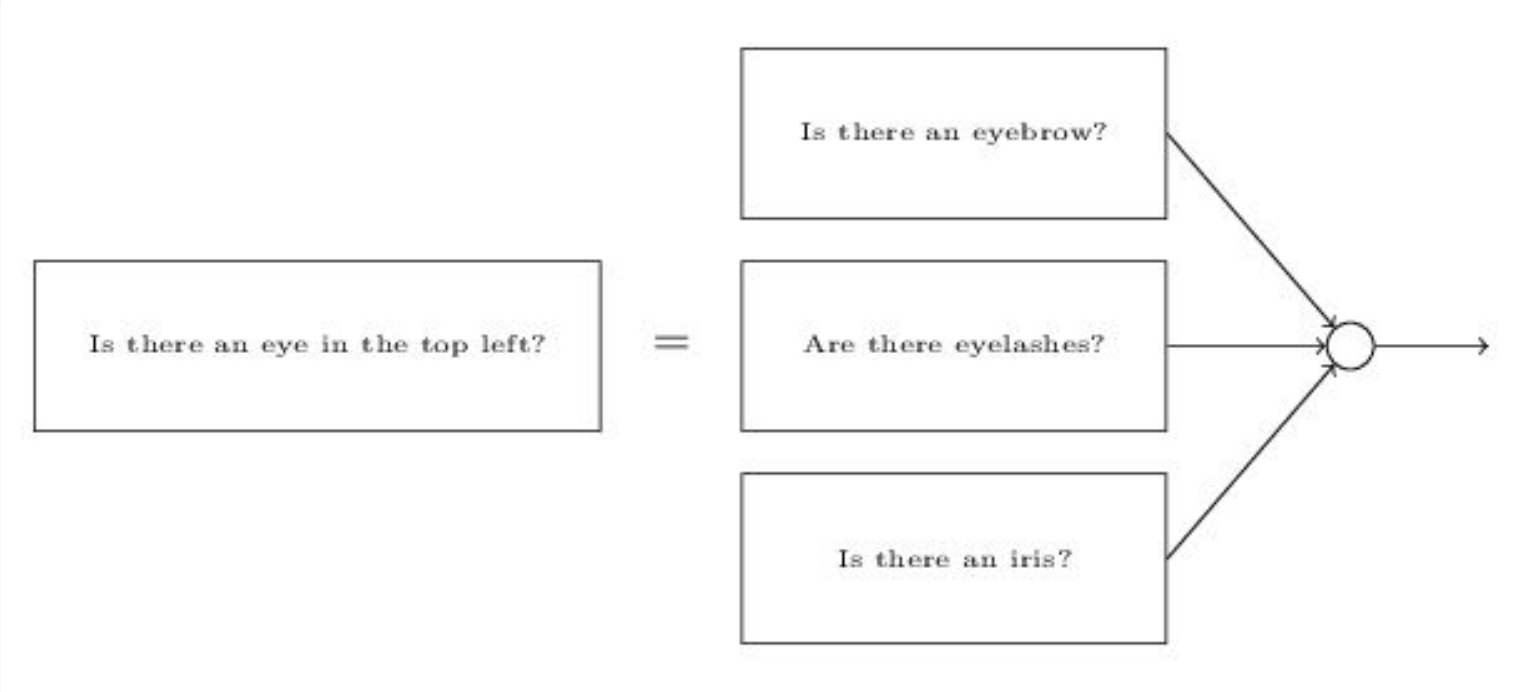
Overview

- Hourglass is a Transformer-based language model
- Builds on Funnel Transformer (Dai et al. 2020) and U-Nets (Ronneberger, Fischer, and Brox 2015).
  - *In contrast to these architectures, the model we present is autoregressive*
- Main point:

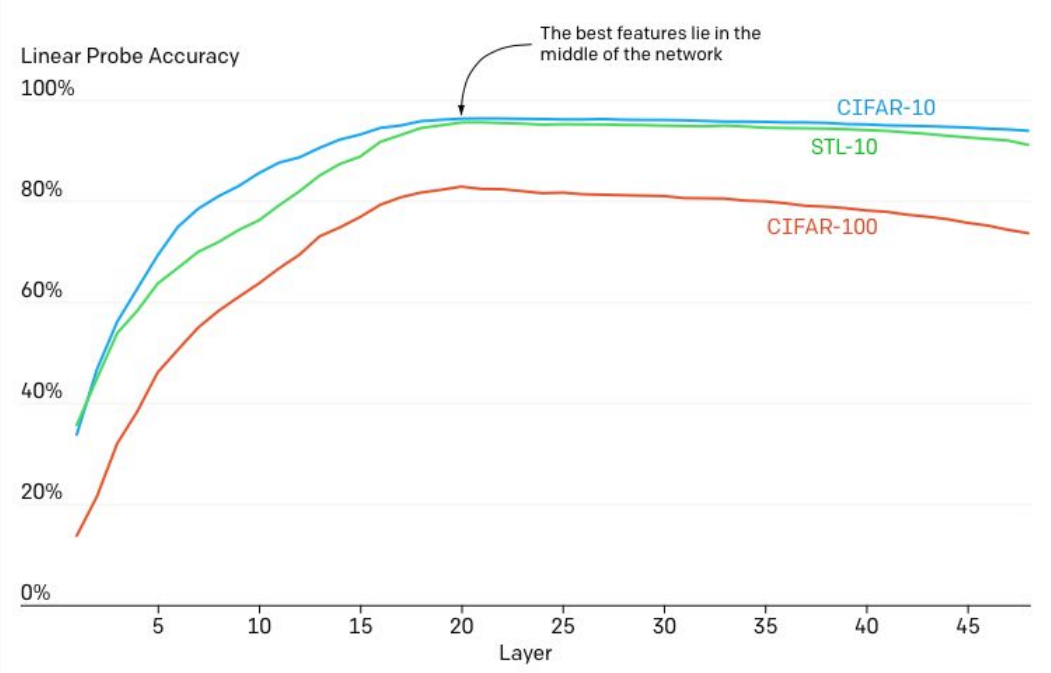
Overview

- Hourglass is a Transformer-based language model
- Builds on Funnel Transformer (Dai et al. 2020) and U-Nets (Ronneberger, Fischer, and Brox 2015).
  - *In contrast to these architectures, the model we present is autoregressive*
- Main point: Hierarchical transformers are more efficient language models

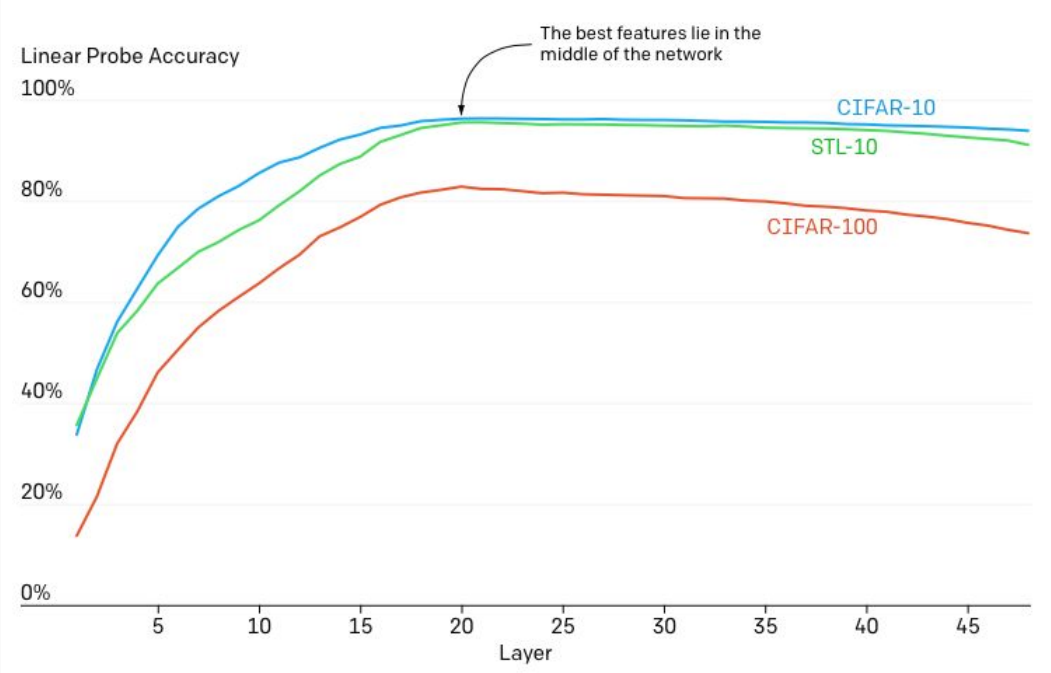
Models learn hierarchy?



