Latent Segment Language Models

Anonymous ACL submission

Abstract

Tokenization is a critical step in every NLP system, yet most works treat it as an isolated component separate from the models they are building. In this paper, we present a framework to jointly learn next-token prediction and segmentation from a sequence of characters or bytes. We evaluate our model on language modeling benchmarks in English, Chinese, and Japanese using both character and byte vocabularies. Our model consistently outperforms baselines on Chinese benchmarks with character vocabulary and shows significant improvements with byte vocabulary. Further latency improvements are achieved by adapting different pooling strategies while maintaining comparable results to the best models. Our main contributions are threefold: we propose a language model that learns to segment the input sequence, conforming to the desired segmentation prior; we demonstrate that our model achieves shorter latency than baselines in token generation; and we show that our model can be applied to three different languages—English, Chinese, and Japanese—demonstrating its potential for wider NLP applications. Our source code will be released on GitHub.

1 Introduction

Tokenization is a vital procedure in every natural language processing (NLP) system because it impacts both computational efficiency and model performance. Tokenization refers to the process of breaking down text into smaller units, such as words or subwords, which can be processed by the model. Before the dominance of neural networks in NLP, word-based tokenization was ubiquitous (Mielke et al., 2021). However, word tokenization has a significant issue: the out-of-vocabulary (OOV) problem, where words encountered during evaluation are not present in the training vocabulary. A common solution to this problem is to map unknown words to a special symbol *<oov>*, which can negatively impact accuracy. To address OOV issues, subword-based tokenization was introduced, segmenting words into sequences of smaller units (Sennrich et al., 2016; Wu et al., 2016; Kudo and Richardson, 2018). While effective in reducing OOV occurrences, this approach introduces a separate learning process for the subword vocabulary. Techniques like Byte Pair Encoding (BPE) are commonly used, but their simple design can lead to suboptimal vocabularies, potentially harming downstream task performance (Bostrom and Durrett, 2020).

Recent research has explored various strategies for discovering linguistic units within text. Segmental language models (Sun and Deng, 2018) model the joint probability of the token sequence and its segmentation, facilitating the discovery of Chinese words from an unlabeled corpus. These models effectively compute the marginal probability of the input sequence by parameterizing the maximum segment length and employing a dynamic programming algorithm for training. Alternatively, some researchers group input characters into a fixed number of segments (Clark et al., 2022; Behjati and Henderson, 2023). These methods train models to aggregate sequences of character representations into shorter abstract representations. However, this approach limits the model to a fixed number of abstract representations during training.

In contrast, Nawrot et al. (2023) proposed improving Transformer efficiency and performance by pooling character representations. Their method overcomes the fixed-length limitation of previous works by dynamically adjusting the character pooling based on a boundary predictor. Through boundary prediction, their model identifies and pools variable-sized segments of characters, guided by either end-to-end learning or supervision from existing tokenizers. Despite these advancements, Nawrot et al.'s (2023) architecture, like most neural language models, still requires the decoder to condition on the entire previous character string. This creates inefficiencies for long character sequences, as characters with minimal impact on future predictions still incur attention-related computational costs. To mitigate these costs, we adapt the decoder to operate on character segments instead of the entire string. This method was initially proposed by Sun and Deng (2018), but their approach relied on an RNN-based encoder and imposed segment length limits. Our work demonstrates improvements in both latency and effectiveness by sampling the next character with an encoder-decoder language model.

In this paper, we propose an encoder-decoder architecture for training auto-regressive language models. First, a causally masked encoder takes the character string as input and outputs boundaries and representations of the input. We then pool these character representations into shortened representations, as done in (Nawrot et al., 2023). The decoder then takes the characters from one pooled segment as input, using all previously pooled representations for cross-attention. This shift allows for more efficient decoding of character strings, focusing the model on the most relevant segments of the input sequence.

Our main contributions are threefold:

- In terms of model training, we formulate a language model capable of learning to segment the input sequence. This model not only learns to segment the sequence but also conforms to the desired segmentation prior.
- Regarding model efficiency, we demonstrate that our model achieves shorter latency than the baseline in token generation.
- For model evaluation, we show that the proposed model can be applied to three different languages - English, Chinese, and Japanese - demonstrating its potential for wider NLP application.

