MANTa: Efficient Gradient-Based Tokenization for Robust End-to-End Language Modeling

Nathan Godey[*1,2] **Roman Castagné**[*1,2] **Éric de la Clergerie**[1] **Benoît Sagot**[1]
[1]Inria, Paris, France
[2]Sorbonne Université, Paris, France
{nathan.godey,roman.castagne,eric.de_la_clergerie,benoit.sagot}@inria.fr

Abstract

Static subword tokenization algorithms have been an essential component of recent works on language modeling. However, their static nature results in important flaws that degrade the models' downstream performance and robustness. In this work, we propose MANTa, a **M**odule for **A**daptive **N**eural **T**okeniz**A**tion. MANTa is a differentiable tokenizer trained end-to-end with the language model. The resulting system offers a trade-off between the expressiveness of byte-level models and the speed of models trained using subword tokenization. In addition, our tokenizer is highly explainable since it produces an explicit segmentation of sequences into blocks. We evaluate our pre-trained model on several English datasets from different domains as well as on synthetic noise. We find that MANTa improves robustness to character perturbations and out-of-domain data. We then show that MANTa performs comparably to other models on the general-domain GLUE benchmark. Finally, we show that it is considerably faster than strictly byte-level models.

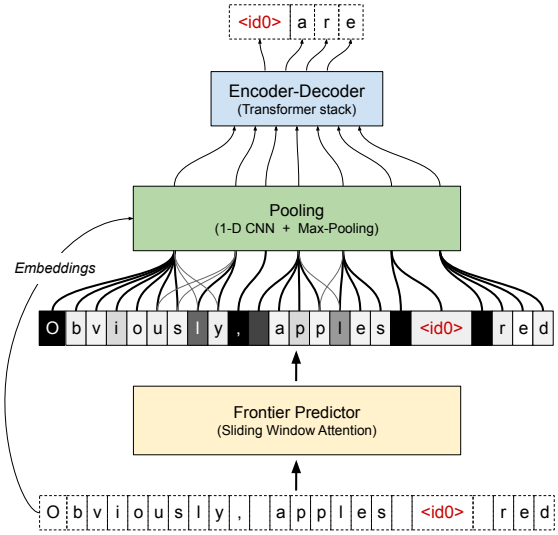


Figure 1: The differentiable tokenization scheme of MANTa-LM. Input bytes are first assigned a *separation probability* using a Sliding Window Attention Transformer. These probabilities are used to compute the contribution of each byte embedding in the pooled representations of the *blocks*. The block embeddings are fed to the Encoder-Decoder layers which predict the masked bytes. All the components are optimized with the LM objective.

1 Introduction

In order to improve Language Models (LMs), the Natural Language Processing field has removed most of the system-induced biases in the last few years. For instance, practices that were once standard such as lemmatization, stemming and feature engineering have progressively disappeared in favor of general architectures trained on huge amounts of data, learning end-to-end which features may be leveraged to attain better performances. However, one essential part of LMs has seen little evolution: tokenization. Tokenizers convert sequences of characters into sequences of tokens (substrings of smaller length) which can then be embedded by the model. Subword tokenization algorithms (Sennrich et al., 2016; Wu et al., 2016; Kudo, 2018) are a specific class of tokenizers designed in such a way that almost every string can be encoded and decoded with very few out-of-vocabulary tokens. They are used in the vast majority of recent LMs, but have been an essential part of NLP systems since much longer (Mielke et al., 2021).

The success of these algorithms can be attributed to several reasons. Firstly, they produce token sequences whose length is greatly reduced compared to the original character sequence. This characteristic is helpful because limitations in compute power and architectural constraints, such as the quadratic complexity with respect to sequence length of Transformers (Vaswani et al., 2017), prevent models from processing arbitrary long sequences. Sec-

[*]Equal contribution.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 2859–2870
December 7-11, 2022

ondly, they compress the corpus using occurrence statistics that may help LMs. For instance, if a word appears frequently in the training corpus, it will be encoded as a single token in the vocabulary and the model will be able to build a representation for that particular token more easily.

However, the induced biases of tokenizers are also harmful for modelization. One such limitation lies in their brittleness to character deformations which are commonly found in real world, noisy data. For instance, BERT's tokenizer (Devlin et al., 2019) encodes "performance" as `["performance"]` but "perfonmance" as `['per', '##fo', '##n', '##man', '##ce']`, which makes it hard for the model to behave similarly in both cases. Moreover, the tokenizer is fixed after its training and is therefore impossible to update, for instance to reflect new domains (El Boukkouri et al., 2020) where tokenization might over-segment specific or technical terms. Clark et al. (2022) list other issues emerging when using static subword tokenizers, especially when modeling languages with a more complex morphology than English.

To overcome these issues, *tokenization-free* models (Clark et al., 2022; Xue et al., 2022; Tay et al., 2021) produce character-based or byte-based embeddings for LMs instead of subword embeddings. These methods improve the robustness of LMs to naturally occurring noise as well as their expressiveness when dealing with out-of-domain or multilingual data. In order to cope with increased input lengths, some of these methods compress sequences with constant reduction rates obtained using specialized modules (Clark et al., 2022; Tay et al., 2021), subsequently removing any notion of subwords.