Models use hierarchy! At least Transformers.

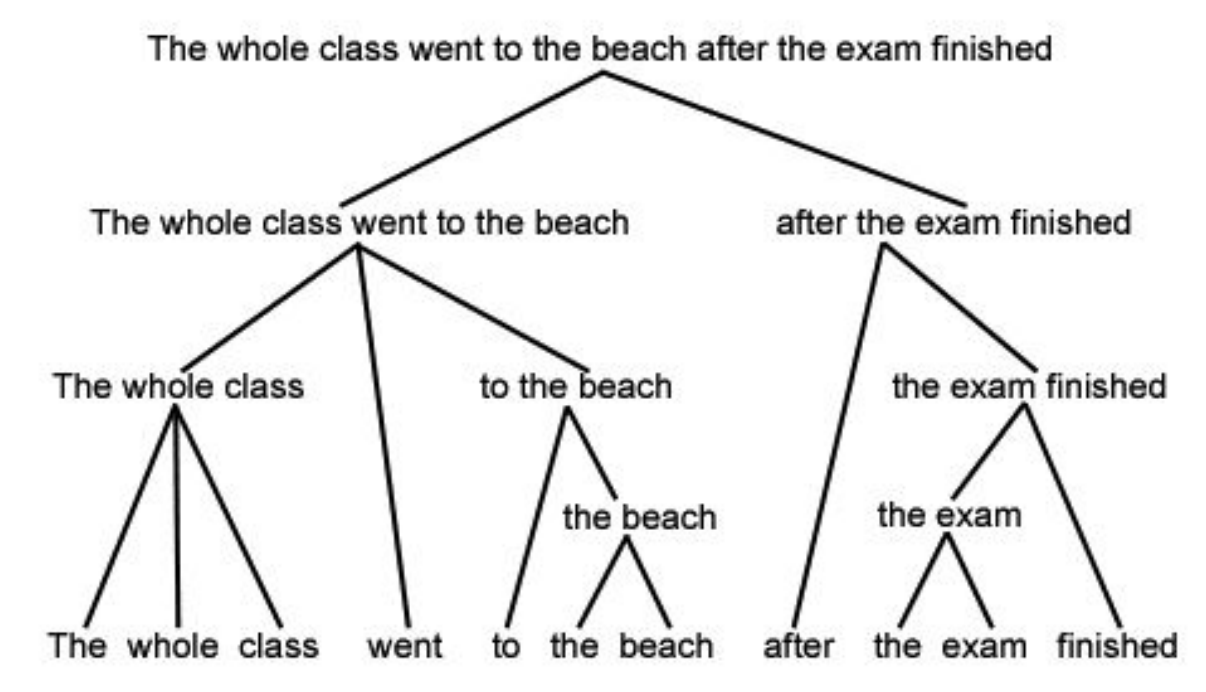


Models use hierarchy! At least Transformers.



Quote from Image GPT (Mark Chen et al.): “This behavior suggests that a transformer generative model operates in two phases: in the first phase, each position gathers information from its surrounding context in order to build a contextualized image feature. In the second phase, this contextualized feature is used to solve the conditional next pixel prediction task”

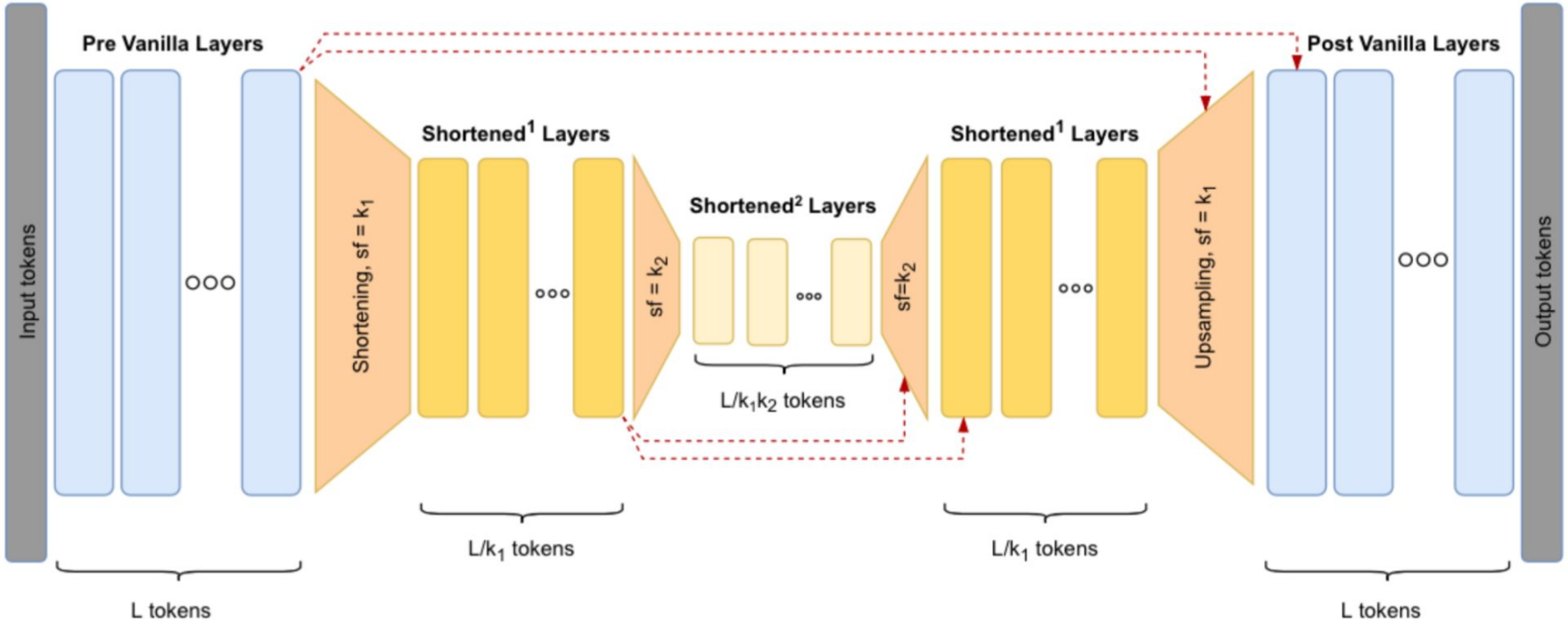
Humans use hierarchy!