2 Latent Segment Language Models

We denote a sequence of T tokens as $X = x_1 x_2 \dots x_T$, where each token x_t can represent either a character or a UTF8 byte, depending on the selected vocabulary. Let $Z = z_1 z_2 \dots z_T$ represent the boundaries of segments within X. The variable $z_t \in \{0, 1\}$ indicates the presence of a

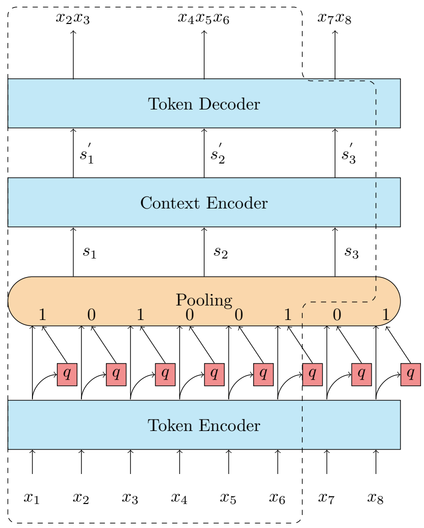

Figure 1: The architecture of the proposed Latent Segment Language Model is illustrated here. x_1 and x_8 represent $< BoS >$ and $< eos >$, respectively. We have not shown $< bos >$ and $< nxt >$ for each output segment in this figure. The segmentation of this sequence is represented as 0s and 1s in the pooling block. The input to all model blocks is causally masked; this means that the i-th output is computed from all the inputs before and including the i-th input. The dashed line highlights the information used to decode the 4th segment, $x_7 x_8$, during inference.

boundary between consecutive tokens x_t and x_{t+1}, as proposed by Nawrot et al. (2023).

The tokens that make up the m-th segment are denoted by $Y_m = y_{m,1} y_{m,2} \dots y_{m,L_m}$, where L_m is the number of tokens in the m-th segment. Here, $z_t = 1$ signifies the end of a segment at position t, and $z_t = 0$ indicates no boundary between x_t and x_{t+1}.

We define the Latent Segment Language Model (LSLM) to jointly model the token sequence X and the segment boundaries Z. The joint probability $p(X, Z)$ is given by:

$$\log p(X, Z) = \sum_{t=1}^{T} \big(\log p(x_t \mid X_{<t}, Z_{<t}) + \log p(z_t \mid X_{\leq t}, Z_{<t}) \big), \quad (1)$$

where $X_{<t} = x_1 x_2 \dots x_{t-1}$ denotes the sequence of tokens generated before the t-th token, and $Z_{<t} = z_1 z_2 \dots z_{t-1}$ denotes the sequence of segment boundaries predicted before the t-th token.

Each boundary prediction z_t is modeled as a discrete latent variable, drawn from a parameterizable distribution:

$$z_t \sim p(z_t; t).$$

The probability distributions $p(x_t \mid X_{<t}, Z_{<t})$ and $p(z_t \mid X_{\leq t}, Z_{<t})$ represent the model's predictions for the next token and the boundary, respectively. The LSLM captures dependencies between token sequences and segment boundaries, providing a more nuanced understanding of the structure within the data.

2.1 Generative model $P(X, Z)$

In the event of a boundary between tokens x_t and x_{t-1}, we aggregate the token representations $h_1 h_2 \ldots h_{t-1}$ into segment representations $S = s_1 s_2 \ldots s_m$. Here, each token is pooled into its corresponding segment. The index m is defined as $m = \sum_{i=1}^{t-1} z_i$, where z_i indicates a boundary. The representation h_t is the contextualized token representation obtained from the token encoder enc_{tok}:

$$h_t = enc_{tok}(h_{t-1}, x_t).$$

Next, we compute the contextualized segment representation s'_m using the context encoder enc_{ctx}:

$$s'_m = enc_{ctx}(s'_{m-1}, s_m).$$

Following this, the model employs an autoregressive decoder to generate the next token x_t of segment $m+1$ using the previous segment representations $s'_{0:m}$:

$$o_{m+1,n} = dec(s'_{0:m}, o_{m+1,n-1}, y_{m+1,n-1}),$$
$$x_t = y_{m+1,n} = softmax(\mathbf{W} o_{m+1,n} + \mathbf{b}).$$

Initially, the previous token of the segment $y_{m+1,n-1}$ is a starting symbol, which provides the initial context for the decoder.

Conversely, when there is no boundary between tokens x_t and x_{t-1}, there is no need for aggregation. In this scenario, the model directly generates the next token x_t of segment m, considering x_{t-1} as the previous token of the segment, denoted by $y_{m,n-1}$. In both cases, whether z_{t-1} indicates a boundary or not, the newly generated token x_t is fed into the token encoder enc_{tok}.

