Efficient Transformers with Dynamic Token Pooling

Piotr Nawrot[†]   **Jan Chorowski**[‡]   **Adrian Łańcucki**[◇♣]   **Edoardo M. Ponti**[†]
[†]University of Edinburgh   [‡]Pathway   [◇]NVIDIA   [♣]University of Wrocław
piotr.nawrot@ed.ac.uk

Abstract

Transformers achieve unrivalled performance in modelling language, but remain inefficient in terms of memory and time complexity. A possible remedy is to reduce the sequence length in the intermediate layers by pooling fixed-length segments of tokens. Nevertheless, natural units of meaning, such as words or phrases, display varying sizes. To address this mismatch, we equip language models with a dynamic-pooling mechanism, which predicts segment boundaries in an autoregressive fashion. We compare several methods to infer boundaries, including end-to-end learning through stochastic re-parameterisation, supervised learning (based on segmentations from subword tokenizers or spikes in conditional entropy), as well as linguistically motivated boundaries. We perform character-level evaluation on texts from multiple datasets and morphologically diverse languages. The results demonstrate that dynamic pooling, which jointly segments and models language, is both faster and more accurate than vanilla Transformers and fixed-length pooling within the same computational budget.

1 Introduction

The Transformer architecture (Vaswani et al., 2017) lies at the heart of cutting-edge generative models, such as GPT-3 (Brown et al., 2020) for text and DALL·E 2 (Ramesh et al., 2022) for images. Its success can be largely attributed to the ability to leverage a considerable amount of data, which yields performance gains (Kaplan et al., 2020) and emergent abilities (Wei et al., 2022) in accordance with well-established scaling laws. Nonetheless, the time and memory efficiency of Transformers remains constrained by their algorithmic complexity of $\mathcal{O}(l^2 n)$, where l stands for sequence length and n for the number of layers.

To remedy this shortcoming without renouncing the expressivity of a deep model, the quadratic self-attention can be sparsified (Child et al., 2019; Roy et al., 2021; Ren et al., 2021) or linearly approximated (Beltagy et al., 2020). Hourglass Transformers (Nawrot et al., 2022) provide an alternative solution, where the sequence length is reduced in the intermediate layers by merging fixed-size groups of tokens, similar to (Dai et al., 2020). These pooled representations are up-sampled back to the original length in order to generate sequences in an auto-regressive fashion (Ronneberger et al., 2015).

Nevertheless, pooling groups of fixed size is sub-optimal in several respects. First, these groups are misaligned with linguistic primitives: units of meaning such as morphemes, words, phrases, and clauses vary in size. Second, the elements of a sequence may carry different degrees of information (for instance, silence and voice in speech). Ideally, the model should perform *hierarchical* computation, relying on the same abstractions as human processing of language, and *conditional*, by allocating resources to sub-sequences in proportion to the model uncertainty. In this work, we demonstrate that dynamic pooling results not only in higher shortening rates of input sequences, and thus increased efficiency, but also superior performance in next token prediction due to adopting the correct inductive bias in grouping tokens.

To this end, we propose a new Transformer variant that jointly learns token sequences and dynamically pools them into latent groupings of variable size (Figure 1). Crucially, the segmentation must preserve the auto-regressive property, and typical subword tokenizers cannot be applied to incomplete sequences during generation. Rather, we learn a neural boundary predictor during training: 1) supervised by tokenizers such as Unigram (Kudo, 2018); 2) supervised by spikes in the conditional entropy of the predictive distribution, which ensure that the computation is adaptive to the level of uncertainty of the sequence model; 3) end-to-end through stochastic re-parameterisation (Maddison et al., 2017; Jang et al., 2017); 4) use natural data
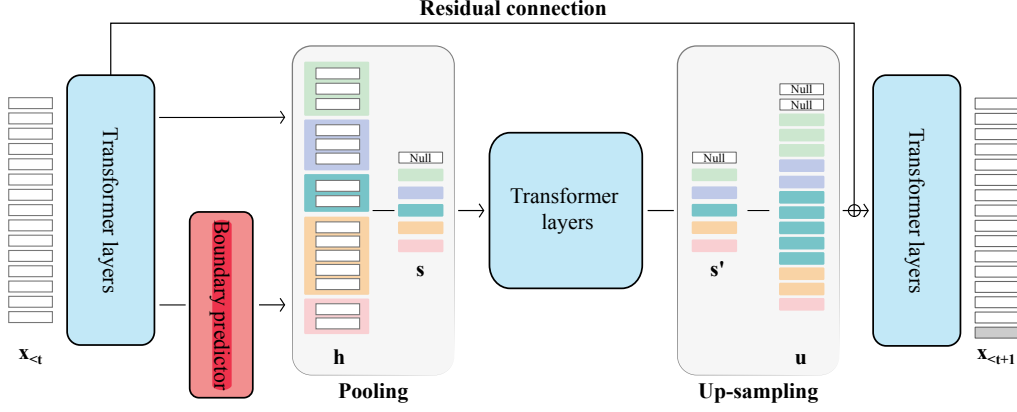
Figure 1: The architecture of a dynamic-pooling Transformer, which jointly performs language modelling and token segmentation. The boundary predictor predicts segment boundaries and pools together groups of variable length by averaging. The shortened sequence is processed efficiently by a series of intermediate layers, then up-sampled back to the original length via duplication. The model generates the next token $\boldsymbol{x}_t$ in the same resolution as the input.

boundaries such as whitespaces, which separate words in many scripts, without a predictor.

To validate our model, we experiment with character-level language modelling of text in several English benchmarks, including text8 (Mahoney, 2006), CC-100 (Wenzek et al., 2020), and wiki40b (Guo et al., 2020), as well as in a series of languages representing different morphological types: Finnish, Hebrew, and Vietnamese. We find that dynamic pooling not only achieves lower time and memory complexity, but even surpasses the performance of vanilla Transformers and fixed-size pooling Transformers in most benchmarks by statistically significant margins.