Hourglass - let’s impose hierarchy

Contracting path to capture context

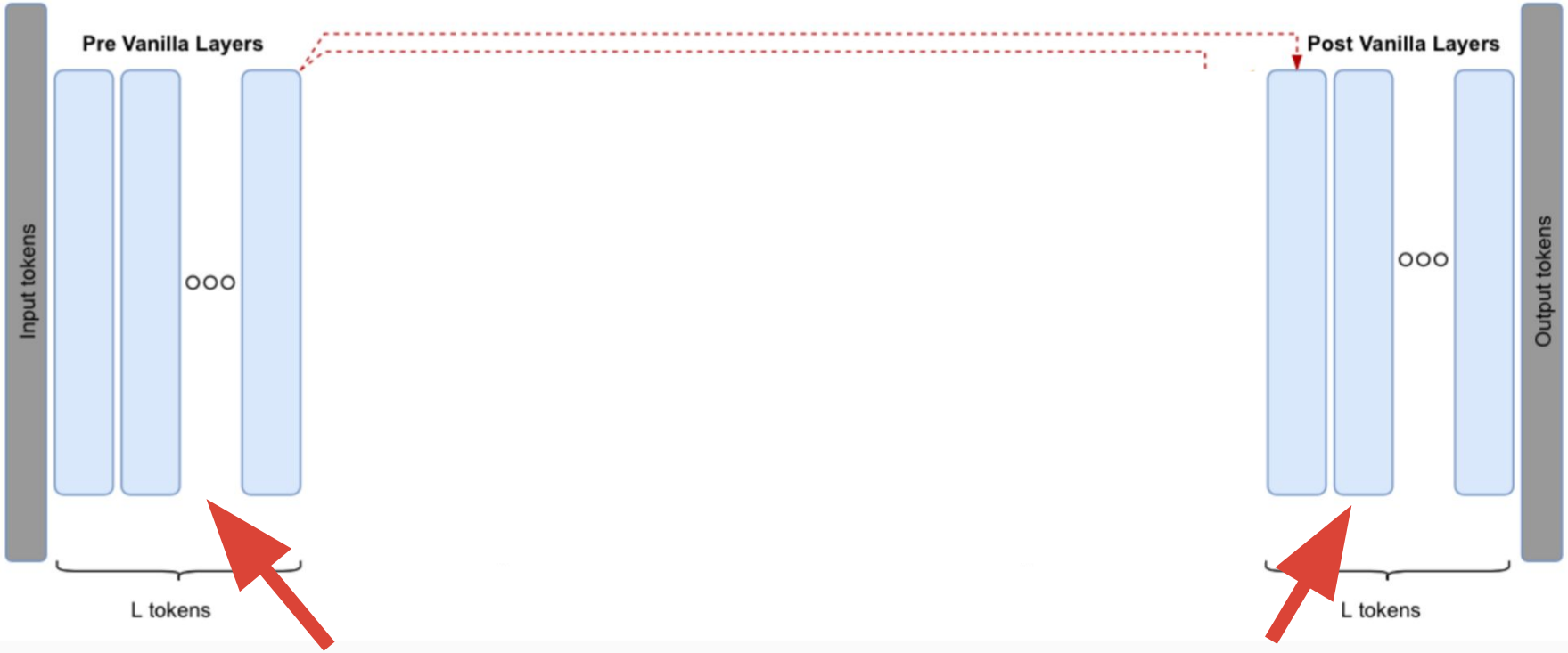
Expanding path for precise generation



Hourglass - let’s impose hierarchy

Contracting path to capture context

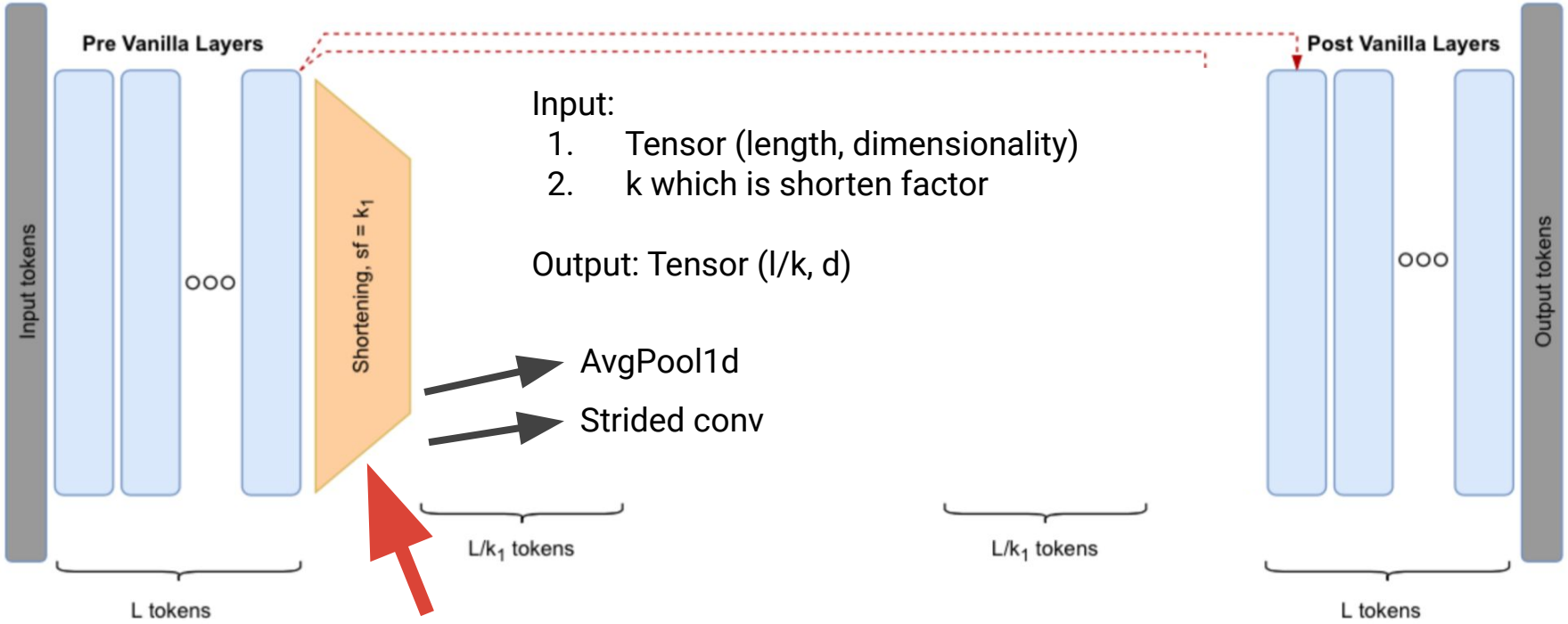
Expanding path for precise generation



Hourglass - let’s impose hierarchy

Contracting path to capture context

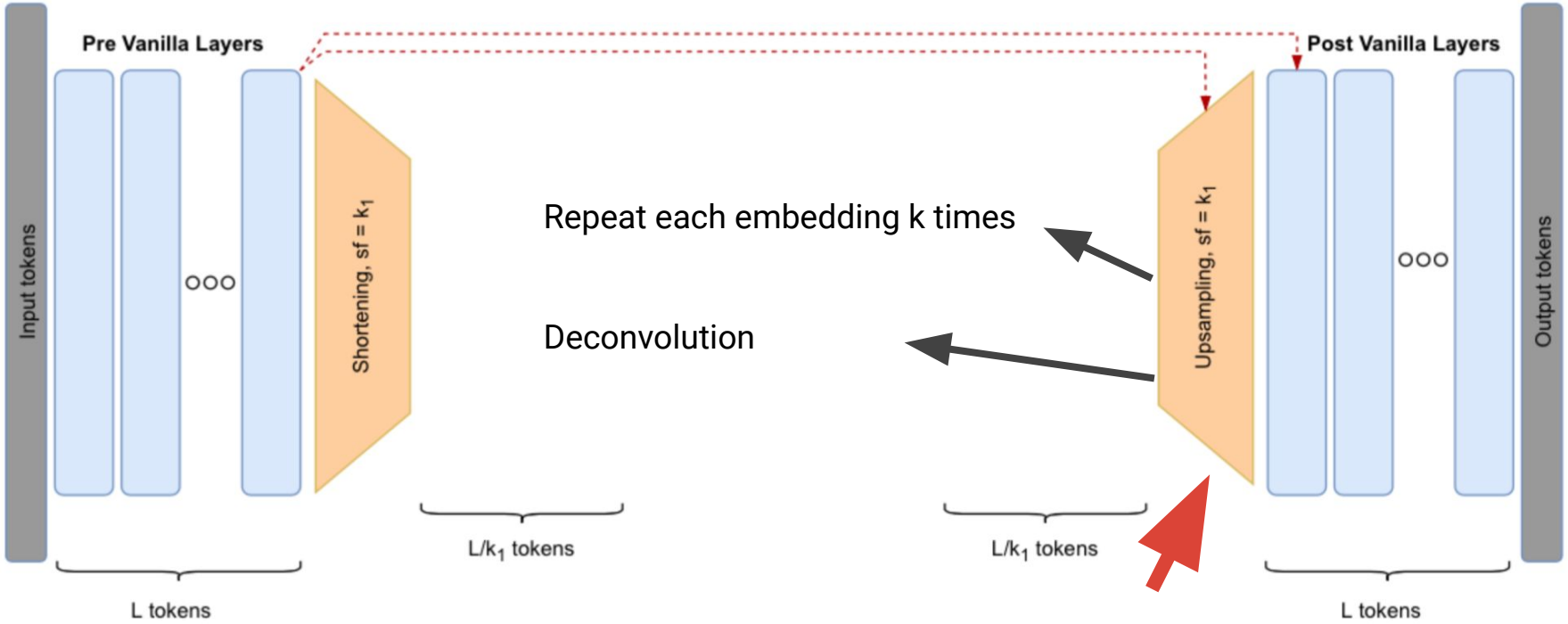
Expanding path for precise generation



Hourglass - let’s impose hierarchy