We argue that learning a subword tokenization together with input representations in an end-to-end fashion is beneficial for language modeling. In this work, we introduce MANTa, a gradient-based tokenizer and embedding module. It can easily be plugged-in to replace the classical combination of fixed tokenizers and trainable subword embedding matrices existing in most encoder-decoder models, without any increase in the total number of trainable parameters. We also introduce MANTa-LM, a Transformer encoder-decoder that incorporates MANTa and that is trained end-to-end. By learning a soft, adaptive segmentation of input sequences jointly with the LM pre-training objective, MANTa-LM produces byte-based representations with sequence lengths similar to those produced by static subword tokenizers. Additionally, by propagating gradients through our soft segmentation module during fine-tuning as well, we are able to adapt the segmentation to new domains, removing the limitations imposed by static subword tokenization.

We show that MANTa-LM is robust to noisy text data and able to adapt to new domains while being significantly faster than byte-level models. Interestingly, MANTa learns a simple but explainable segmentation using only the LM objective while effectively reducing the length of byte sequences.

In summary, the contributions of this paper are the following:

- We introduce MANTa, a gradient-based tokenization and pooling module that can learn jointly with an encoder-decoder LM;
- We train MANTa-LM on English data and we evaluate its robustness to synthetic and natural variation and its ability to adapt to new domains compared to byte-level models.

2 Related Work

Non-neural subword-level tokenization methods have dominated in the last few years as the default way to encode textual data, the most used being BPE (Sennrich et al., 2016), WordPiece (Wu et al., 2016) and Unigram (Kudo, 2018). However, they have inherent flaws that limit their multilingual performance (Rust et al., 2021), their adaptability to new languages and new domains after pre-training (El Boukkouri et al., 2020; Garcia et al., 2021) and the downstream performance of language models in general (Bostrom and Durrett, 2020).

To alleviate these issues, tokenization-free (or character-level) models leverage characters instead of subwords to build text representations. Some of the first neural networks for sequence generation used characters directly as inputs (Sutskever et al., 2011; Graves, 2013), and following works modified the approach to create input word representations based on characters (Kim et al., 2016; Józefowicz et al., 2016; Peters et al., 2018). Similar architectures were recently adapted to work with Transformers (El Boukkouri et al., 2020; Ma et al., 2020). Nevertheless, they still rely on fixed tokenization heuristics (for instance segmenting using whitespaces) which may not be suited to some

languages or certain types of language variations. Recent works have tried to remove these induced biases by working purely with characters or bytes as input (Clark et al., 2022; Tay et al., 2021; Xue et al., 2022). However, they either have to use various tricks to reduce the sequence lengths based on other induced biases like downsampling rates (Clark et al., 2022; Tay et al., 2021) or have extremely low training and inference speeds (Xue et al., 2022). Chung et al. (2016) create tokens in a differentiable manner by predicting frontiers and using the representations of each character inside a "token", but it remains unclear how their model could be adapted to be used with newer architectures such as Transformers. Mofijul Islam et al. (2022) propose to segment tokens using a trained "frontier predictor." Nevertheless, this differentiable tokenizer is not trained with the main language model objective but instead mimics a BPE subword tokenizer, carrying some of its flaws.

3 MANTa

3.1 Differentiable Tokenization

Our main contribution is the introduction of an end-to-end differentiable tokenization architecture that consists in softly aggregating input bytes into what we refer to as *blocks*. As an analogy with hard tokenization schemes, blocks can be compared to tokens with smooth borders.

We decompose the tokenization process into several differentiable operations, ensuring that our model can be trained end-to-end. Our approach consists in predicting a segmentation, and then combining byte embeddings according to this segmentation. MANTa can be divided in three different parts:

- Predicting block frontiers using a parameterized layer to assign a probability p_{F_i} to each input byte b_i of being a frontier;[1]
- Building a byte-block unnormalized joint distribution using the frontier probabilities $(p_{F_i})_{i\in[1,L]}$ corresponding to a soft assignment from bytes to blocks;
- Pooling byte representations for each block B_j weighted by the probability of each byte to belong in the current block $P(b_i \in B_j)$.

This process results in a sequence of embeddings that can be given directly to the encoder-decoder model. We provide an overview of the entire model in Figure 1. We also summarize the process of mapping byte embeddings to block embeddings in appendix D.

3.1.1 Predicting Subword Frontiers

Our frontier predictor consists in a parameterized module mapping each byte b_i to the probability of being a block frontier p_{F_i}. In a first part, we embed each byte b_i to an embedding e_{b_i}. Working with bytes instead of characters allows modeling a larger array of symbols while having very small embedding matrices with $256 \times hidden\ size$ parameters. Since the input sequences fed to the frontier predictor may be particularly long, we use a Transformer with sliding window attention (Beltagy et al., 2020). This layer achieves a linear complexity with respect to sequence length by computing attention using only a local context. This reduced context forces the model to focus on local surface features rather than long-range dependencies which may be hard to model at the byte level.

We make the assumption that long-range dependencies are not relevant for segmentation and that this reduced context window should not harm the quality of the tokenization.