Overall, our results indicate a promising direction to further accelerate training and therefore facilitate scaling. A FAQ section about our methods, findings, and the experimental setup is available in Appendix A. We release the code at https://github.com/PiotrNawrot/dynamic-pooling.

2 Background

2.1 Language Modelling with Transformers

Let $\boldsymbol{x} = (x_1, \ldots, x_l)$ denote the input sequence. A language model assigns a probability value to any possible sequence of tokens from a vocabulary $\mathcal{V}$. The parameters of a model $\boldsymbol{\theta}$ are optimised to maximise the aggregate probability of all $\boldsymbol{x} \in \mathcal{V}^*$ in the training set $\mathcal{D}$:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{t=1}^{l} \log p(x_t \mid \boldsymbol{x}_{<t}, \boldsymbol{\theta}), \qquad (1)$$

where t indexes time steps. In our experiments, $\boldsymbol{\theta}$ corresponds to the parameters of an autoregressive Transformer model (Vaswani et al., 2017).

A key advantage of Transformers is their ability to scale, which ultimately reaps the largest benefits according to (Sutton, 2019)'s 'bitter lesson' and reveals surprising emergent capabilities of language models (Kaplan et al., 2020; Wei et al., 2022). Nevertheless, the algorithmic complexity of self-attention, $\mathcal{O}(l^2)$ where l is the length of the sequence, creates a bottleneck. To alleviate this cost, previous work (Clark et al., 2022; Tay et al., 2022; Nawrot et al., 2022) proposed to reduce the sequence length after the initial layers by pooling together groups of tokens. A single shortening by a factor k reduces the complexity to $\mathcal{O}(\frac{l^2}{k^2})$. This allows for increasing either the model efficiency or its depth within the same compute budget.

2.2 Hourglass Transformer

Naïve length reduction through pooling would reduce the length of output, however language models operate with the same input and output resolutions. For this reason, (Nawrot et al., 2022) introduced the Hourglass Transformer composed of three blocks of Transformer layers, which downsample, process, and upsample the tokens back to the original granularity. The first block encodes each input token x_t into $\boldsymbol{h}_t$. Afterwards, groups of adjacent tokens of fixed length k are mean-pooled

to form $\lceil \frac{l}{k} \rceil$ representations $\boldsymbol{s}$:

$$\boldsymbol{s}_m = \frac{1}{k} \sum_{i=mk-k+1}^{mk} \boldsymbol{h}_i \qquad (2)$$

Next, each pooled representation $\boldsymbol{s}_m$ is processed by the middle block of Transformer layers, which operates with complexity $\mathcal{O}(\frac{l^2}{k^2})$, yielding $\boldsymbol{s}'_m$. This sequence is up-sampled to its original resolution by duplication: $\boldsymbol{u}_t = \boldsymbol{s}'_{\lceil \frac{t-k+1}{k} \rceil}$, and added to the hidden representations $\boldsymbol{h}$ from before shortening through a skip connection, and passed to the third block.

Note that we subtract $k - 1$ from the index. This is because pooling and up-sampling in an autoregressive model pose a risk of data leakage from the future to the past. In fact, up-sampled representations might encompass future tokens if no measures are taken to prevent this. As a remedy, Hourglass Transformer shifts the up-sampled sequence to the right by $k-1$ positions, and pads it with a learnable null-group representation $\boldsymbol{u}_0$ at the beginning. This is sufficient to satisfy the autoregressive property in the fixed pooling scenario.[1]

Hourglass Transformer was shown to improve time and space complexity in a number of language and image modelling tasks, for a given parameter count. However, this came at the expense of degrading the perplexity of the language model, especially with shortening factors $k > 2$. We conjecture that this undesirable side effect is due to two main reasons. Firstly, the distribution of lengths of natural units of meaning such as morphemes and phrases in natural languages is uneven: for instance, word length is correlated with its frequency (Zipf, 1949; Bentz and Ferrer-i Cancho, 2016). Secondly, information content tends to be distributed uniformly across units of meaning (Meister et al., 2021).

As a consequence, fixed pooling creates segments with incongruous boundaries and unequal information content. For instance, in speech, this results in giving silence and voice the same importance. Instead, an ideal model should allocate compute *conditionally* on the information content of a given token. This would also ultimately lead to interpreting language *hierarchically* based on the same abstractions that humans adopt for language processing. Hence, we present a method to enable variable-length pooling and up-sampling in autoregressive language models.

---

[1] We refer to (Nawrot et al., 2022) for more details.

3 Dynamic-Pooling Transformer

3.1 Boundary Prediction

In order to augment the Hourglass architecture with variable-size pooling, we seek to find a sequence of segment boundaries $\boldsymbol{b} \in \{0, 1\}^l$ for every input $\boldsymbol{x}$. Let $b_t = 1$ denote a segment boundary between elements x_t and x_{t+1}. The boundary predictor is implemented as a Multi-Layer Perceptron with parameters ϕ. As shown in Figure 1, this module maps each representation $\boldsymbol{h}_t$ encoded by the first stack of Transformer layers into a Bernoulli probability distribution:

$$\hat{b}_t = p(b_t{=}1) = \mathrm{sigmoid}\left(\mathrm{MLP}_\phi\left(\boldsymbol{h}_t\right)\right). \qquad (3)$$

Since segment boundaries are discrete, sampling from this distribution is not differentiable with respect to the model perplexity. Hence, we optimise this latent variable through stochastic reparametrisation (Jang et al., 2017; Maddison et al., 2017) via hard Gumbel-sigmoid (Section 3.1.1), jointly learning the language model and boundary predictor. We favour this solution over a score-function estimator of the gradient, as it suffers from high variance and computation costs due to sampling (Schulman et al., 2015).