Each segment Y_m is generated in an autoregressive manner, augmented with a *<bos>* symbol at the beginning as $y_{m,0}$ to signify the start of the segment. Furthermore, a *<nxt>* symbol is appended at the end of Y_m to inform the model to transition to the next segment. The augmentation of Y_m allows the model to treat the *<nxt>* symbol as a boundary between y_{m,L_m} and $y_{m+1,1}$ during inference.

2.2 Parametrization of z

To make the boundary context-dependent, the LSLM employs a causally-masked encoder that tokenizes the input sequence incrementally, processing one token at a time. Upon decoding a token, the model appends it to a list and concurrently outputs a boundary prediction to ascertain the formation of a new segment. When a token is identified as an endpoint, the tokens comprising the nascent segment are extracted from the list and fed into the encoder. Since segments consist solely of consecutive tokens, their extraction from the partially formed X and Z is straightforward, negating the need for supplementary data structure.

For the complete computation of sequence probability, LSLM introduces a distinct initial symbol, *<BoS>*, at the beginning of sequence X to establish the context for ensuing segments and to distinguish it from the segment's starting symbol, *<bos>*. Analogously, the symbol *<eos>* is appended to the end of X. In sequence Z, an initial *ending* symbol indicates that *<BoS>* inaugurates a segment. The model is engineered to predict *ending* deterministically when encountering *<eos>* during training. This generation procedure is maintained until the decoder produces an *<eos>* symbol in the inference phase. Throughout this document, T signifies the length of the augmented sequences X and Z.

More specifically, let $Z = \{ending\}$ be the trace of token types of a sequence before any token generation. Each element of Z can either be *ending* or *non-ending*. Without loss of generality, we define z_t as 1 for ending states and 0 for non-ending states. Before generating token x_{t+1}, the boundary prediction z_t is sampled from a Bernoulli distribution based on the current representation $\hat{h_t}$:

$$z_t \sim Bernoulli(p_t), \quad p_t = \sigma(\overrightarrow{1}^\top FFN(h_t)). \tag{2}$$

The FFN consists of two linear transformations without bias terms with a ReLU activation in between (Vaswani et al., 2017), transforming from h_t into $\hat{h_t}$ of the same size. In preliminary experiments, we found the model to be more stable when the FFN's output is summed in this way than transforming it into a scalar.

2.3 Pooling token representations

To compute the segment representations s_m terminating at token x_t, we aggregate the representations of the tokens starting from x_t backwards to include the token at position $t - m$. This aggregation is

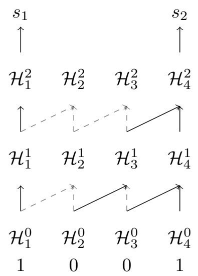

Figure 2: An illustration of 2-hops pooling with a sequence of 4 tokens and its Z shown at the bottom. Each representation, $\mathcal{H}$, is obtained by summing the representations from the source nodes connected by solid lines.

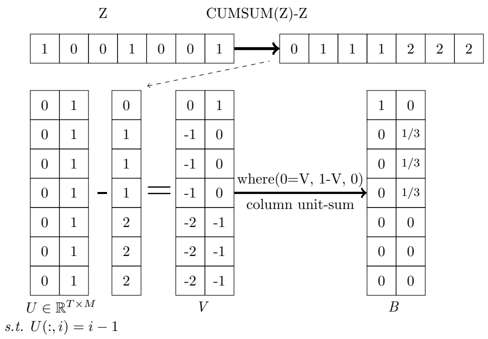

Figure 3: The computation of the binary matrix B for dynamic pooling of token representations is adapted from Bhati et al. (2021). Instead of applying a bounded function, we allow zeros in V to be transformed to $1-V$, and the rest are replaced by zeros, as done by Nawrot et al. (2023). M is the number of segments excluding the segment which contains the <*eos*> symbol.

defined such that $z_{t-j} = 0, \forall j, (j \geq 1 \wedge j \leq m)$, and $z_{t-m-1} = 1$. In this study, we evaluate two distinct pooling methods tailored to different research objectives.

The first method, referred to as N-hops, is formalized as follows:

$$\mathcal{H}_t^n = z_t \mathcal{H}_t^{n-1} + (1 - z_{t-1})\mathcal{H}_{t-1}^{n-1}, \qquad (3)$$

where $\mathcal{H}_t^0 = h_t$. Equation 3 indicates that $\mathcal{H}_t^{n-1}$ is maintained when $z_t = 1$, and $\mathcal{H}_{t-1}^{n-1}$ is merged into $\mathcal{H}_t^n$ if $z_{t-1} = 0$. To derive the segment representations S, we collate all $\mathcal{H}_t^N$ for which $z_t = 1$ and omit the others. With a high value of N, the segment representations $\mathcal{H}_t^N$ encapsulate the full range of token representations within each segment. Conversely, a lower value of N prioritizes capturing only the latter token representations within a segment, yielding quicker but coarser segment representations.