3.1.2 Modeling the Byte-Block Assignment

Once the frontier probabilities $(p_{F_i})_{i\in[1,L]}$ are predicted for the whole sequence, we use them to model an assignment between bytes and block slots. Each byte is given a probability distribution over the available block slots, and the expected block position of a byte in the block sequence increases along the byte sequence (i.e. the next byte is always more likely to be assigned to the next block).

Let us introduce $(B,\ b_i)$, the slot random variables for each byte b_i, describing the position of the block containing b_i in the block sequence. In other words, the event $(B = k,\ b_i)$ describes the fact that the i-th byte belongs in the k-th block. These variables can only take values in $[1, L]$, as there cannot be more blocks than there are bytes. We can model the $(B,\ b_i)$ as a *cumulative sum* of the random variables F_i: the position of the block in which a byte belongs is exactly the number of frontier bytes before this one.

Since $F_i \sim \mathcal{B}(p_{F_i})$, we can model the block index variables B depending on the index of the bytes b using the Poisson Binomial distribution $\mathcal{PB}$ which models the cumulative sum of Bernoulli variables: $(B,\ b_i) \sim \mathcal{PB}\Big((p_{F_k})_{k\leq i}\Big)$. There exists

[1] F in p_{F_i} stands for *Frontier*.

no closed form for this distribution's mass function, but some fast methods have been developed to avoid exploring the whole event tree (Biscarri et al., 2018; Zhang et al., 2017). However, to reduce computational cost, we use a truncated Gaussian kernel G with the same mean and variance to approximate the $(B,\ b_i)$ probability mass function:

$$\forall k \in [1, L_B], P(B = k,\ b_i) \simeq P_{k,i} \triangleq \frac{1}{Z} G_{\mu_i,\sigma_i}(k)$$

where $Z = \sum_{1 \leq k \leq L_B} G_{\mu_i,\sigma_i}(k)$ is a normalization term, and:

$$\begin{cases} L_B = \min\left(L, (\mu_L + 3\sigma_L)\right) \\ \mu_i = \sum_{k=1}^{i} p_{F_i} \\ \sigma_i = \sqrt{\sum_{k=1}^{i} p_{F_i}\left(1 - p_{F_i}\right)} \end{cases} \quad (1)$$

We denote by $P_{k,i}$ the approximation of the probability of membership of the byte i to block k. We display an example of this map at different steps during training in Figure 2. We truncate the block sequences after $(\mu_L + 3\sigma_L)$ since all the probabilities beyond this position are negligible.

3.1.3 Pooling Block Embeddings

At this point in the forward pass, we have estimated the position of the block in which each input byte belongs, along with the block sequence maximum plausible length L_B. In order to provide block embeddings to the LM, we now focus on the contribution of each byte to the block given by the block-byte assignment map. For each block position $k \in [1, L_B]$, this map actually provides an unnormalized contribution $(P_{k,i})_{i \in [1,L]}$ of each byte in this block. We can then use the byte embeddings e_b from the frontier predictor described in Section 3.1.1 and, for the k-th block, build a block embedding where each byte b_i contributes based on its probability of being in this block $P_{k,i}$.

To build e^B_k, the embedding of block B_k in k-th position, we first compute the weighted byte embeddings $\left(P_{k,i} \times e^b_i\right)_{i \in [1,L]} \in \mathbb{R}^H$, with H the hidden size of the byte embeddings. To make the block embeddings aware of the ordering of the bytes (so that *ape* and *pea* can have different representations), we proceed to a depthwise 1-D convolution along the dimension of the bytes after weighting. This convolution also improves the expressiveness of the block embeddings. We discuss our efficient implementation of these operations in Appendix A.

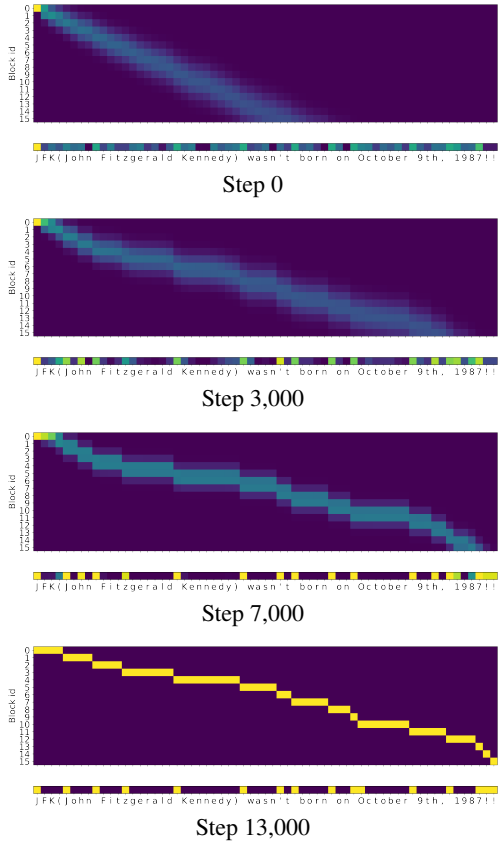


Figure 2: The block-byte assignment P during the first pre-training steps. MANTa learns to downsample input sequences so that no information is lost through truncation, but also converges towards a sharp segmentation.