As an alternative, we explore training the boundary predictor module with a binary cross-entropy loss with respect to two different sources of supervision: a Unigram tokenizer (Section 3.1.2) and spikes in conditional entropy (Section 3.1.3). Finally, we consider resorting to linguistically inspired boundaries (Section 3.1.4). During training and evaluation, we perform maximum likelihood inference for these variables. In other words, each $\hat{b}_t$ from Equation (3) is rounded to the closest binary scalar such that $b_t = \lfloor \hat{b}_t \rceil$.

3.1.1 Segmenting with Gumbel-Sigmoid

In order to learn the input segmentation end-to-end based on the model perplexity, we can reparameterise the Bernoulli distribution of Equation (3) by injecting stochasticity in this form:

$$\hat{b}_t = \mathrm{sigmoid}\left[\log \frac{\hat{b}_t\, u}{(1 - \hat{b}_t)\,(1 - u)}^{1/\tau}\right]$$
$$u \sim \mathrm{Uniform}(0, 1). \qquad (4)$$

where τ is the temperature, a hyper-parameter. This estimator, however, is biased and might lead to sub-optimal results. As a consequence, we also propose methods based on supervised learning of the boundary predictor in the following sections.
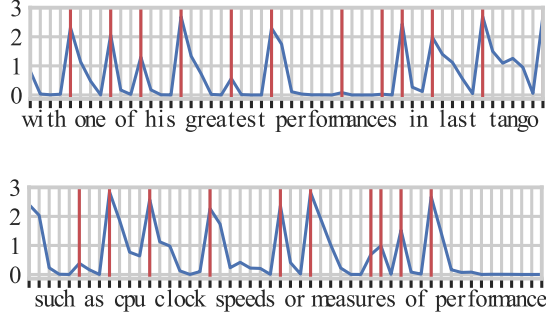
Figure 2: **Entropy of a Transformer character-level language model in two text segments**. Red vertical lines indicate the boundaries according to spikes in conditional entropy. Most of them coincide with whitespaces, due to the high uncertainty at word starts, but they also fall after morphemes like '*great*' or '*measure*'. Segmentation may vary based on the context, e.g., of the word '*performance*'.

3.1.2 Segmenting with Subword Tokenizers

Widespread algorithms for extracting variable-length boundaries for text are subword tokenizers, including Unigram (Kudo, 2018), Byte Pair Encoding (BPE; Sennrich et al., 2016), and WordPiece (Schuster and Nakajima, 2012). However, these create subwords greedily, and might change the segmentation of a given sequence prefix after more tokens are observed. For instance, consider the phrase '*civil aviation*'. A Unigram model might segment its prefix '*civil aviatio*' differently before and after observing the character '*n*':

```
_civil _a vi ati o
_civil _a vi ation
```

During training an entire sentence is tokenized, but during inference a prefix is extended one character at a time and re-tokenized, possibly changing the boundaries like in the example above. Hence, deploying off-the-shelf tokenizers naïvely during inference does not recover the oracle segments and creates a mismatch between training and evaluation boundaries.

As a remedy, we provide the training tokenization as supervision to our autoregressive boundary predictor instead. More specifically, we employ a Unigram tokenizer (Kudo, 2018), as it aligns with morphological units better than other algorithms (Bostrom and Durrett, 2020). To prevent subword units from crossing word boundaries, we split the text on whitespace characters beforehand. Vocabulary size is a tunable hyper-parameter which offers different efficiency–performance trade-offs.

3.1.3 Segmenting with Entropy Spikes

As an alternative to providing supervision through Unigram, we also propose a new segmentation method based on spikes of conditional entropy, which is agnostic about the presence of natural boundaries (such as whitespaces) or the availability of tokenizers. These properties make it suitable for other modalities in addition to text, such as speech and vision. Moreover, this enables top-down supervision and end-to-end training without external dependencies.

Intuitively, in natural language the information content tends to be spread evenly throughout a sentence, to facilitate communication. The conditional entropy is the expectation of such information content over the tokens in the vocabulary:

$$\mathcal{H}(x_t \mid \boldsymbol{x}_{<t}) = \sum_{x \in \mathcal{V}} p(x_t \mid \boldsymbol{x}_{<t}) \underbrace{(-\log p(x_t \mid \boldsymbol{x}_{<t}))}_{\text{information content}}$$

(5)

Therefore, peaks in this conditional entropy provide indications of surprisal, and can serve as natural boundaries between segments. More formally, let $\mathcal{H}_t$ be the conditional entropy at time t. We select local spikes by comparing their value within a (left) window of size k. We place boundaries according to the following conditions:

$$b_t = \begin{cases} 1 & \text{if } \mathcal{H}_t > \mathcal{H}_i \quad \forall i \in \{t-k, \ldots, t-1\} \\ 0 & \text{otherwise.} \end{cases}$$

(6)

Empirically, entropy spikes in language models overlap with word boundaries to a significant degree (Hutchens and Alder, 1998). However, they are also more flexible as they enable conditional computation based on the model's confidence about its next token prediction. As an example of segmentation based on entropy spikes, consider Figure 2.

3.1.4 Linguistically Inspired Segments

Finally, perhaps the most straightforward source of segmentation is word boundaries. In fact, in many scripts, these are marked by whitespace characters.[2] The simplicity of this method of segmentation comes with the obvious drawback of not providing control over the rate of shortening, while we found that the optimal rate varies with the language. Hence its efficiency–performance trade-off is not tunable.

---

[2]Several scripts such as Chinese characters, however, do not adopt this convention.

Segment boundaries are placed in between two symbols. In our experiments, we put a boundary *after* a whitespace character. Thus, we do not need to train a boundary predictor, since predicting a whitespace character is a signal to close the group in the next iteration of auto-regressive generation. This would not be possible, had we chosen to put a boundary before a whitespace character.