Contracting path to capture context

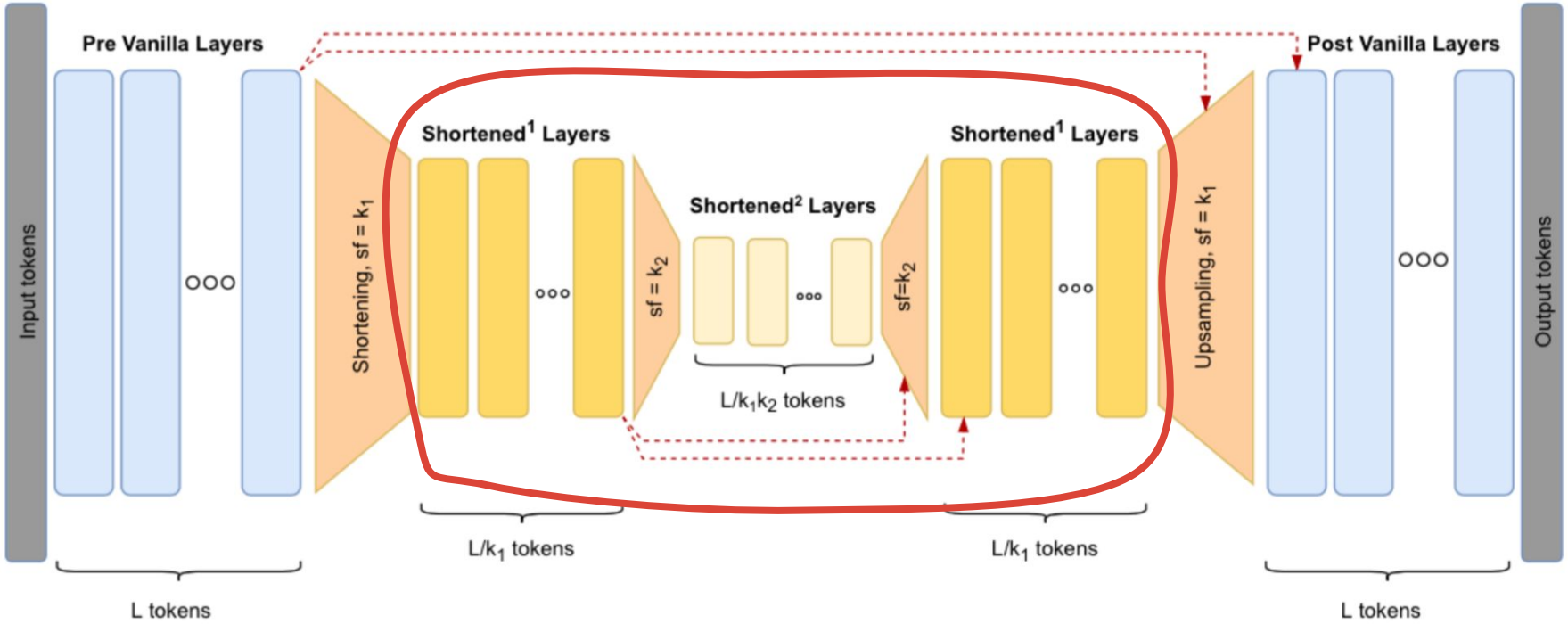
Expanding path for precise generation



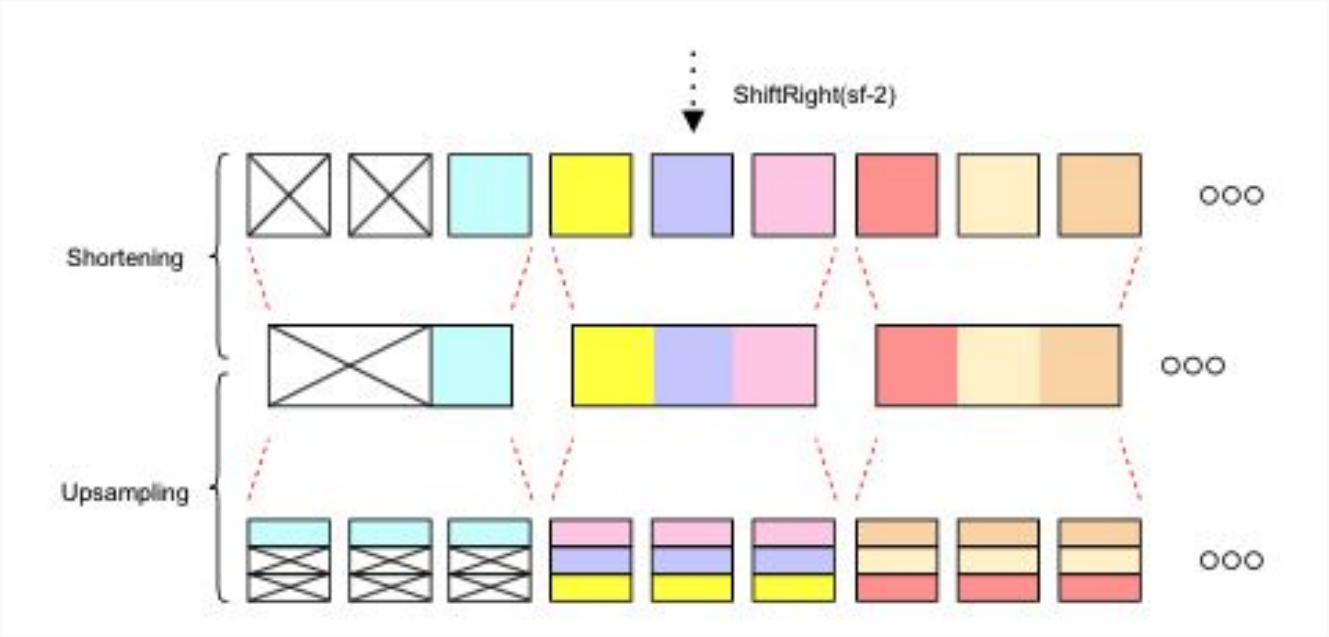
Hourglass - let's impose hierarchy

Contracting path to capture context

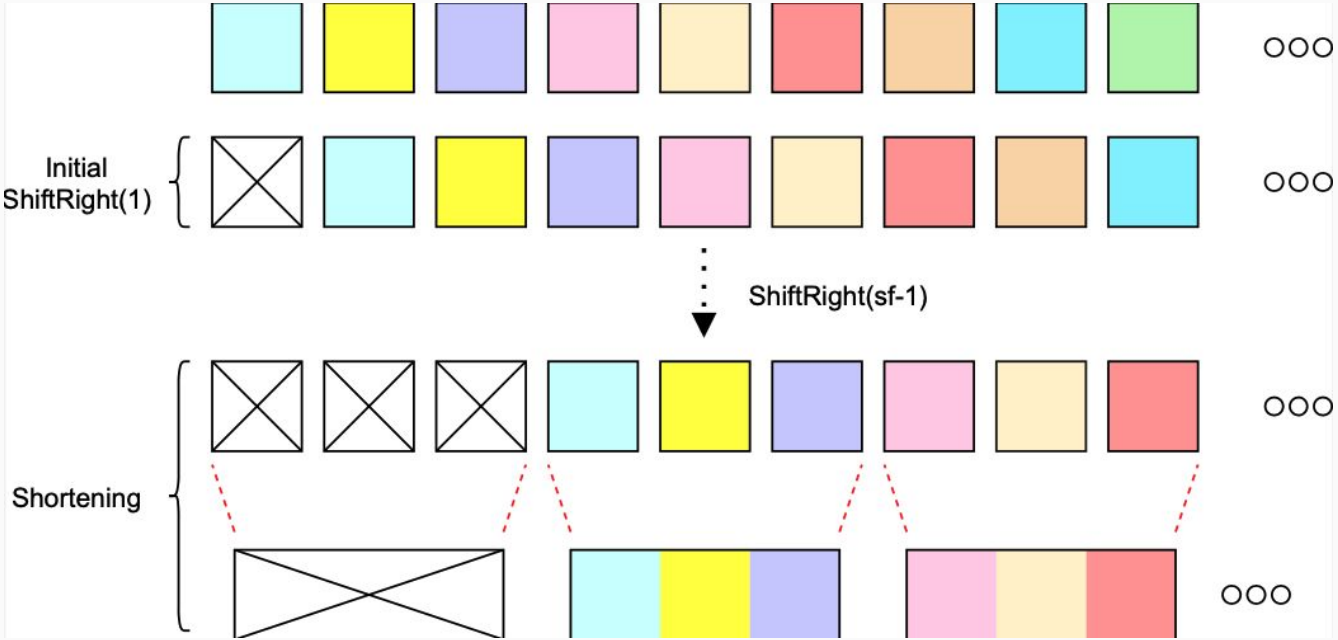
Expanding path for precise generation



Be careful with shortening/upsampling



Be careful with shortening/upsampling



Hourglass - recap with code

The idea of hourglass is easy to code

The Transformer blocks and the chosen self-attention stays the same.