We finally apply a max-pooling operation on the contextualized weighted byte embeddings for each block. This yields one embedding per block, with the same dimension as the byte embeddings. We use a linear layer to map the block embeddings to the right input dimension for the encoder-decoder model, i.e. its hidden size.

The final step consists in truncating the block embedding sequence to a fixed length $\hat{L} = \min(L_B, L/K)$ with $K \in \mathbb{N}^*$ a fixed *truncation factor*. This simple heuristic ensures that all sequences fed to the encoder-decoder have a length at least K times shorter than the input byte sequence length. We choose $K = 4$ throughout the paper which is in average the number of bytes in an English BPE token. Most importantly, this truncation incentivizes the frontier predictor to produce sufficiently long blocks. We discuss the influence of this mechanism in more depth in Section 6.1.

3.2 Model Training

We obtain from the differentiable tokenizer and pooling module a sequence of block embeddings

that can be used exactly like subword embeddings. Thus, we use an encoder decoder architecture identical to T5 (Raffel et al., 2020). Nevertheless, since we do not have a fixed subword vocabulary, our decoder operates at the byte level similarly to ByT5 (Xue et al., 2022).

3.2.1 Pre-Training Details

Objective Our objective is identical to the one used in ByT5. We mask 15% of bytes randomly and choose a number of spans such that each has an average length of 20 bytes. Each span is then replaced by an `<extra_id_i>` token with `i` identifying the order of the span in the sequence. On the decoder side, the model has to predict in an autoregressive way the span identifier and the masked bytes.

Data We pre-train our model on English text data using C4 (Raffel et al., 2020), a large corpus scraped from the Internet. This corpus is particularly suited to our pre-training due to its diversity in terms of content and linguistic variations. In addition, it enables a better comparison with other tokenizer-free models trained using it such as Charformer. Since this dataset is not available publicly, we use the English split of the mC4 distributed by AllenAI. We filter long documents containing more than 2^{15} bytes, which is a simple proxy to remove important quantities of unwanted code data.

Hyperparameters We pre-train two versions of our model: MANTa-LM$_{Small}$ and MANTa-LM$_{Base}$. Each of them stacks a MANTa$_{Small}$ (resp. MANTa$_{Base}$) tokenizer and embedding module and a T5$_{Small}$ (resp. T5$_{Base}$) encoder-decoder model stripped of its tokenizer and subword embedding matrix. Details about MANTa hyperparameters can be found in Appendix B.

Following T5 and ByT5, we use the Adafactor optimizer with a learning rate of 10^{-2} for the encoder-decoder model, parameter scaling for the whole system and no weight decay. However, to maintain stability of our differentiable tokenizer, we use a learning rate of 10^{-3} for the parameters of the byte embeddings, the frontier predictor, and the pooling module. We also use a triangular learning rate schedule with 1000 (resp. 5000) warm-up steps for batch size 1024 (resp. 64).

Training We train T5$_{Small}$, MANTa-LM$_{Small}$, and MANTa-LM$_{Base}$ for 65k steps with a batch size of 1024. Sequence lengths are respectively 1024 for $Small$ models and 2048 for the $Base$ model. Thus, the models are trained on roughly the same amount of bytes as in Tay et al. (2021), where a batch size of 64 is used for 1M steps.

We also train a ByT5$_{Small}$ model on the same data, using a batch size of 64 and a sequence length of 1024. We consider the "Scaled" architecture which provides the encoder with more layers than the decoder (Xue et al., 2022). To avoid prohibitive computation costs and ensure fairness in terms of available resources between models, we limit its training time to the one of MANTa-LM$_{Small}$. Hence, our ByT5$_{Small}$ is only trained for 200k steps.

4 Experiments and Results

4.1 Evaluation on GLUE

To ensure that our model is competitive with existing language models exploiting subword tokenization algorithms, we evaluate it on several English datasets and compare it with other baseline models.

Setup We use GLUE (Wang et al., 2018), a Natural Language Understanding benchmark consisting of 7 tasks, to evaluate our model. Similarly to T5, we cast the classification tasks as generation tasks where the model has to predict autoregressively the bytes forming the answer.

We compare our model to an encoder-decoder model with subword tokenization (pre-trained with the same denoising objective as T5) and a fully byte-level encoder-decoder, similar to ByT5. We compare $Small$ models with our pre-trained versions, and $Base$ models with results mentioned in Tay et al. (2021). We report the number of parameters given in Tay et al. (2021) for Byte-level T5$_{Base}$, and gather from its low value that their implementation corresponds to a T5$_{Base}$ architecture trained on byte-level inputs.

Results Results can be found on Tables 1 and 2. Overall, MANTa-LM exhibits a performance slightly below Charformer but stays within a small margin on average (1.1 points below). Nonetheless, the main objective of our method is to balance decent performance with robustness and speed which we show in the following sections.

4.2 Robustness to Domain Change

Static subword tokenizers tend to show important limitations when used with texts originating from

Model	$\|\theta\|$	MNLI	QNLI	MRPC	SST-2	QQP	STSB	COLA	AVG
$T5_{Small}$	60M	**79.7/79.7**	**85.7**	80.2/86.2	89.0	**90.2/86.6**	80.0	30.3	76.6
MANTa-LM_{Small} (ours)	57M	79.2/78.6	84.5	**82.3/87.2**	**89.6**	89.9/**86.5**	**81.4**	**32.0**	**77.1**

Table 1: Results on dev sets for the GLUE benchmark for small models following our pre-training procedure.