3.2 Pooling and Up-sampling

In the pooling step (Figure 1) a generated sequence of boundaries b is used to pool the tokens belonging to the same segment by averaging. Thus, we form $\sum_{t=1}^{l} b_t + 1$ shortened representations s, which are then passed to the middle block of Transformer layers. Note that for Gumbel-sigmoid, to keep pooling differentiable, we algebraically manipulate $b \in \mathbb{R}^l$ into $B \in \mathbb{R}^{l \times 1 + \sum_t b_t}$, i.e. a binary matrix that maps from the original length to the shortened length, following (Bhati et al., 2021). The cell B_{ij} is 1 if token i is merged into the j-th group, and 0 otherwise. Thus, $s = hB / \sum_i B_{i\star}$, where the denominator unit-normalises the matrix columns.

To obtain the up-sampled representation u_t while preserving the autoregressive property, we calculate the largest index m so that the output of the middle block s'_m does include future information: $u_t = s'_m$, where $m = \sum_{i=1}^{t} b_i$. As a consequence, a segment representation s'_m can only be added to the last token pooled into group m. For all the other non-final tokens, we take the representation of a previous segment s'_{m-1}. Similar to Hourglass, the representation for the first (null) group s_0 is a learnable vector. Afterwards, u_t is added to the highway layer representation h_t.

3.3 Auxiliary Objectives

In addition to minimising the language modelling loss with respect to the parameters θ as shown in Equation (1), we use auxiliary objectives to train the boundary predictor parameters ϕ. For supervised learning with subword tokenizers and entropy spikes, we minimise the cross-entropy between predicted boundaries b and gold ones. For end-to-end learning with Gumbel softmax, we introduce a regularizer based on a Binomial prior. Let $k = \sum_t b_t$:

$$\text{Binomial}(\alpha; l, k) = \binom{l}{k} \alpha^k (1 - \alpha)^{l-k} \quad (7)$$

where $\alpha \in [0, 1]$ is a hyper-parameter. This regularizer prevents the model from collapsing into trivially predicting each position as a boundary.

4 Experimental Setup

4.1 Datasets

In addition to English, we evaluate our model on data in three languages, which represent different morphological types: Finnish for agglutinative, Hebrew for introflexive, and Vietnamese for isolating. Thus, we ensure that dynamic pooling is robust to different word length distributions. For English, we use text8 (CC-BY-SA) (Mahoney, 2006), CC-100 (MIT) (Conneau et al., 2020) and wiki40b (CC-BY-SA) (Guo et al., 2020) as they are established benchmarks for character-level language models. For the rest of the languages, we use the corresponding subsets of wiki40b. To make results comparable across languages and prevent data imbalance, we limit the size of CC-100 and wiki40b to the first 400M tokens of the training set and the first 2M tokens of the validation set. We retain the original splits for each dataset.

For all datasets and languages, we follow the same pre-processing steps of (Mahoney, 2006) for creating text8. Specifically, for each language we keep only the characters from its script, as well as whitespace and an end-of-line. The text is lower-cased, and the digits are spelt out in the target language. For wiki40b, we also remove special structural markers and normalise homoglyphs. Finally, for Hebrew we also remove diacritics as they are not required to understand the text. This way, we filter out excerpts in different languages, which are known to contaminate noisy multilingual texts (Kreutzer et al., 2022). The pre-processing scripts can be found as part of our code.

4.2 Models

All of our experiments, except for the scaling ablation, use 12-layer Hourglass Transformers with 2 layers in the first block, 8 layers in the second block which operates on shortened sequences, and 2 layers in the final block, following (Nawrot et al., 2022). For every Transformer layer, the hidden dimension is 512, the intermediate feed-forward dimension is 2048. Self-attention is split into 8 heads. We use a post-norm architecture, GELU activation function (Hendrycks and Gimpel, 2016) in feed-forward layers and the relative attention parametrisation from Transformer XL (Dai et al., 2019). In total, the model has ~41M parameters.

The boundary predictor is a 2-layer MLP that takes a hidden state as input and outputs a scalar at every time step. For models with dynamic pooling,

this module adds around 1M additional parameters. We use the SentencePiece (Kudo and Richardson, 2018) library to train Unigram segmentation for every dataset separately. We detect spikes in conditional entropy according to a window of size $k = 2$, which we select from range $k = 1 \dots 4$ for optimal BPC on text8. For Gumbel Sigmoid, we set the prior probability of a boundary α to 0.2 for English, Vietnamese and Hebrew, and 0.37 for Finnish. The Gumbel temperature parameter was set to 0.5 in all experiments. For Unigram vocabulary size, we set $|\mathcal{V}| = 10000$ for English and Vietnamese and $|\mathcal{V}| = 200$ for Finnish and Hebrew. We list training hyper-parameters in Appendix B.

5 Results

The results for the experiments on character-level language modelling are shown in Table 1. In addition to the four proposed segmentation methods, we include a vanilla Transformer and fixed-size pooling Transformers with multiple shortening factors as baselines. Every model is evaluated with respect to two metrics: bits per character (BPC; ↓) and shortening factor (SF; ↑). The former measures the negative log-probability of the language model predictions, and thus its quality; the latter measures the average reduction of the sequence length in intermediate layers, and thus the model efficiency. Figure 5 shows how higher SF translates to lower training time and memory consumption in practice, as measured on a common GPU with an optimised model implementation.

