Toucan: Token-Aware Character Level Language Modeling

Anonymous ACL submission

Abstract

Character-level language models obviate the need for separately trained tokenizers, but efficiency suffers from longer sequence lengths. Learning to combine character representations into tokens has made training these models more efficient, but they still require decoding characters individually. We propose Toucan, an augmentation to character-level models to make them *"token-aware"*. Comparing our method to prior work, we demonstrate significant speed-ups in character generation without a loss in language modeling performance. We then explore differences between our learned dynamic tokenization of character sequences with popular fixed vocabulary solutions such as Byte-Pair Encoding and WordPiece, finding our approach leads to a greater amount of longer sequences tokenized as single items. Code and data will be released at time of submission.

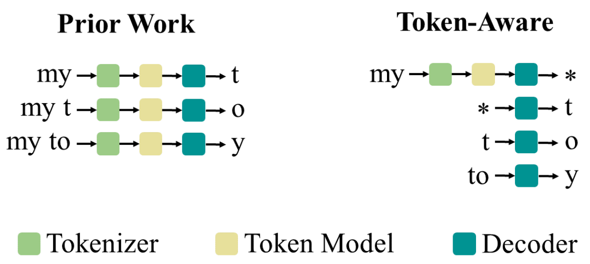

Figure 1: Token-aware generation does not require reprocessing the entire sequence at each step for every character. *: special end-of-token character.

1 Introduction

Most modern language models (LMs) are trained using the transformer architecture (Vaswani et al., 2017) on a fixed vocabulary of tokens (Brown et al., 2020; Devlin et al., 2019; Touvron et al., 2023; Penedo et al., 2023). Tokenizers and language models are commonly trained using separate objectives. For example, Byte-Pair-Encoding (BPE) (Sennrich et al., 2016) selects tokens based on their frequency and not by their ability to predict the next token in a sequence. The fixed vocabulary and misaligned objectives suggest that current tokenization schemes are potentially suboptimal.

Training transformers directly on character or byte-level sequences removes the need for tokenization, but the increased sequence length suffers from the transformer's quadratic complexity. Several variations have been developed to address the issue by pooling fixed-length contiguous character representations into smaller sets of patch representations (Dai et al., 2020; Nawrot et al., 2022; Yu et al., 2023; Tay et al., 2022). Although this can improve efficiency, Edman et al. (2022) discuss the limitations of length, position, and morpheme inconsistency when using fixed versus dynamic-width representations.

Qin and Van Durme (2023) address these issues, but rely on existing tokenization schemes. They introduce a scoring network which selects "nuggets" from a sequence of contextualized vectors, then pool information into those selections via transformer layers. The selected sequence of nuggets is then used to represent the text moving forward.

Similar approaches can be taken with character-based models (Nawrot et al., 2023; Edman et al., 2022; Godey et al., 2022). For example, Nawrot et al. (2023) modify Hourglass Transformers (Nawrot et al., 2022) with a boundary predictor network for segmenting characters into dynamic-width tokens. This enables jointly training tokenization and token-level language modeling end-to-end. While the training of these models is efficient, the decoding of new text requires repeatedly passing the entire sequence through the model for every new character generated. This contrasts with token-level transformers which produce an entire token worth of characters before the sequence is reprocessed by the model.

We therefore propose a variant of the Hourglass Transformer with dynamic pooling augmented to