Model	$\|\theta\|$	MNLI	QNLI	MRPC	SST-2	QQP	STSB	COLA	AVG
$BERT^{\dagger}_{Base}$	110M	**84.4** / -	88.4	86.7/-	**92.7**	-	-	-	-
$T5^{\dagger}_{Base}$	220M	84.2/**84.6**	90.5	**88.9/92.1**	**92.7**	**91.6/88.7**	**88.0**	53.8	84.3
$CharBERT^{\S}_{Base}$	125M	-	**91.7**	87.8/-	-	91/-	-	**59.1**	-
Byte-level $T5^{\dagger}_{Base}$	200M	82.5/82.7	88.7	87.3/91.0	91.6	90.9/87.7	84.3	45.1	81.5
$Charformer^{\dagger}_{Base}$	203M	82.6/82.7	89.0	87.3/91.1	91.6	91.2/88.1	85.3	42.6	81.4
MANTa-LM_{Base} (ours)	200M	77.5/78.8	88.2	82.4/88.2	91.3	90.8/87.7	79.2	51.0	80.3

Table 2: Results on dev sets for the GLUE benchmark. † indicates results obtained by Tay et al. (2021), which are very similar to our models in terms of compute, but use a smaller batch size which may enhance their performance. § indicates results obtained by Ma et al. (2020). The top section concerns model trained using a subword tokenizer.

Model	Accuracy
$BERT^{\ddagger}_{Base}$	77.7
$CharacterBERT^{\ddagger}_{Base}$	77.9
$T5_{Small}$	75.3
MANTa-LM_{Small} (ours)	75.6

Table 3: Results on MedNLI. ‡ indicates results from El Boukkouri et al. (2020), who use a different pre-training corpus than C4. All other results are from models trained with our codebase.

a domain unseen during training. For instance, El Boukkouri et al. (2020) show that tokenizing medical texts with a tokenizer trained on Wikipedia data often results in an over-segmentation of technical terms which in turn affects the downstream performance. By removing this static bottleneck in MANTa-LM, we hope that it should be able to adapt more easily to new domains. To test this hypothesis, we finetune it on a medical Natural Language Inference dataset.

Setup We finetune MANTa-LM on MEDNLI (Romanov and Shivade, 2018), a dataset consisting of 14,049 sentence pairs extracted from clinical notes. We follow the same finetuning setup than for the GLUE Benchmark i.e. use the same batch size and learning rate. We compare our results to the ones obtained by El Boukkouri et al. (2020) with models pretrained on the general domain.

Results We present our results on Table 3. Although we notice a significant drop in performance compared to the encoder models trained by (El Boukkouri et al., 2020), we believe this drop may be due to the different pretraining data used—CharacterBERT uses splits of Wikipedia, which may be helpful to learn some technical terms related to the clinical domain—, and the different model sizes—CharacterBERT uses all of its parameters to encode example, while we keep half of the parameters in the decoder. Nonetheless, we note that MANTa-LM reaches a better performance than its subword tokenization counterpart T5.

4.3 Robustness to Noisy Data

Although LMs may learn complex patterns even from noisy input texts, this ability is conditioned by how the tokenizer segments character sequences. Since MANTa is not static and can be finetuned on non-standard data, we expect it should be able to learn to be more robust to variation/noise compared to a subword tokenizer paired with a LM. To evaluate this hypothesis, we study how MANTa-LM behaves on both naturally occurring text variation and multiple levels of synthetic noise.

4.3.1 Naturally Occurring Noise

Setup Similarly to Tay et al. (2021), we test our model on a toxicity detection task constructed with user generated data. We use the TOXICCOMMENTS dataset (Wulczyn et al., 2017) which contains 223,549 sentences annotated with a binary label indicating whether each sentence can be classified as toxic or not. We also use the same finetuning setup here as the one used for evaluating on the

Model	Accuracy
$T5^{\dagger}_{Base}$	91.5
$Charformer^{\dagger}_{Base}$	92.7
$MANTa\text{-}LM_{Base}$ (ours)	**93.2**

Table 4: Results on the TOXICCOMMENTS dataset. Results indicated by † are from Tay et al. (2021).

Model	$\lvert\theta\rvert$	Seconds/step
Byte-level $T5_{Small}$	57M	9.06 ($\times$ 8.0)
$MANTa\text{-}LM_{Small}$	57M	2.61 ($\times$ 2.3)
$T5_{Small}$	60M	1.13 ($\times$ 1)

Table 5: Comparison of training speeds. All the experiments were run on 16 NVIDIA V100 GPUs using a batch size of 1024 and a sequence length of 1024 bytes or 256 tokens

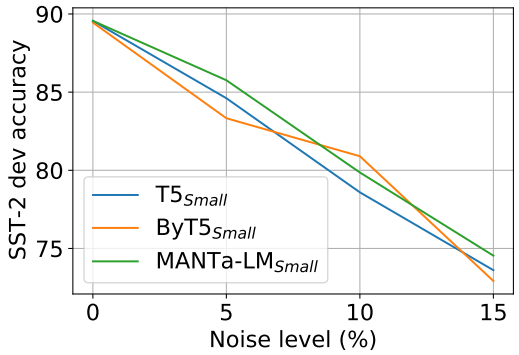
GLUE benchmark.