Segmentation Methods In all the English evaluation benchmarks (text8, wiki40b, and CC-100), both whitespace-based and Unigram-based segmentations achieve the lowest BPC, outperforming both vanilla and fixed-pooling Transformers by statistically significant margins.[3] Moreover, the same two methods achieve the highest degrees of shortening. Note that for equivalent SFs, fixed-size pooling becomes detrimental to performance. The approaches based on entropy spikes and Gumbel-Sigmoid are generally inferior to the alternatives for dynamic pooling. However, for comparable shortening factors, they always outperform vanilla and fixed-pooling Hourglass models. Moreover, they make the fewest assumptions about the data and the availability of external supervision, so they might be appropriate for other domains (such as speech

---

[3]We indicate with a ⋆ wherever this is the case according to a Paired Student's t-test with $p < 0.05$.
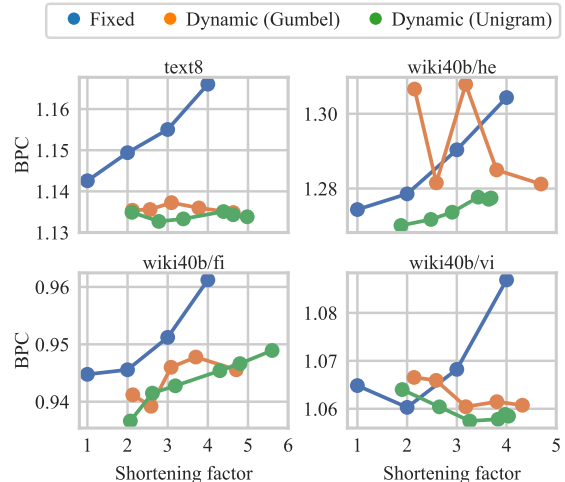

Figure 3: Test BPC (↓) and shortening factor (SF; ↑). The higher the SF, the more efficient the model is (cf. Figure 5 in the Appendix). SF increases with higher vocabulary size (Unigram) or smaller prior boundary probability (Gumbel). Dynamic pooling methods shift the Pareto front, i.e., increase performance for the same efficiency (and vice versa). Note that fixed-pooling at $k = 1$ corresponds to the vanilla Transformer model.

and vision) in future work. In general, providing a Transformer with the correct inductive bias for pooling variable-size segments not only facilitates scaling but also enhances prediction quality.

Notably, the gains resulting from whitespace segmentation are not identical in all languages, due to their inherent differences in morphological types and average word length. Shortening Factors for this method range from $3.8\times$ in introflexive Hebrew, to $7.9\times$ in agglutinative Finnish, whereas isolating Vietnamese and mildly fusional English lie in between with $4.4\times$ and $5.7\times$, respectively. The larger SFs of dynamic pooling methods translate into higher training speed, from $1.7\times$ for Unigram in Hebrew to over $2.5\times$ for whitespaces in English, while simultaneously lowering BPC. Overall, the gains from dynamic pooling are robust cross-lingually, but the optimal segmentation method may vary.

Efficiency–Performance Pareto Front While both low BPC and high SF are desirable, there exists a trade-off between them which is specific to each boundary prediction method. Hence, the ideal model should strike the right balance to improve in both respects simultaneously. Intuitively, vocabulary size in Unigram and the prior α in Gumbel-Sigmoid provide easily controllable knobs to study

| | text8 | | wiki40b | | cc-100 | | Finnish wiki40b | | Hebrew wiki40b | | Vietnamese wiki40b | |
| | | | English | | | | | | | | | |
	BPC	SF	BPC	SF	BPC	SF	BPC	SF	BPC	SF	BPC	SF
Vanilla	1.143	(1.0x)	1.091	(1.0x)	1.225	(1.0x)	0.945	(1.0x)	1.274	(1.0x)	1.065	(1.0x)
Fixed (SF=2)	1.149	(2.0x)	1.084	(2.0x)	1.224	(2.0x)	0.946	(2.0x)	1.279	(2.0x)	1.060	(2.0x)
Fixed (SF=3)	1.155	(3.0x)	1.093	(3.0x)	1.229	(3.0x)	0.951	(3.0x)	1.290	(3.0x)	1.068	(3.0x)
Fixed (SF=4)	1.166	(4.0x)	1.102	(4.0x)	1.240	(4.0x)	0.961	(4.0x)	1.304	(4.0x)	1.087	(4.0x)
Gumbel	1.136*	(4.6x)	1.080	(4.7x)	**1.212***	(4.6x)	0.941	(2.6x)	1.281	**(4.7x)**	1.061	(4.3x)
Entropy	1.138*	(4.1x)	1.083	(4.1x)	1.218*	(3.8x)	0.949	(4.1x)	1.276	(3.6x)	1.072	(4.2x)
Unigram	1.134*	(5.0x)	1.078*	(5.0x)	**1.212***	(4.8x)	**0.937**	(2.1x)	**1.270***	(1.9x)	1.058	(4.0x)
Whitespaces	**1.133***	**(5.7x)**	**1.077***	**(5.6x)**	1.214*	(5.2x)	0.955	**(7.9x)**	1.284	(3.8x)	**1.057***	(4.4x)

Table 1: Language modelling results on 3 English datasets and 3 other morphologically diverse languages. For each pair of method and dataset, we report test BPC ($\downarrow$) and average shortening factor (SF; $\uparrow$). We run each experiment 3 times with different random seeds. We mark with a star ($\star$) symbol results that are statistically better than both the vanilla Transformer baseline and fixed shortening by means of a Paired Student's t-test with $p < 0.05$. We report results based on the best hyper-parameter configuration for each language.

this interaction: as they change, so does the shortening factor. In Figure 3, we plot BPC and SF for six vocabulary sizes (200, 500, 1k, 3k, 5k, 10k) and five α values (0.20, 0.25, 0.30, 0.37, 0.45) and compare them with fixed-size pooling in Hourglass Transformers. Manifestly, dynamic pooling enhances the Pareto front by finding more optimal trade-offs between efficiency and performance. Moreover, while fixed pooling follows a similar trend cross-lingually, dynamic pooling behaves more idiosyncratically: e.g. BPC in Vietnamese and English surprisingly improves with higher SFs. During our study of the Efficiency–Performance Pareto Front, we noticed that the Gumbel-Sigmoid pooling approach exhibits greater instability compared to the Unigram-based pooling method. This can be observed through artifacts such as the spikes in BPC for Hebrew, depicted in Figure 3.