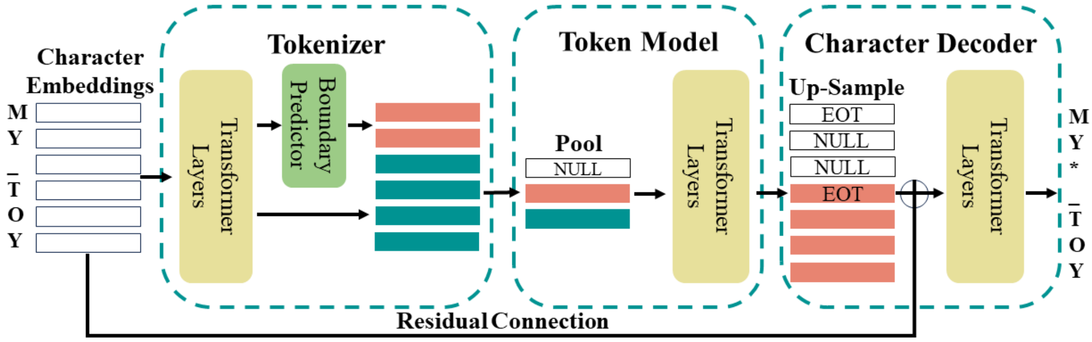

Figure 2: The architecture for Toucan, the token-aware Hourglass Transformer. End-of-token (EOT) vectors and labels (*) are inserted into the character sequences so that the decoder learns token boundaries during training. As per the original model, learned NULL vectors are used to predict the characters in the first token.

become *"token-aware"* in the decoding step. This approach, which we refer to as Toucan, enables decoding entire tokens using a fraction of the compute, without a loss in LM performance. An illustration of the difference is shown in Figure 1.

The contributions of this paper are as follows:

- A technique for modifying character-level language models for more efficient decoding.
- An application of our approach to Nawrot et al. (2023)'s Hourglass Transformer, resulting in over 2x faster character decoding.
- A comparison of popular tokenizers with those learned end-to-end with our models.

2 Background

2.1 The Hourglass Transformer

Nawrot et al. (2022) designed the Hourglass Transformer to address challenges modeling long sequences. Specifically, they introduce contiguous fixed-width pooling at various stages of a typical transformer to shorten the effective length of the sequence being processed. They then up-sample back to the original length with a residual connection from the pre-pooled representation.

2.2 Dynamic Token Pooling

Nawrot et al. (2023) modified the Hourglass Transformer to perform dynamic-width pooling of character sequences. The pooled characters' representation is then processed as a token representation as in traditional transformers. Like Qin and Van Durme (2023), the segmentation of these tokens is selected by a separate feed-forward network. While Nawrot et al. (2023) developed several strategies for training this boundary predictor, our work focuses on their use of the gumbel-sigmoid, which allows end-to-end unsupervised learning of tokenization at a compression rate controlled with a user-defined prior. In keeping with their work, we refer to the achieved compression rate as the shortening factor (SF). Our main contribution is augmenting their architecture to significantly improve its decoding efficiency.

2.3 Tokenizers

We later compare the tokenization learned by our model with two popular alternatives: Byte-Pair-Encoding (BPE) (Sennrich et al., 2016) and WordPiece (Schuster and Nakajima, 2012).

BPE first considers the unique words in a dataset. A set of learned tokens is initialized with the unique characters found among the words. The set is then iteratively expanded to a user-defined size by adding the most frequent combination of an existing token with an additional character.

WordPiece (Schuster and Nakajima, 2012) is a similar tokenization algorithm popularized by its use in training BERT (Devlin et al., 2019). It differs from BPE in that characters that begin a token are treated as separate symbols than their counterparts internal to a token. Instead of frequency, the expansion of the token set is done based on a scoring function that prefers merging tokens that appear more frequently together than they do apart.

3 Token-Aware Decoding

The Toucan architecture is shown in Figure 2. The three components of the architecture are derived from Nawrot et al. (2023) but include changes for improving decoding efficiency. We label the three components of the architecture as the tokenizer, the token model, and the character decoder. First, the tokenizer contextualizes character embeddings and segments the characters into tokens using the boundary predictor. Character representations are pooled to form each token representation. To ensure the model is auto-regressive, the sequence of token vectors are offset using learned null vectors (Nawrot et al., 2023). The token model processes the sequence of token vectors with typical transformer layers. The outputs of the token model are token-contextualized vectors which will be up-sampled and used by the character decoder to predict the characters of the next token.

Decoding a single character x_t from Nawrot et al. (2023)'s original model requires passing the entire sequence $x_{1:t-1}$ through all three model components, regardless of how the preceding characters had been segmented. We leverage the fact that characters segmented into the same token share the same contextualized representation after up-sampling. This representation is reused to predict each character in the next token and therefore provides an opportunity to reduce computations.