What changes is that blocks operate on shortened representations.

Algorithm 1: HourglassLM

procedure HOURGLASS($x, [k, ...shorten\_factors]$)
  $x \leftarrow PreVanillaLayers(x)$
  $x' \leftarrow Shortening(ShiftRight(x, k-1), k)$
  if EMPTY($shorten\_factors$) **then**
    $x' \leftarrow ShortenedLayers(x')$
  else
    $x' \leftarrow$ HOURGLASS($x', shorten\_factors$)
  end if
  $x \leftarrow x + Upsampling(x, x', k)$
  $x \leftarrow PostVanillaLayers(x)$

Complexity

Model Architecture	Complexity per Layer
Recurrent	$O(n)$
Transformer, (Vaswani et al., 2017)	$O(n^2)$
Sparse Tansformer, (Child et al., 2019)	$O(n\sqrt{n})$
Reformer, (Kitaev et al., 2020)	$O(n \log(n))$
Performer	$O(n)$

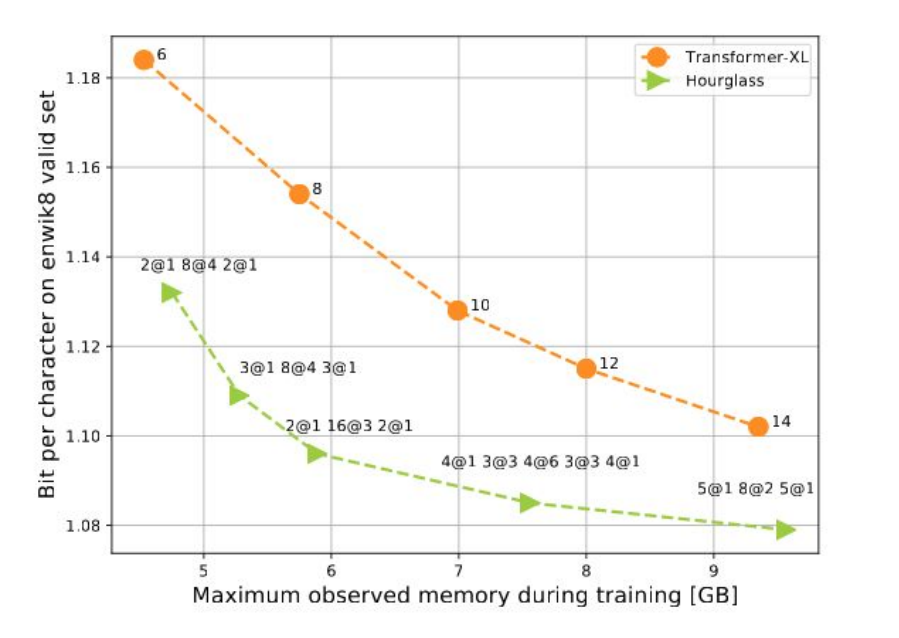
Hourglass

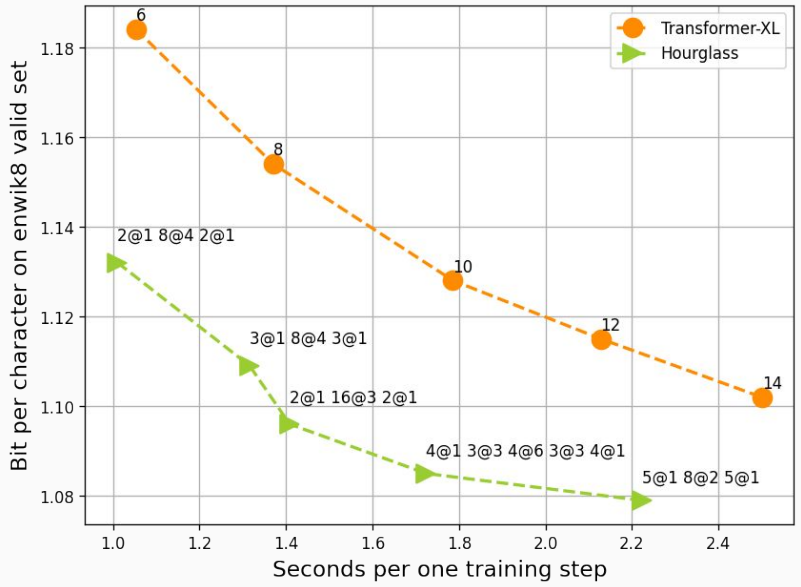
→ n^2 / k^2

→ n / k

n - sequence length
k - accumulated shorten factor

Results on generating enwiki8



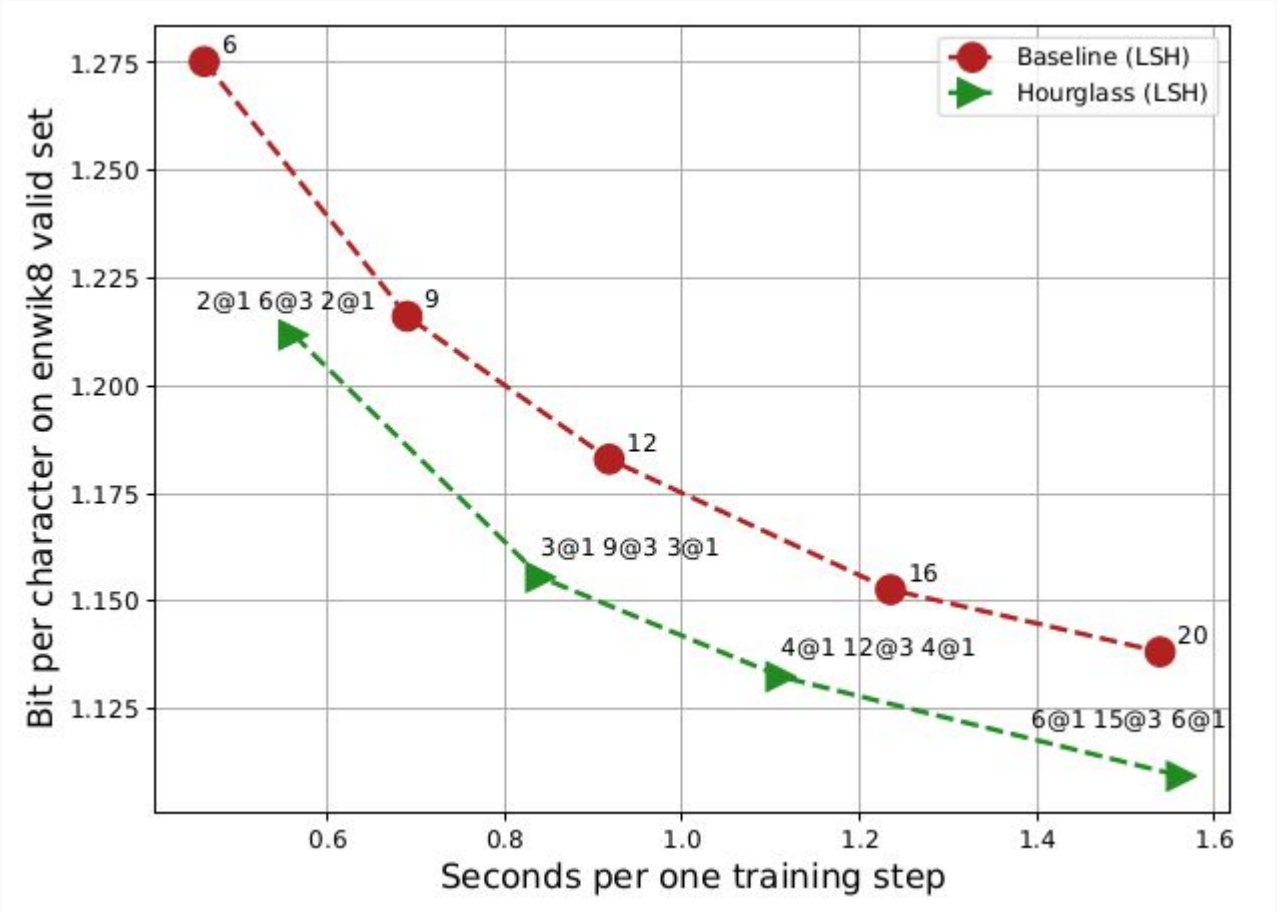


Results on generating imagenet

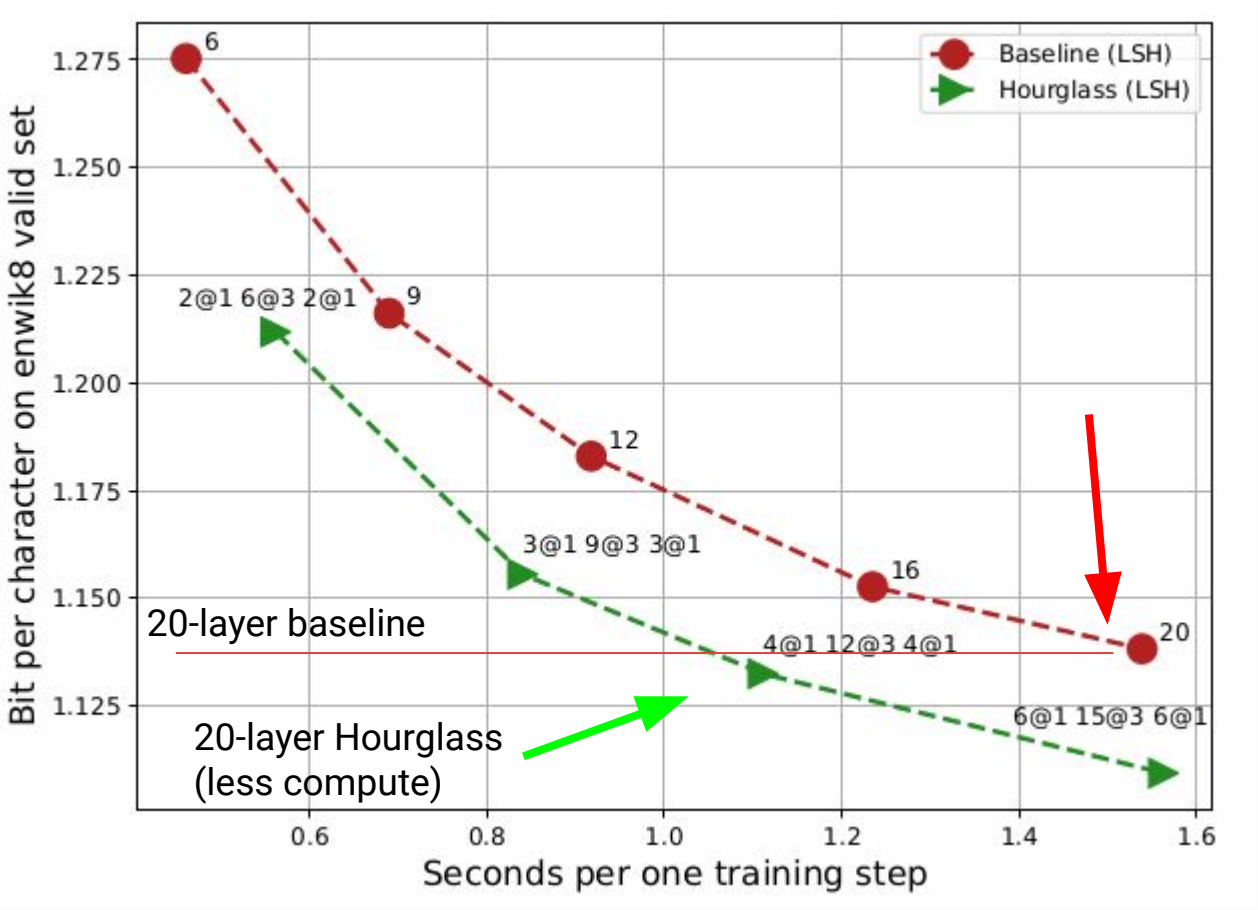
Figure 7: Examples of our model completions, where bottom half of each image was generated by our model, prompted by the upper half.

ImageNet32	Cost	BPD