Time and Space Complexity To capture the concrete gains in efficiency of models with higher SFs, we have measured the memory consumption and training time of our PyTorch implementation of `text8` models on a typical GPU (NVIDIA GV100 32GB). The results in Figure 5 apply to dynamic-pooling (Gumbel, Whitespace, Unigram, and Entropy), fixed-pooling, and vanilla Transformers (only for SF=1). Note that these results are identical for both fixed-pooling and dynamic-pooling Hourglass for the same SF as the cost of the boundary predictor is negligible. With a shortening factor $SF = 2$, the model reduces both memory consumption and training time by over 40%, compared to a vanilla Transformer. At $SF = 4$, where dynamic-pooling Hourglass still achieves superior BPC scores, resource consumption is reduced be-

tween 50% and 60% and training is $2.5\times$ faster. This allows models to increase in size with the same compute budget (which depends on the hardware), while simultaneously benefiting their performance.

Scaling the Model We investigate if dynamic-pooling Transformers scale well in terms of model size, by adding more layers in the middle block (Figure 4). We focus on this block as it increases the model depth (and hence its capacity) while retaining a higher efficiency due to operating on shortened sequences. We find that the gains from dynamic pooling are consistent across all numbers of layers. Extrapolating from the trends, dynamic pooling holds promise to continue providing benefits even in extremely large language models.

Average-pooling vs Sub-sampling As an ablation, we also compare two different methods to represent groups of tokens when shortening the input sequence length: average pooling, used in our experiments, and sub-sampling, i.e. selecting only the last token as a representative for each group. As it emerges from Table 2, average pooling yields superior performance in all models, including both fixed and dynamic pooling Transformers.

| | Shortening | |
Segmentation	Avg-Pooling	Sub-sampling
Fixed (SF = 2)	**1.149**	1.180
Entropy	**1.138**	1.151
Whitespaces	**1.133**	1.144

Table 2: BPC results on `text8` for two shortening methods (average-pooling and sub-sampling) and three segmentation methods.
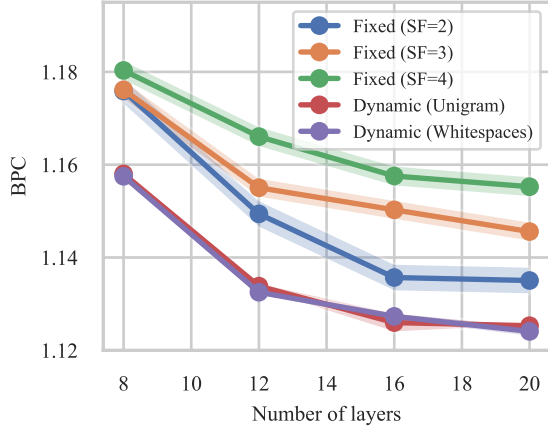
Figure 4: Test BPC on `text8` plotted against the number of Transformer layers for different shortening methods. We use two layers in the first and last transformer block and only scale the middle, downsampled block. There are 28M parameters in models with 8 layers, up to 69M parameters in models with 20 layers. For all variants we observe performance gains with dynamic pooling.

Other Efficient Transformer Models Finally, we remark that our method differs from most efficient Transformer algorithms, which reduce the quadratic complexity of attention (Child et al., 2019; Lee-Thorp et al., 2022; Choromanski et al., 2021; Wang et al., 2020), as it focuses on length reduction. While previous efficient variants tend to trade quality for efficiency, we have shown that the dynamic-pooling mechanism improves both simultaneously in our experiments. Moreover, Nawrot et al. (2022) has shown that combining both strategies yields further gains.

6 Related Work

Dynamic RNNs Our approach is inspired by variants of RNNs that process sequences at varying time scales by introducing a hierarchy of hidden units. For instance, RNNs that mimic speed-reading by introducing hidden units that can skip over some input elements (Campos et al., 2018; Seo et al., 2018). Similarly, (Chung et al., 2017) discovers the latent hierarchy of an input sequence using a stack of LSTMs. Each layer is equipped with a binary gate responsible for hard boundary detection, where lower-level boundaries determine state updates made by higher-level layers. Whenever the detector ends a segment, its representation is fed to the upper layer.

Early slow- and fast-changing units were already described by (Hihi and Bengio, 1995). Similarly,
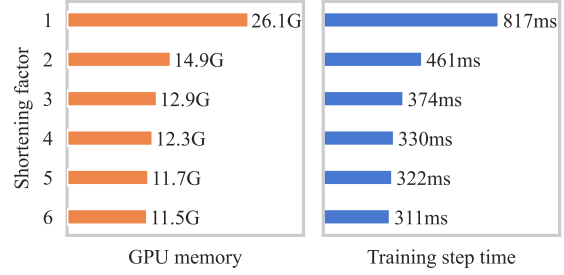

Figure 5: Memory consumption and duration of a training step for different shortening factors on English `text8`. These results apply to both dynamic pooling and fixed pooling Hourglass models, as well as vanilla Transformers (for SF=1).

Clockwork RNN (Koutnik et al., 2014) introduces a hierarchy of hidden state units that make transitions at a set of different, fixed frequencies. Adaptive Computation Time networks perform a different amount of computation on each sequence item (Graves, 2016). Both ideas were combined in Fast-Slow RNNs (Mujika et al., 2017) which can choose a heavy or light transition between timesteps.