To increase decoding speed, we would like the decoder to generate all characters in a token without repeatedly reprocessing the entire sequence with the tokenizer and token model. To this end, we inject a learned end-of-token vector after each token in the up-sampled sequence. The labels for training the decoder are adjusted so that the last character of each token predicts an end-of-token symbol, and the injected end-of-token vector predicts the first character in the next token. We further remove the decoder's dependence on the tokenizer by moving the residual connection from the tokenizer to the embedding layer as in Yu et al. (2023).

A trained Toucan model should be able to generate an entire token using only the embedding layer and character decoder by sampling new characters from the decoder until the end-of-token symbol is predicted. The generated token is then appended to the sequence and processed by the entire model to begin the generation of the next token.

4 Experimental Setup

4.1 Baseline and Evaluation

We use the architecture from Nawrot et al. (2023) as our baseline model and replicate their experiments on the text8 (Mahoney, 2006) and English wiki40b (Guo et al., 2020) datasets.[1] We follow their exact training and evaluation procedures, model size, and hyper-parameters for both the baseline and our models. To evaluate decoding speed, we report wall-clock time while decoding characters on a single NVIDIA Quadro RTX-6000.

[1]Data was gathered and preprocessed using their project repository: https://github.com/PiotrNawrot/dynamic-pooling/

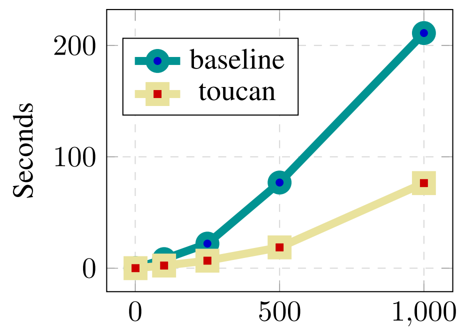

Figure 3: Token generation speed as we increase the number of tokens. Both models trained using a (2,8,2) layer configuration and binomial prior of 0.2.

4.2 Comparing Tokenizers

We compare the tokenization learned by our model with two popular alternatives. First, we compute the number of unique tokens in our training set as reported by our learned tokenizer. We then train BPE and WordPiece models using our unique token count as the vocabulary size.[2] We tokenize our training data with all models and provide tokenization statistics and examples in Section 5.3.

5 Results

5.1 Decoding Speed

As per Nawrot et al. (2023), we trained all models with a (2-8-2) layer configuration for the tokenizer, token model, and character decoder respectively. Toucan only uses the decoder for token completion, while the baseline model requires all layers for each character. The improvement in generation speed while generating an increasing number of tokens is shown in Figure 3. As expected, Toucan is significantly faster as it is using only the last two layers to produce all but the first character per token instead of the entire twelve layers required by the baseline. We verify our modifications have little impact on modeling performance and include language modeling metrics in Appendix A.

[2]Byte-pair-encoding and WordPiece Tokenizers are trained using the Huggingface TOKENIZERS package: https://huggingface.co/docs/tokenizers/index.

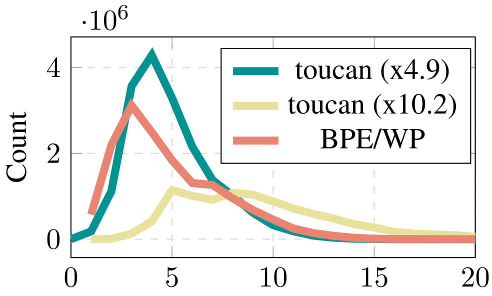

Figure 4: Distribution of token lengths per tokenization algorithm. The plot is cut-off at token length 20, but all algorithms have thin tails extending out past this value.

5.2 Speed Performance Tradeoff

By reducing the binomial prior, the tokenizer is encouraged to increase its compression rate of characters. The trade off between language modeling performance, shortening factor, and generation speed is shown in Table 1. The model generates characters faster with an increased shortening factor, but the language model performance suffers as a result. The models trained at the highest compression rate performed poorly; we omit them from further comparisons.