PixelCNN (van den Oord et al. 2016)	-	3.83
Image Transformer (Parmar et al. 2018)	12	3.77
Axial Transformer (Ho et al. 2019)	24	3.76
Hourglass	16	3.74
VDM (Kingma et al. 2021)	-	3.72
ImageNet64	Cost	BPD
Performer (Choromanski et al. 2021)	12	3.64
Combiner-Mixture (Ren et al. 2021)	12	3.50
Hourglass	10	3.44
Sparse Transformer (Child et al. 2019)	48	3.44
Routing Transformer (Roy et al. 2020)	24	3.43
Combiner (Ren et al. 2021)	30	3.42
VDM (Kingma et al. 2021)	-	3.40

Hourglass works with different attention types



Comparison with fixed number of parameters - is the hierarchical model better?



Shorten factor = more layers!

Model Architecture	Complexity per Layer
Recurrent	$O(n)$
Transformer, (Vaswani et al., 2017)	$O(n^2)$

Hourglass

n^2 / k^2

2 vanilla layers = 2 * n^2

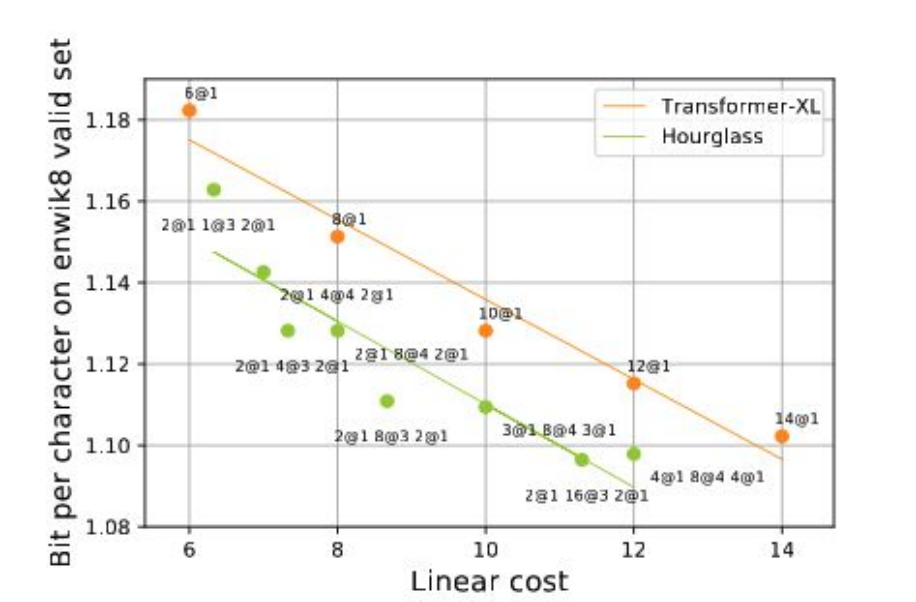
8 layers, sequence shortened by 2 = 8 * n ^ 2 / 4 = 2 * n^2

18 layers, sequence shortened by 3 = 18 * n ^ 2 / 9 = 2 * n^2

Ablation study - Shortening strategies

Consistently it is good to:

- Shorten at least once
- Shorten by a factor of at least 3
- Keep more than 2-3 vanilla layers



Intuition behind Hourglass - why it works, or when it doesn’t?