Results We present our results in Table 4 and compare them to the ones reported in Tay et al. (2021). As expected, noisy user generated data is particularly harmful for models using subword tokenization. On the other hand, constructing sentence representations with byte-level information helps and our model is more accurate than Charformer. This gain may be due to a better segmentation of specific terms encountered in the data.

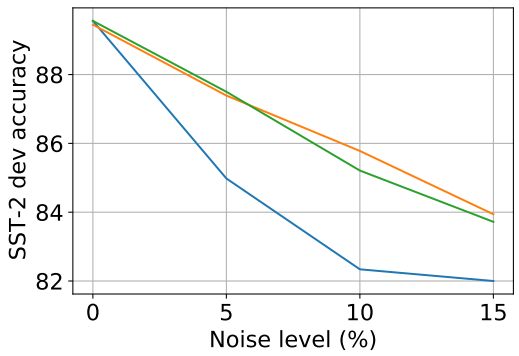
4.3.2 Synthetic Noise

Setup We also compare T5 and ByT5 with our approach when facing different levels of noise. This study pictures how these models react to unseen noise at evaluation time (DEV-ONLY setup) and how they adapt to a given noise via fine-tuning (TRAIN-DEV setup). We apply synthetic noise at different levels $\tau \in \{0.05, 0.10, 0.15\}$ by picking randomly $\tau \times L$ positions in the byte sequences and equiprobably deleting, replacing or inserting bytes at these positions.

Results We found that models performed similarly for the different noise levels in the DEV-ONLY setting. On the contrary, in the TRAIN-DEV setting, MANTa-LM can be finetuned as well as ByT5 for all levels of noise, while the performance of T5 quickly degrades.



(a) DEV-ONLY



(b) TRAIN-DEV

Figure 3: Best accuracy on the SST-2 development set as the noise level increases. The TRAIN-DEV setting corresponds to models finetuned on noisy data while models in the DEV-ONLY setting have been finetuned on clean data.

5 Training Speedups

In terms of speed, we compare our model to $MANTa\text{-}LM_{Small}$ to $T5_{Small}$ counterparts: one that is trained at the classical subword-level, and one trained at byte-level, hence using sequences that are roughly 4 times longer. We also report the speed of the larger $ByT5_{Small}$ architecture as described in Xue et al. (2022).

MANTa-LM is approximately 4 times faster than Byte-level $T5_{Small}$, and 5 times faster than $ByT5_{Small}$, which can be explained by the reduced sequence length we use in the encoder-decoder model. MANTa-LM is only 2.3 times slower than $T5_{Small}$ which furthermore benefits from already tokenized sequences at training time.

6 Discussion

6.1 Truncating Embedding Sequences

Once we obtain block embeddings, the final step in MANTa consists in truncating sequences to a length 4 times smaller than the original byte sequence, as described in Section 3.1.3. This is essential to make MANTa-LM work.

First, it increases the control over the encoder-decoder's computation cost. Without this bottleneck, the Transformer can receive sequences vary-

Original	Oh, it's me vandalising?xD See here. Greetings,
MANTa	Oh|,| it|'s| me| vandalising|?|xD| See| here|.| Greetings|,
T5 tokenizer	Oh|,| it|'|s| me| van|d|al|ising|?|x|D| See| here|.| |Greeting|s|,
Original	The patient was started on Levophed at 0.01mcg/kg/min.
MANTa	The| patient| was| started| on| Levophed| at| 0|.|01|mcg|/|kg|/|min|.
T5 tokenizer	The| patient| was| |started| on| Le|vo|p|he|d| at| 0|.|01|m|c|g|/|kg|/|min|.

Table 6: Examples of segmentations produced by our module (pre-trained only) and by T5's BPE tokenizer. The sentences are samples from TOXICCOMMENTS and MEDNLI.

ing from a single block containing the whole sequence ($L_B = 1$) to one block per byte in the sequence ($L_B = L$). In the latter case, which mimics ByT5's input segmentation, the computation becomes extremely slow due to the quadratic cost of the attention with respect to the sequence length. Using the bottleneck ensures that we can control the worst case complexity of the encoder Transformer and keep it similar to that of a subword-based encoder model.

Second, it serves as a kind of regularization for the block segmentations. We noted that training our module without the bottleneck often led to block sequences as long as byte sequences ($L_B = L$). This may be due to the beginning of training where having very local information helps - for instance bytes to the left and right of masked spans. However, such a segmentation degrades the model speed and performance later in training. Truncating the sequence forces the model to construct larger blocks in order to "fit" all the information from the input sequence.