Binomial Prior	BPC	SF	Gen@1000
0.05	1.652	(x24.4)	1.7s
0.1	1.127	(x10.2)	3.9s
0.2	0.997	(x4.9)	6.1s

Table 1: Bits-per-char, shortening factor, and time to generate 1k characters for varying binomial priors.

5.3 Tokenization

The distribution of token lengths for each model are shown in Figure 4. The Toucan models tend to segment sequences into tokens with lengths close to their shortening factor. We plot a single distribution for BPE and WordPiece which appear nearly identical across all versions. The achieved compression rates for BPE and WordPiece were x5.8 with the smaller vocabulary and x5.9 for the larger. Because the algorithms pre-tokenize on white space, the larger vocabulary captured all unique words in the training set. We include plots for the individual models in Appendix B.

We show the top-10 most frequent tokens per model in Table 2. We observe that the Toucan (x4.9) model has similar top tokens as BPE and WordPiece but prefers segmenting suffixes more frequently than the other models. Toucan (x10.2)'s top tokens are disjoint from the other models as it tends to use those words in longer token phrases. Further comparison in Appendix B.

BPE	WP	Toucan (x4.9)	Toucan (x10.2)
the	the	_the	_that
of	of	_of	_with
and	and	_and	_from
one	one	_one	_it
in	in	_in	_which
a	a	ing	_were
to	to	_a	_but
zero	zero	_to	_also
nine	nine	ion	_eight
two	two	_zero	_seven

Table 2: Top ten tokens per model. Top tokens for BPE/WP remained the same for increased vocabulary. '_': space character for the Toucan models.

The Toucan models tend to tokenize based on spaces, suffixes, word roots, and short phrases when using the higher shortening factor. Unlike BPE and WordPiece, the learned tokenizers can also identify tokens not seen in the training data. In Table 3 we show the tokenization of an example phrase from the test set which includes the unseen word *armalite*. Further examples in Appendix C.

Model	Tokenization
Toucan (x4.9)	ac:qu:is:it:ion: of: the: rifle: from: armal:ite
Toucan (x10.2)	acquisition: of the rifle: from: armalite
BPE	acquisition:of:the:rifle:from:armal:ite
WP	acquisition:of:the:rifle:from:arma:##lite

Table 3: Example tokenization from first entry in the test set. ## : WP marker for a token internal to words.

6 Conclusion

We proposed Toucan, a method for augmenting character-level models to generate learned tokens using a fraction of the compute compared to existing approaches. We applied Toucan to the Hourglass Transformer with dynamic token pooling and demonstrated significant speed ups in character generation without a loss in language modeling performance. We explored the differences between our learned tokenization and popular alternatives such as Byte-Pair-Encoding (BPE) and WordPiece. Our end-to-end tokenizers learn natural tokenization boundaries such as spaces, suffixes, word roots, and short phrases completely unsupervised. In contrast to Byte-Pair-Encoding and WordPiece, our tokenizers are capable of segmenting complete tokens unseen in the training data.

7 Limitations

We have identified three limitations to our work. First, while our approach is language independent, we only evaluate results on English datasets. There are potentially other benefits to this approach in a multilingual setting which should be explored in future work.

Second, we use the architecture of Nawrot et al. (2023) as our only baseline. Our approach can be applied to many character-level models which dynamically pool representations. A broader range of experiments could be extended to architectures used in works such as Edman et al. (2022) or Godey et al. (2022).

Lastly, we do not compare the generation speed of our models with traditional token-based approaches. Our main contribution is improving the efficiency of character-level models, but future analysis comparing performance and efficiency to token-based approaches is warranted.

8 Ethics Statement

We are unaware of any negative impact this work inherently introduces. However, the improved efficiency of our approach has the potential to exacerbate any existing risk from the use of character-level language models by malicious actors.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Lukas Edman, Antonio Toral, and Gertjan van Noord. 2022. Subword-delimited downsampling for better character-level translation. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 981–992, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Nathan Godey, Roman Castagné, Éric de la Clergerie, and Benoît Sagot. 2022. MANTa: Efficient gradient-based tokenization for end-to-end robust language modeling. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2859–2870, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Matt Mahoney. 2006. Large text compression benchmark.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

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, Seattle, United States. Association for Computational Linguistics.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only.