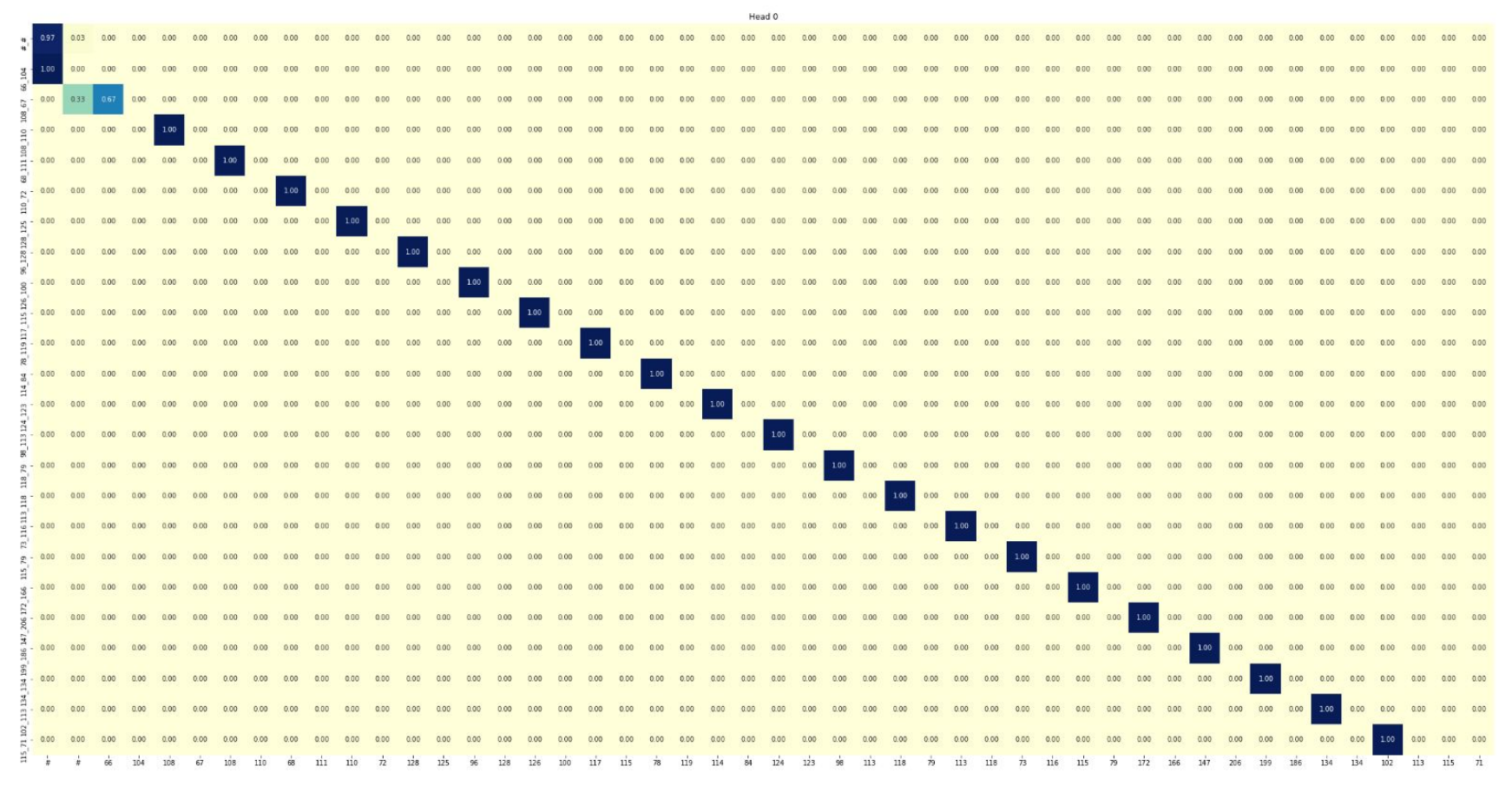
Hierarchical prior

- 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
- We see that in some cases, given fixed amount of parameters, our hierarchical model is still slightly better than baseline

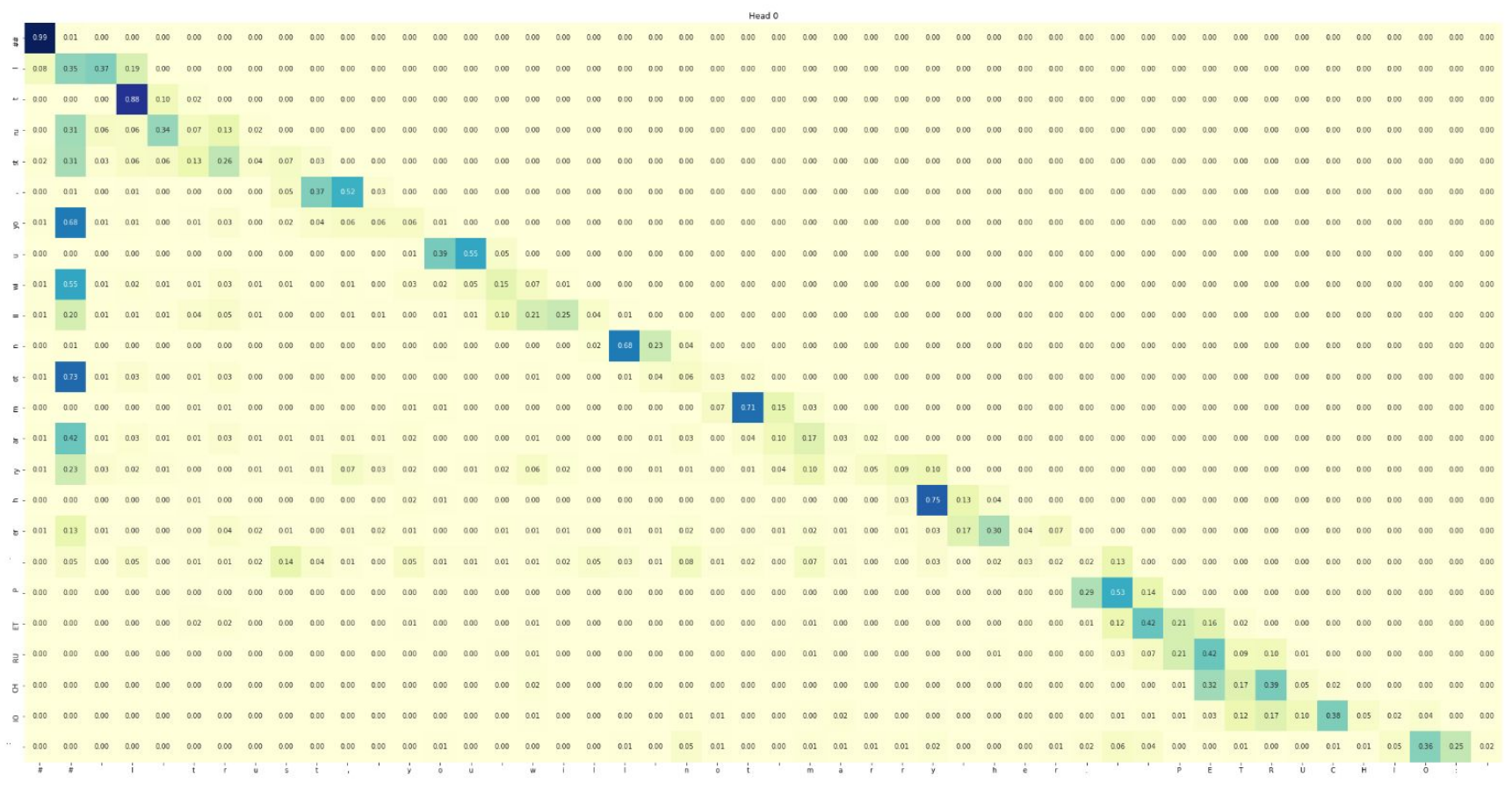
More parameters

- The model has more layers which implies it can memorize more information in its parameters which might simply lead to better results on some datasets
- The computation burden is shifted from attention to feedforward layers which can be a better prior for some tasks
- More layers can perform more multi-step reasoning, it can help even if the data the layers operate on is compressed

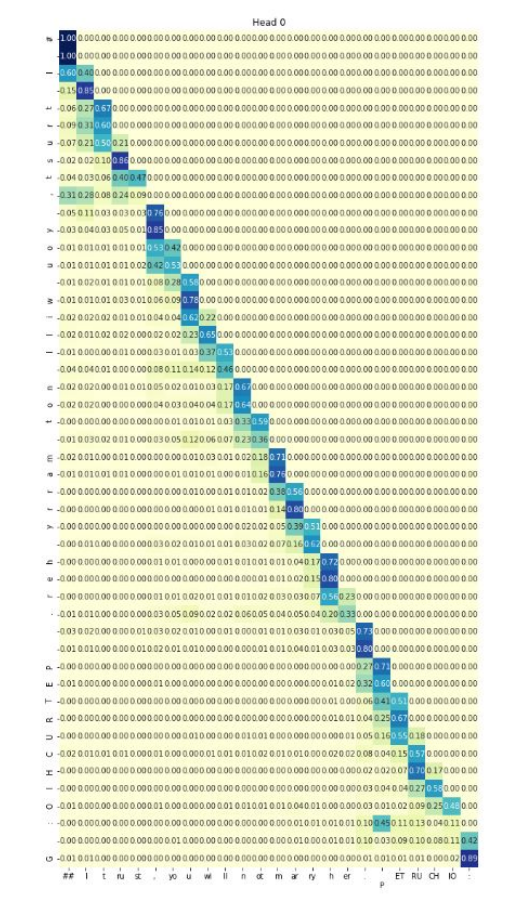
Shortcomings - fixed-size pooling - greediness