6.2 Learnt Block Segmentation

Segmentation examples can be found in Table 6. For each byte, we retrieve the expected block position produced by MANTa and approximate it with the closest integer to mimic hard tokenization. We found that MANTa is not keen to produce subword level segmentations. Most of the key symbols for word separation have been identified as block delimiters during pre-training. As expected, MANTa is less prone to over-segmentation of unknown words like named entities. We also found that a trained MANTa produced spiked separation probabilities, meaning that it converged towards a "hard" segmentation. This can also be observed by monitoring the value $\min(p_{F_i}, 1 - p_{F_i})$ which always converges towards values of magnitude 10^{-5}.

6.3 Gradient-Based Segmentation

We employ a radically different downsampling approach compared to other gradient-based tokenization methods such as CANINE (Clark et al., 2022) or Charformer (Tay et al., 2021). While CANINE downsamples sequences using a fixed rate after byte contextualization and Charformer's GBST (Gradient Based Subword Tokenizer) pools representations created using various downsampling rates, MANTa only applies downsampling right before the LM to limit the length of block sequences. Hence, our model is able to build word-level representations of *arbitrary length* as long as it divides the whole byte sequence length by a fixed factor.

We also argue that our method yields more explainable pooled representations as the segmentation can be explicitly derived from the outputs of MANTa. Indeed, contrary to CANINE and Charformer, MANTa disentangles the segmentation of blocks from their representations, allowing to study each part separately.

6.4 Main hyperparameters

We discuss here some of the major hyperparameters of our method. Constrained by limited computational resources, we were unable to assess their exact importance on MANTa's performance. We try to give some intuitions on their influence.

Frontier Predictor We used a small Transformer network with sliding window attention for this module. A much larger network would be slower and may not bring significant improvements to the overall performance of the model, since it is only used for predicting the block byte assignment but does not "expand" the overall expressivity of the model.

Convolution kernel applied on byte embeddings This kernel adds positional information to the byte embeddings and expressivity when constructing the block embeddings. Using a larger kernel or a concatenation of kernels might help for better block representations. However, our experiments did not

show any significant difference in the pretraining performance.

Block embedding sequence truncation factor Trimming block sequences was instrumental to produce meaningful input segmentations and blocks containing more than a single byte. We settled for a factor of 4 since other values led to minor degradations early in training. This factor roughly corresponds to the average number of bytes in a subword created by an English tokenizer.

We believe that a more thorough hyperparameter search could improve the performance of our model. We leave this for future work due to computational limitations.

7 Conclusion

In this work, we present MANTa, a fully differentiable module that learns to segment input byte sequences into blocks of arbitrary lengths, and constructs a robust representation for these blocks. We train this module jointly with an encoder-decoder LM on a span denoising objective to obtain MANTa-LM. We then show that MANTa-LM is more robust when applied to noisy or out-of-domain data than models using static subword tokenizers. At the same time, it performs on par with fully byte-level models on these setups while operating with a much reduced computational cost.

Beyond the noisy and out-of-domain settings, we believe that our approach could lead to interesting results for a number of languages, especially those whose writing system do not use whitespace separators, such as Chinese.

Finally, tokenizers are hypothesized to be an important limiting factor when segmenting multilingual data (Rust et al., 2021). We believe MANTa could be used in the multilingual setting to ensure a more balanced segmentation between languages.

Limitations

Although MANTa can help alleviate some of the inherent issues accompanying subword tokenizers, it also suffers some flaws that we believe could be addressed in future work.

Contrary to encoder-decoder models that can decode long sequences efficiently, our model has to decode sequences byte-per-byte (similarly to Clark et al. (2022); Xue et al. (2022); Tay et al. (2021)) which adds an important computational overhead at generation time. Previous works have attempted to reduce this computational cost by decreasing the size of the decoder layers compared to the encoder (Xue et al., 2022) or by projecting embeddings to a smaller latent space (Jaegle et al., 2021) for the decoding.

Finally, we presented in this work a proof of concept of adaptive segmentation algorithms on relatively small models, ranging from 50M to 200M parameters. Although we hypothesize that our model would scale relatively well since it keeps most of the encoder-decoder architecture untouched, this hypothesis should be tested in a future work.

Acknowledgements

This work was funded by the last authors' chair in the PRAIRIE institute funded by the French national agency ANR as part of the "Investissements d'avenir" programme under the reference ANR-19-P3IA-0001. This work was granted access by GENCI to the HPC resources of IDRIS under the allocation 2022-AD011012676R1.

References

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

William Biscarri, Sihai Dave Zhao, and Robert J. Brunner. 2018. A simple and fast method for computing the poisson binomial distribution function. *Computational Statistics Data Analysis*, 122:92–100.

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, Online. Association for Computational Linguistics.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

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.

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.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun'ichi Tsujii. 2020. CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Xavier Garcia, Noah Constant, Ankur Parikh, and Orhan Firat. 2021. Towards continual learning for multilingual machine translation via vocabulary substitution. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1184–1192, Online. Association for Computational Linguistics.

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

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier J. Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. 2021. Perceiver IO: A general architecture for structured inputs & outputs. *CoRR*, abs/2107.14795.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam M. Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *ArXiv*, abs/1602.02410.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*.

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, Melbourne, Australia. Association for Computational Linguistics.

Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. 2020. CharBERT: Character-aware pre-trained language model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 39–50, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*.