Guanghui Qin and Benjamin Van Durme. 2023. Nugget: Neural agglomerative embeddings of text. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

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, Berlin, Germany. Association for Computational Linguistics.

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.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

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

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. Megabyte: Predicting million-byte sequences with multiscale transformers.

A Language Modeling Performance

	text8			wiki40b		
	BPC	**BPT**	**SF**	**BPC**	**BPT**	**SF**
baseline	1.195	5.840	(x4.9)	1.115	5.533	(x5.0)
toucan	0.997	5.911	(x4.9)	0.957	5.699	(x5.0)

Table 4: Language model performance for the baseline and our Toucan model. Both versions were trained with a binomial prior of 0.2 encouraging a roughly (x5) shortening factor.

Our changes to the Hourglass Transformer were designed to improve decoding efficiency with minimal impact to language modeling performance. Because the Toucan model has an additional character in its vocabulary, the bits-per-character comparison is biased in our favor. Therefore, we also report bits-per-token (BPT). For an average token length $\bar{w}$ we compute bits-per-token (BPT) as

$$bpt = bpc * \bar{w}. \tag{1}$$

This metric favors the baseline, because the same tokenization with Toucan will include additional bits for the end-of-token character. We show performance metrics between the baseline and our architecture in Table 4 and conclude that our modifications have little impact on performance.

B Tokenization Statistics

We plot the distribution of tokens by length for each tokenization algorithm in Figures 5 and 6.

Table 2 highlighted a difference in top tokens for the Toucan (x10.2) model versus BPE and WordPiece. We report the first occurrence of their top tokens for Toucan (x10.2) in Table 5. The Toucan model tokenized these common words into short phrases seen frequently in the dataset.

C Tokenization Examples

We provide several example tokenizations from our test data in Tables 6, 7, 8, and 9. We observe similar tokenizations from BPE and WordPiece while the Toucan (x4.9) model breaks up longer words more frequently. The Toucan (10.2) model tends to group whole words and short phrases as tokens.

Word	First Occurrence	Index
the	the first	60
of	of these	175
and	and other	135
one	one eight	18
in	in one eight	107
a	a number	422
to	to make	351
zero	two zero zero five	124
nine	one nine eight	39
two	two zero zero five	124

Table 5: First occurrence of top BPE/WP tokens in the Toucan (x10.2)'s top tokens.

Toucan (x4.9)
his: career: desp:ite: announc:ing: plans: to: ret:ire
eleven: straight: commerc:ial: disappointments
they: are: temperature: pres:sure: water: vapor
writes: on: the: mod:if:icat:ion: of: clouds
includes: eukaryotes: with: a: nucleus: such: as: fung:i
capac:ity: of: hard: drives: was: measured: in: megabytes
accused: of: irregular:it:ies: in: invest:igat:ing
geolog:ists: to: refer: to: an: extraterrestr:ial: mesa
in: engl:ish: poetry: feet: are: determ:ined: by: emphas:is
mistakes: could: be: corrected: by: apply:ing: correct:ion
pol:ish: parl:iament: in: september: one: nine: nine: seven
nasal: lateral:ity: is: the: release: of: airflow
a: motherboard: also: known: as: a: mainboard: log:ic: board
tombs: insp:ire: the: ant:i: arch:itectural
salt: cellar: of: gold: and: ebony: in: one: five: four: zero
is: called: a: capac:it:ive: manometer: vacuum: gauge
a: relat:ively: late: development: reconstruct:ion
microwaves: at: a: frequency: of: two: four: gigahertz
antony: octav:ian: became: uncontested: ruler: of: rome
morphogenes:is: from: the: greek: morph: shape: and: genes:is

Table 6: Tokenization of phrases from the test data using Toucan (x4.9).