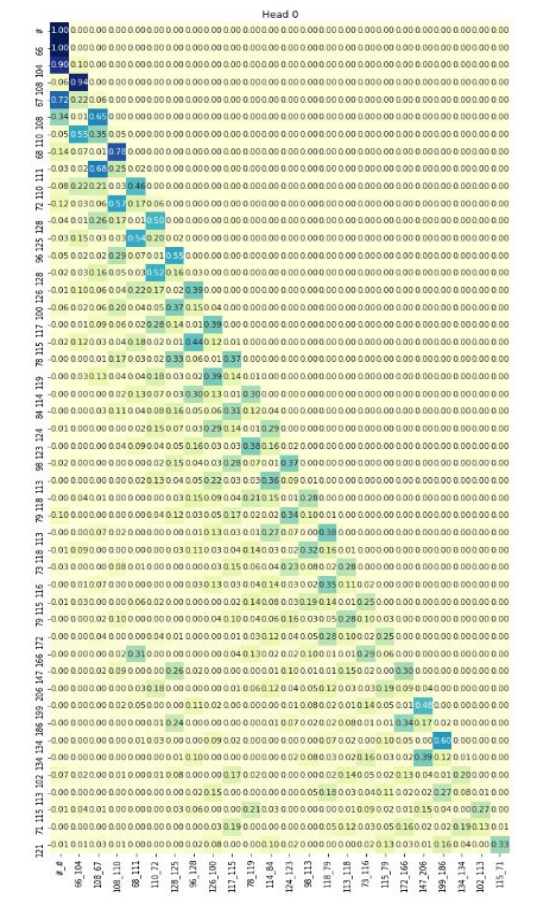


Shortcomings - fixed-size pooling



Shortcomings - fixed-size pooling - greediness (attention upsampling layer)



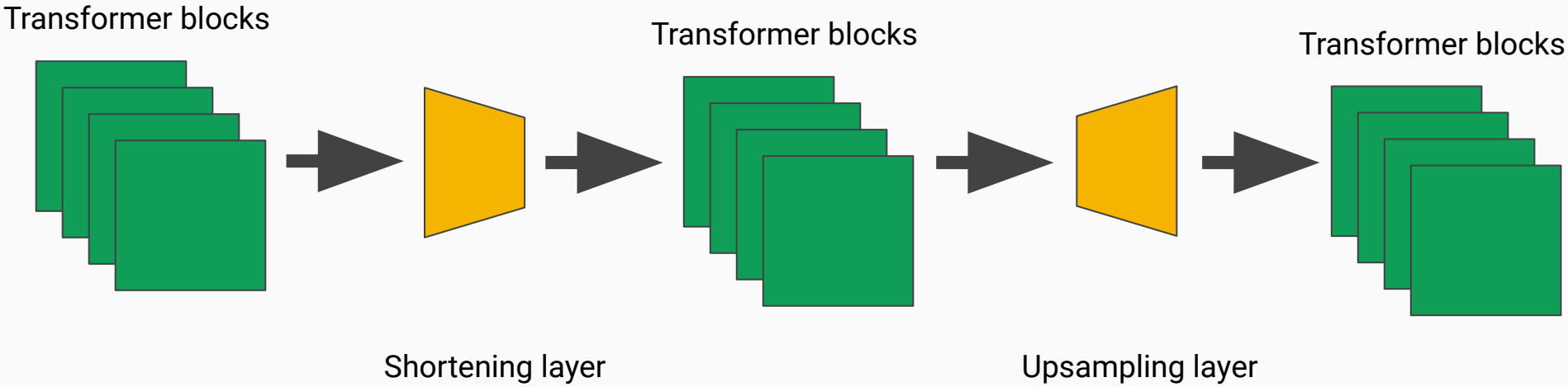


Towards dynamic pooling in transformers (ongoing research)

- We hypothesize that fixed-size pooling is one of the reasons preventing us from building multi-level hierarchies that work better than just 2-level ones.
- Intuition: on character/byte-level data, shortening by 3 or 4 even if it is fixed-size, is a very useful prior (grouping rgb channels in images, grouping chars/bytes to subwords). On token-level data, this no longer holds and the model needs to figure out the right hierarchy for the data - preferably in a dynamic, content-based way.
- Our high-level aim is to replace tokenizers in NLP and use dynamic pooling to train hierarchical transformer models, surpassing baselines based on fixed tokenizers

Shorten factor dropout - a remedy for fixed-size pooling?

Sample the shorten factor from {2, 3, 4} in each training step

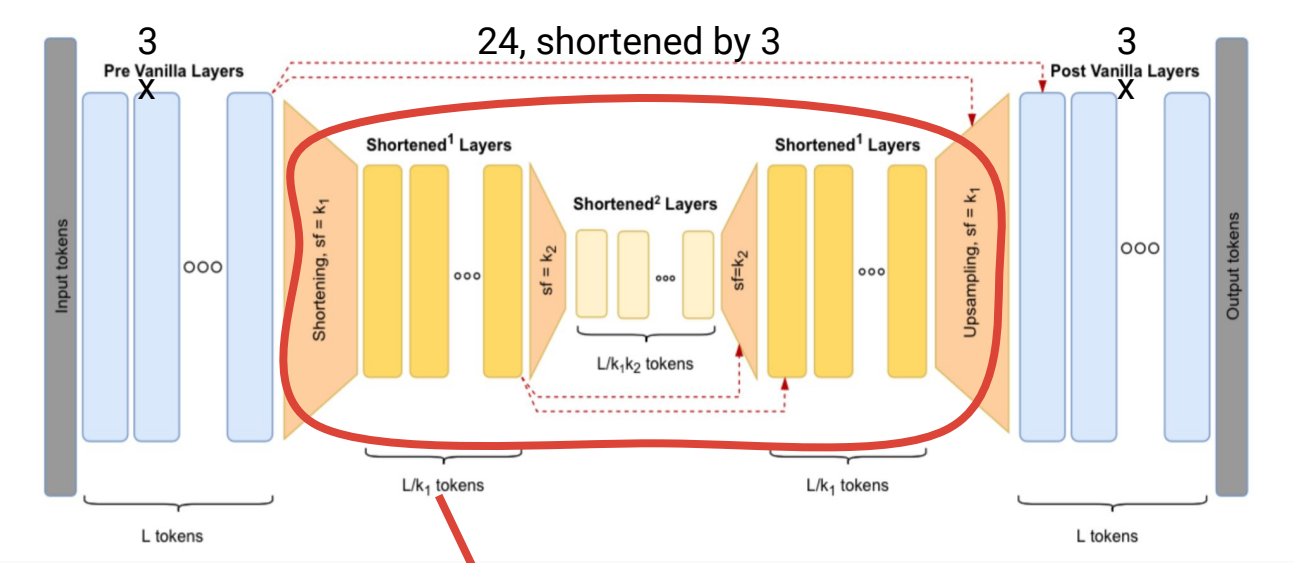


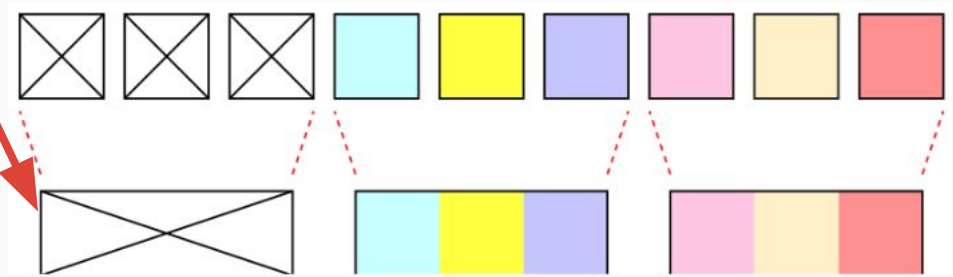
Shorten factor dropout - results

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.

Shortened layers = faster inference





The evaluation in sub-network changes every 3 steps

So don't run this part in each time step

Only once every 3 steps

Takeaway message

We don’t need to operate on original granularity of the sequence throughout the whole layer stack.

Takeaway message

We don’t need to operate on original granularity of the sequence throughout the whole layer stack.

Let’s impose hierarchical structure to process the input and use the savings to have more parameters!

Takeaway message

We don’t need to operate on original granularity of the sequence throughout the whole layer stack.

Let’s impose hierarchical structure to process the input and use the savings to have more parameters!

Hierarchical Transformers are better Language Models

Bibliography

- https://arxiv.org/abs/2110.13711 - our paper
- https://arxiv.org/abs/1706.03762
- https://arxiv.org/abs/1901.02860
- https://arxiv.org/abs/2006.03236
- https://arxiv.org/abs/2001.04451
- Visuals for the language modeling/transformers intro taken from:
  - https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture06-rnnlm.pdf
  - https://jalammar.github.io/illustrated-gpt2/
  - http://peterbloem.nl/blog/transformers
  - https://www.youtube.com/watch?v=piT1_k8b9uM