Md Mofijul Islam, Gustavo Aguilar, Pragaash Ponnusamy, Clint Solomon Mathialagan, Chengyuan Ma, and Chenlei Guo. 2022. A vocabulary-free multilingual neural tokenizer for end-to-end task learning. *arXiv e-prints*, pages arXiv–2204.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Alexey Romanov and Chaitanya Shivade. 2018. Lessons from natural language inference in the clinical domain. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1586–1596, Brussels, Belgium. Association for Computational Linguistics.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

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.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *ICML*.

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

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.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Ellery Wulczyn, Nithum Thain, and Lucas Dixon. 2017. Ex machina: Personal attacks seen at scale. In *Proceedings of the 26th international conference on world wide web*, pages 1391–1399.

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.

Man Zhang, Yili Hong, and Narayanaswamy Balakrishnan. 2017. An algorithm for computing the distribution function of the generalized poisson-binomial distribution.

A Improving Pooling Speed

Applying the 1-D convolution requires computing and storing $\mathcal{O}(L_B \times L \times H)$ parameters since we apply the 1D-convolution on every row of the weighted embedding map $P(e^b)^T$. Therefore, this operation may be particularly costly, especially if the frontier predictor outputs a high number of blocks. However, we can use the fact that the weighted embedding map has a special form to reduce the memory load when computing the convolution. Let K be the convolution kernel size, $(C_j)_{j\in[1,K]} \in \mathbb{R}^{K\times H}$ the convolution filters and "$\cdot$" denote the element-wise product. Then, omitting padding and biases :

$$
\begin{aligned}
e^B_k &= \max_{i\in[1,L]} \sum_{j=1}^{K} C_j \cdot \left(P_{k,i+j} \cdot e^b_{i+j}\right) \\
&= \max_{i\in[1,L]} \sum_{j=1}^{K} P_{k,i+j} \cdot \left(C_j \cdot e^b_{i+j}\right)
\end{aligned}
$$

Notice how the product between the convolution filters and the byte embeddings $C_j \cdot e^b_{i+j} \in \mathbb{R}^H$ does not depend on the block anymore. We cache this computation, storing $\mathcal{O}(K \times L \times H)$ parameters and only later apply the convolution per block by summing these products with the block-byte membership map P. Caching greatly lowers the speed and memory requirements of MANTa, allowing to save $L_B - 1$ element-wise products.[2] K is usually small, so the products can be stored easily.

B Hyperparameters

Hyperparameter	MANTa$_{Small}$	MANTa$_{Base}$
Input Embeddings size	64	128
Num. layers	1	2
Num. heads	8	8
Attention window	16	16
Convolution kernel size	3	3

Table 7: Hyperparameters for MANTa

Hyperparameter	ByT5$_{Small}$	T5$_{Small}$	T5$_{Base}$
Hidden size	1472	512	768
Num. layers (encoder)	12	6	12
Num. layers (decoder)	4	6	12
Num. heads	6	8	12
Feed-forward dim.	3584	2048	3072
Dropout rate	0.1	0.1	0.1

Table 8: Hyperparameters for encoder-decoders

[2] This caching would be exactly similar if the convolution was not depthwise.

C Additional results

We include here the scores obtained by MANTa-LM on the GLUE test sets for reproducibility and future comparisons. The development sets are used in the main body to allow a fair comparison, as the test scores are not reported in Charformer (Tay et al., 2021) and CharBERT (Ma et al., 2020).

Model	$\|\theta\|$	MNLI	QNLI	MRPC	SST-2	QQP	STSB	COLA	AVG
MANTa-LM$_{Base}$ (ours)	200M	78.1/78.2	88.6	83.6/88.6	91.0	70.7/88.6	74.1	45.0	78.7

Table 9: Results on test sets for the GLUE benchmark.

D MANTa Module

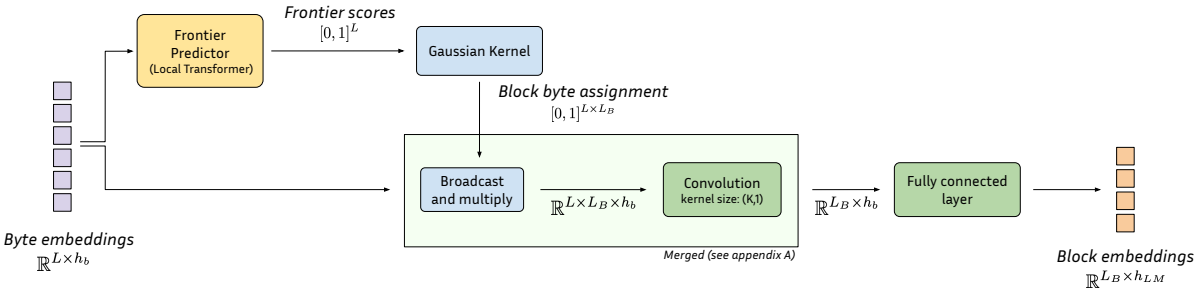


Figure 4: A detailed view of the MANTa module described in section 3.1. We denote by h_b the dimension of the byte embeddings, by h_{LM} the dimension of the block embeddings that will be fed to the encoder-decoder model, L the length of the input sequence and L_B the length of the block sequence. We omit batch sizes for simplicity.