Toucan (x10.2)
his career: despite: announcing: plans: to retire
eleven: straight: commercial: disappointments
they: are temperature: pressure: water: vapor
writes: on the modification: of clouds
includes: eukaryotes: with: a nucleus: such: as fungi
capacity: of hard: drives: was measured: in megabytes
accused: of irregularities: in investigating
geologists: to refer: to an extraterrestrial: mesa
in english: poetry: feet: are determined: by emphasis
mistakes: could: be corrected: by applying: correction
polish: parliament: in september: one nine nine seven
nasal: laterality: is the release: of airflow
a motherboard: also: known: as a mainboard: logic: board
tombs: inspire: the anti: architectural
salt: cellar: of gold: and ebony: in one five four zero
is called: a capacitive: manometer: vacuum: gauge
a relatively: late: development: reconstruction
microwaves: at a frequency: of two four gigahertz
antony: octavian: became: uncontested: ruler: of rome
morphogenesis: from: the greek: morph: shape: and genesis

Table 7: Tokenization of phrases from the test data using Toucan (x10.2).

Byte-Pair-Encoding
his:career:despite:announcing:plans:to:retire
eleven:straight:commercial:disappointments
they:are:temperature:pressure:water:vapor
writes:on:the:modification:of:clouds
includes:eukaryotes:with:a:nucleus:such:as:fungi
capacity:of:hard:drives:was:measured:in:megabytes
accused:of:irregularities:in:investigating
geologists:to:refer:to:an:extraterrestrial:mesa
in:english:poetry:feet:are:determined:by:emphasis
mistakes:could:be:corrected:by:applying:correction
polish:parliament:in:september:one:nine:nine:seven
nasal:later:ality:is:the:release:of:airflow
a:motherboard:also:known:as:a:mainboard:logic:board
tombs:inspire:the:anti:architectural
salt:cellar:of:gold:and:ebony:in:one:five:four:zero
is:called:a:capacitive:man:ometer:vacuum:gauge
a:relatively:late:development:reconstruction
microwaves:at:a:frequency:of:two:four:gigahertz
antony:octavian:became:uncontested:ruler:of:rome
morphogenesis:from:the:greek:morph:shape:and:genesis

Table 8: Tokenization of phrases from the test data using Byte-Pair-Encoding.

WordPiece
his:career:despite:announcing:plans:to:retire
eleven:straight:commercial:disappointments
they:are:temperature:pressure:water:vapor
writes:on:the:modification:of:clouds
includes:eukaryotes:with:a:nucleus:such:as:fungi
capacity:of:hard:drives:was:measured:in:megabytes
accused:of:irregularities:in:investigating
geologists:to:refer:to:an:extraterrestrial:mesa
in:english:poetry:feet:are:determined:by:emphasis
mistakes:could:be:corrected:by:applying:correction
polish:parliament:in:september:one:nine:nine:seven
nasal:lateral:##ity:is:the:release:of:airflow
a:motherboard:also:known:as:a:mainboard:logic:board
tombs:inspire:the:anti:architectural
salt:cellar:of:gold:and:ebony:in:one:five:four:zero
is:called:a:capacitive:mano:##meter:vacuum:gauge
a:relatively:late:development:reconstruction
microwaves:at:a:frequency:of:two:four:gigahertz
antony:octavian:became:uncontested:ruler:of:rome
morphogenesis:from:the:greek:morph:shape:and:genesis

Table 9: Tokenization of phrases from the test data using WordPiece.