Pooling Transformer models While pooling blocks in Transformers are related to slowly varying units in RNNs, their operation is different. RNNs suffer from unreliable transport of information across long time spans. Units that act like skip-connections over time can help them to carry information (Krueger et al., 2017). In a Transformer network, a unit at time t can directly communicate with any other unit, including previous ones, and we find it important to confirm the benefits of dynamic pooling in Transformer models.

Perhaps the most similar approach to ours is Funnel Transformer (Dai et al., 2020) which uses a similar, hourglass-shaped Transformer architecture. After passing through the first block, the data is pooled at a fixed rate, processed by the deep middle Transformer block, and up-sampled for the last block. Canine (Clark et al., 2022) has a similar three-part architecture, and processes Unicode inputs, which are downsampled with Transformer and convolution layers. (Tay et al., 2022) implements gradient-based subword tokenization within a Transformer model, which learns dynamic groupings of tokens into fixed-size groups. In (Bai et al., 2021), sentence and paragraph boundaries were used as additional conditioning for the model.

Boundary Detection   We investigate boundaries provided by an external model, derived directly from the data, or top-down from the model's entropy. (Kreuk et al., 2020) shows a bottom-up approach to phoneme segmentation task combining contrastive learning (van den Oord et al., 2019) with a method for boundary detection based on dissimilarity between subsequent frames. It was later extended by (Bhati et al., 2021) to segment the sequence of speech frames dynamically. Recently, (Cuervo et al., 2022) introduced a hierarchical sequence processing model in which units in the upper layer operate on a dynamically shortened sequence, with the shortening guided by a boundary prediction model.

(Rocki et al., 2016) control the activity of LSTM gates with the model's output cross-entropy. (Alpay et al., 2019) used a similar mechanism based on information content to guide the copying of individual activations in an LSTM network. Similarly, we employ the entropy of model predictions to choose where to insert boundaries.

7   Conclusions

We proposed a new family of language models that pool variable-size segments of tokens in the intermediate layers in order to enhance the efficiency and performance of the Transformer architecture. In particular, we learn a boundary predictor either end-to-end through stochastic re-parameterisation, through supervision (obtained from subword tokenization or spikes in the conditional entropy), or based on linguistic boundaries such as words. We evaluate this model extensively on multiple language modelling benchmarks in English and in other typologically diverse languages: Finnish, Hebrew, and Vietnamese. Compared to vanilla Transformers and fixed pooling, we observe a significant decrease in model perplexity as well as time and space complexity. This opens up the perspective to develop Transformer models capable of computing language both hierarchically, with the same abstractions humans perform at different levels of linguistic structure, and conditionally on the information content of each segment.

In the future, our dynamic-pooling Transformer can be combined with methods relying on external memory (Wu et al., 2022), encoders operating at a fine resolution (Xue et al., 2022; Tay et al., 2022), and more generally any task with long-context inputs (Shaham et al., 2022). This may further facili-

tate the scalability of current language modelling architectures.

8   Limitations

Linguistic variation   Our results are highly dependent on the target language and its morphology. For example, word boundaries might seem like an obvious choice for dynamic segmentation, and in fact they achieve the best performance in English and Vietnamese. However, for some languages like agglutinative Finnish, whitespaces are less frequent, which is detrimental to model performance. Explicit word boundaries are not available for all scripts. For example, in Chinese characters, or in modalities other than text like speech or vision, there is no obvious equivalent to whitespaces. However, segmentation based on stochastic re-parameterisation, subword tokenizers and spikes in conditional entropy overcomes these limitations.

Contiguous segments   In its current formulation, dynamic pooling only allows for merging contiguous segments of tokens in a sequence. However, this is not ideal for morphology types like Hebrew where morphemes are discontinuous: vowels are interspersed between consonant roots for inflection. Moreover, future works should consider higher levels of linguistic structure than words, such as dependency trees, for pooling. In this case, discontinuous segments may be necessary to handle non-projective syntactic dependencies.

Independent boundary decisions   The decision to emit a boundary at time step t depends on previous boundaries only indirectly through the hidden representation of the first Transformer block, as this preserves the efficiency of the boundary predictor. Instead, a recurrent model could be explicitly conditioned on previous boundary decisions, which however would negatively affect the time complexity of the language model.

Work contribution of authors

Acknowledgements

References

Tayfun Alpay, Fares Abawi, and Stefan Wermter. 2019. Preserving activations in recurrent neural networks based on surprisal. *Neurocomputing*, 342(C):75–82.

He Bai, Peng Shi, Jimmy J. Lin, Yuqing Xie, Luchen Tan, Kun Xiong, Wen Gao, and Ming Li. 2021. Segatron: Segment-aware transformer for language modeling and understanding. In *AAAI*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Chris Bentz and Ramon Ferrer-i Cancho. 2016. Zipf's law of abbreviation as a language universal. In *Proceedings of the Leiden workshop on capturing phylogenetic algorithms for linguistics*, pages 1–4.

Saurabhchand Bhati, Jesús Villalba, Piotr Żelasko, Laureano Moro-Velazquez, and Najim Dehak. 2021. Segmental contrastive predictive coding for unsupervised word segmentation. *arXiv preprint arXiv:2106.02170*.

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Víctor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang. 2018. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz

Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. 2021. Rethinking attention with performers. In *International Conference on Learning Representations*.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In *International Conference on Learning Representations*.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451.

Santiago Cuervo, Adrian Łańcucki, Ricard Marxer, Paweł Rychlikowski, and Jan Chorowski. 2022. Variable-rate hierarchical CPC leads to acoustic unit discovery in speech. *arXiv preprint arXiv:2206.02211*.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. 2020. Funnel-Transformer: Filtering out sequential redundancy for efficient language processing. *Advances in Neural Information Processing Systems*, 33:4271–4282.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. 2020. Wiki-40B: Multilingual language model dataset. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2440–2452.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

Salah Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems*, volume 8.

Jason L. Hutchens and Michael D. Alder. 1998. Finding structure via compression. In *New Methods in Language Processing and Computational Natural Language Learning*.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*.

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. *arXiv preprint arXiv:2001.08361*.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. 2014. A clockwork RNN. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1863–1871.

Felix Kreuk, Joseph Keshet, and Yossi Adi. 2020. Self-Supervised Contrastive Learning for Unsupervised Phoneme Segmentation. In *Interspeech 2020*, pages 3700–3704.

Julia Kreutzer, Isaac Caswell, Lisa Wang, Ahsan Wahab, Daan van Esch, Nasanbayar Ulzii-Orshikh, Allahsera Tapo, Nishant Subramani, Artem Sokolov, Claytone Sikasote, Monang Setyawan, et al. 2022. Quality at a Glance: An Audit of Web-Crawled Multilingual Datasets. *Transactions of the Association for Computational Linguistics*, 10:50–72.

David Krueger, Tegan Maharaj, Janos Kramar, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Christopher Pal. 2017. Zoneout: Regularizing RNNs by randomly preserving hidden activations. In *International Conference on Learning Representations*.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2022. FNet: Mixing tokens with Fourier transforms. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4296–4313.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*.

Matt Mahoney. 2006. Large text compression benchmark. http://www.mattmahoney.net/dc/text.html. (Online; accessed November 5, 2022).

Clara Meister, Tiago Pimentel, Patrick Haller, Lena Jäger, Ryan Cotterell, and Roger Levy. 2021. Revisiting the Uniform Information Density hypothesis. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 963–980.

Asier Mujika, Florian Meier, and Angelika Steger. 2017. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 30.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical transformers are more efficient language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1559–1571.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*.

Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang, Jure Leskovec, Dale Schuurmans, and Bo Dai. 2021. Combiner: Full attention Transformer with sparse computation cost. In *Advances in Neural Information Processing Systems*, volume 34, pages 22470–22482.

Kamil Rocki, Tomasz Kornuta, and Tegan Maharaj. 2016. Surprisal-driven zoneout. *arXiv preprint arXiv:1610.07675*.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. 2015. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, volume 28.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Neural speed reading via skim-RNN. In *International Conference on Learning Representations*.

Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, et al. 2022. Scrolls: Standardized comparison over long language sequences. *arXiv preprint arXiv:2201.03533*.

Richard Sutton. 2019. The bitter lesson. http://incompleteideas.net/IncIdeas/BitterLesson.html. (Online; accessed November 5, 2022).

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2022. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2019. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. In *International Conference on Learning Representations*.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

George Kingsley Zipf. 1949. *Human behavior and the principle of least effort: An introduction to human ecology*. Addison–Wesley.

Appendix

A Frequently Asked Questions

A.1 Pros and Cons of shortening methods

	Pros	Cons
Fixed	- Simple	- Sub-optimal results, especially for SF > 2
Whitespaces	- Linguistically inspired - Does not require a boundary predictor	- Not available in all languages, e.g., Chinese - No control over SF
Entropy	- Better performance than Fixed - Suitable for other modalities such as speech and vision	- Requires a boundary predictor - Worse than Unigram and Gumbel
Unigram	- Best trade-off between efficiency and performance - Shown to align well with morphological units	- Requires a boundary predictor - Works only in sequential discrete data - Requires training a tokenizer up-front
Gumbel	- Good trade-off between efficiency and performance - Suitable for other modalities such as speech and vision	- Requires a boundary predictor - High variance performance

Table 3: Pros and cons of different shortening methods. SF is a shorthand for Shortening Factor.

A.2 What is the ultimate segmentation method?

While Whitespace offers the best performance in many cases, this is not always true even in the linguistic domain. In agglutinative languages (e.g., Finnish), words are longer than in English, which has a detrimental effect on the Whitespace method. For such languages, other dynamic methods that allow for controlling the shortening factor (SF), such as Unigram, are better suited. Moreover, languages with non-Latin scripts (like Chinese) may lack explicit whitespaces. For modalities different from text, such as speech and vision, Gumbel and Entropy are to be favoured as they do not assume the discreteness of the input sequence.

A.3 Why evaluating on language modelling rather than downstream tasks?

Since we present a proof of concept for dynamic-pooling Transformers, we limit the experiments to language modelling because: 1) it is a foundational NLP task; 2) previous efficient Transformer variants were evaluated on similar benchmarks. Crucially, there is a strong correlation between performance in language modelling and downstream tasks.

A.4 How do you ensure that the results are reliable?

Our code is based on the optimised, open-source implementation of Transformer-XL from NVIDIA (Apache 2.0 License), which reproduces the scores reported by (Dai et al., 2019). Our implementation of the fixed-pooling Hourglass Transformer model similarly reproduces the results from (Nawrot et al., 2022). We make our code publicly available, under the Apache 2.0 License, inheriting from the original source, to ensure the reproducibility of our results. Moreover, memory utilisation was measured by controlling resource allocation on GPUs (Figure 5) rather than through a naive `nvidia-smi` readout, as this would overestimate the reserved buffers.

B Hyper-parameters

Following (Dai et al., 2019), we train for $2 \cdot 10^5$ steps with a batch size of 8 and a learning rate $2.5 \cdot 10^{-4}$ on 2x NVIDIA RTX 3080. Each training run took from approximately 12h to 30h, depending on the configuration. We use a linear warm-up schedule for the first 4k steps, followed by a single-cycle cosine scheduler. We use an Adam optimiser with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e{-}8$, and clip the gradients at 0.25. We apply a 0.1 dropout rate in the attention matrix and feed-forward layers. Before every epoch, we cyclically shift the text stream, divide it into non-overlapping chunks of 2048, and shuffle. During the evaluation, to provide context to the model, we split the test set into partially overlapping sequences of size $l = 2048$ with a step size of 512 and calculate the model perplexity only over the last 512 tokens.