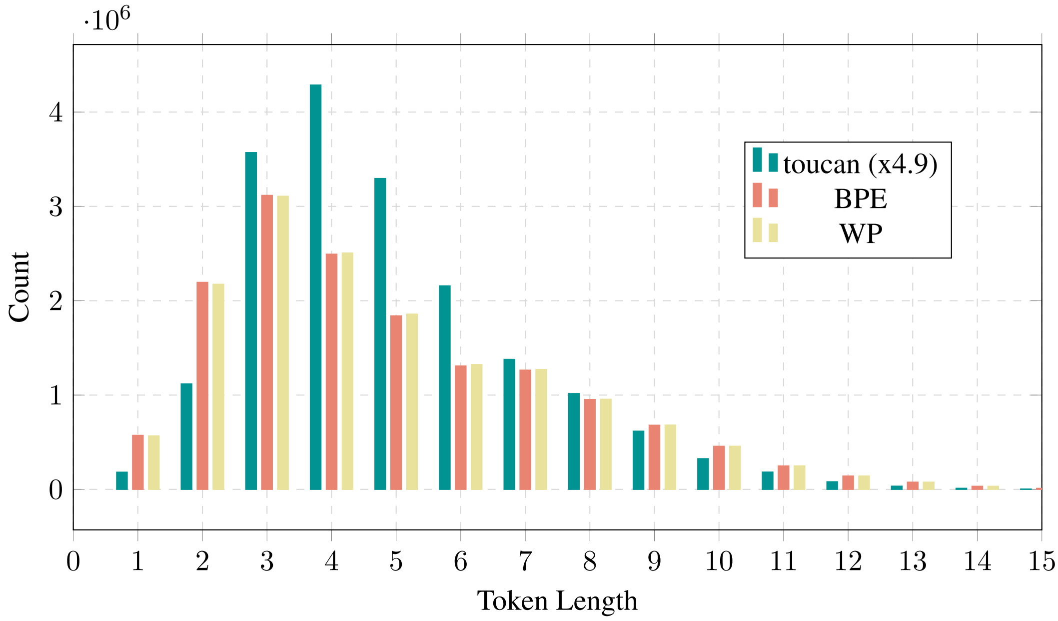

Figure 5: Distribution of token lengths per tokenization algorithm. BPE and WP tokenizers were trained with a vocabulary size of 192,293. The plot is cut-off at token-length 15, but all algorithms have thin tails extending out past this value.

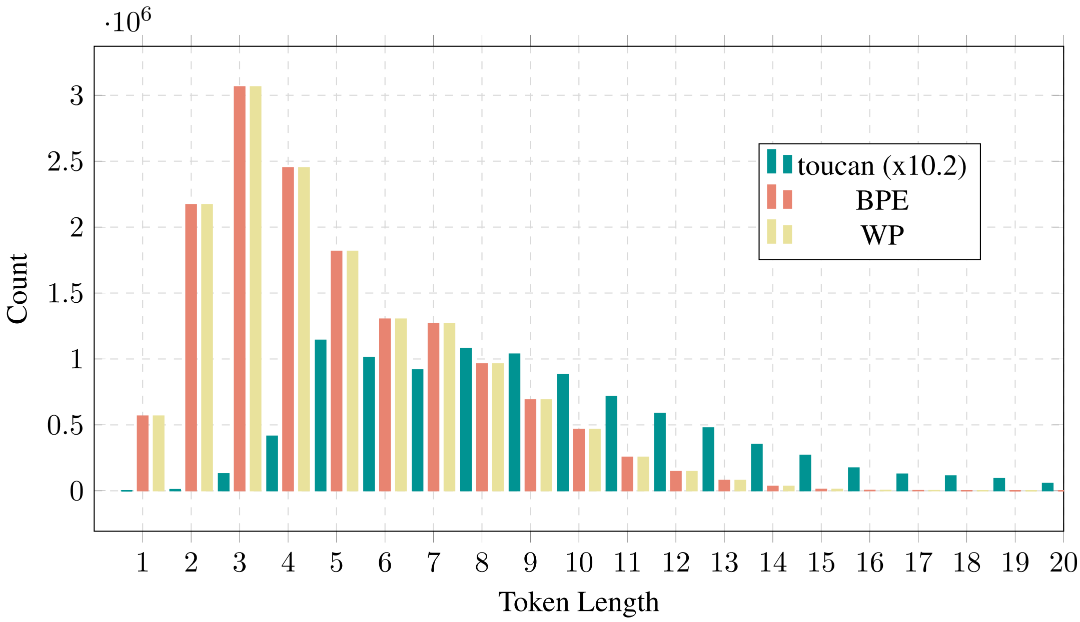

Figure 6: Distribution of token lengths per tokenization algorithm. BPE and WP tokenizers were trained with a vocabulary size of 1,011,543. The plot is cut-off at token-length 20, but all algorithms have thin tails extending out past this value.