The computation of segment representations that encapsulate the entire set of token representations within each segment may lead to inefficiencies, particularly as a high N incurs computational overhead for segments that necessitate fewer hops. To enhance the efficiency of pooling segment representations, we introduce a method that utilizes a binary matrix $B \in \mathbb{R}^{T\times M}$ derived from Z (Bhati et al., 2021), where M represents the number of segments excluding those containing the <*eos*> symbol. Figure 3 demonstrates the computation of B, resulting in $S = B^\top H$, where $H \in \mathbb{R}^{T\times D}$ encompasses the token representations of the sequence X. This method is termed dynamic pooling (DP) (Nawrot et al., 2023).

Before these segment representations are fed into the context encoder, we normalize them by the number of tokens pooled per segment. Normalization ensures that each segment representation reflects the average contribution of its tokens, rather than being biased by segment length. While the computation of segment representations is executed in a single step during the model's training phase, during inference, these representations are calculated each time the decoder emits the <*nxt*> symbol.

2.4 Optimization

Training the model without supervision of Z requires marginalizing over Z. However, this marginalization becomes computationally infeasible as the sequence length increases. To address this issue, we adopt variational inference (Kingma and Welling, 2014) to approximate the true posterior $p(z|X)$ with a variational posterior distribution $q_\phi(z|X)$, also referred to as the inference model, over segments Z. This approximation involves maximizing the Evidence Lower Bound (ELBO), which serves to minimize the Kullback-Leibler (KL) divergence between $q_\phi(z|X)$ and the true posterior. The ELBO is employed as our objective function:

$$\log p(X) \geq \sum_t \mathbb{E}_{q_\phi(z_t|X_{\leq t})} [\log p_\theta(x_t|X_{<t}, Z_{<t}) \\ -\mathrm{KL}(q_\phi(z_t|X_{\leq t})||p(z))] . \qquad (4)$$

Here, θ and ϕ represent the parameters of the generative model and the inference model, respectively. To improve training efficiency, we utilize a simplified inference model that considers only the prior token history, aligning with the second term of Eq. 1. Consequently, we employ the feed-forward network described in Eq. 2 for the inference model. For $p(z)$, we use Beta(a, b), the conjugate prior of the Bernoulli distribution, to express the variety of segmentation for different languages.

To train the model, we use Gumbel-Sigmoid reparameterization (Geng et al., 2020) to sample from the approximate posterior $q_\phi(z|X_{\le t})$, making the model differentiable:

$$\hat{z}_t = \sigma(\hat{p}_t + g' - g''), \tag{5}$$

where $\hat{p}_t$ is derived from Eq. 2 before sigmoid activation, and g' and g'' are two independent Gumbel noises. Given the discrete nature of boundaries, we discretize the boundary z_t as follows:

$$z_t = \begin{cases} 1, & \text{if } \hat{z}_t \ge 0.5 \\ 0, & \text{if } \hat{z}_t < 0.5. \end{cases} \tag{6}$$

We employ the straight-through estimator (Bengio et al., 2013) to Eq. 5, enabling gradient propagation through Eq. 6 as if it were continuous.

Parameter updates (θ) follow the interleaved optimization strategy of Li et al. (2020). We update the parameters of the generative model for k steps, then update the approximate posterior parameters (ϕ) for a single step. Empirically, we find a few mini-batches are sufficient for the model to adhere to the desired segmentation prior when k is set to 1. For enhanced exploration, we set $k = 3$. A hyperparameter, β, modulates the KL term's influence within the loss function. This approach leverages the interdependence between the generative model and the variational posterior, ensuring that improvements in q_ϕ directly benefit the learning of p_θ, and conversely.

3 Experimental Setup

3.1 Datasets

We evaluate LSLM on three languages representing distinct morphological types: English (fusional), Chinese (isolating), and Japanese (agglutinative). For English, we use the Penn Tree Bank (Marcus et al., 1993) with preprocessing from Mikolov et al. (2011), where only the top 10K words are retained, and all other words are mapped to an <unk> token. We also follow their data split for training, development, and testing. For Chinese, we use the MSR corpus as presented in the Second International Chinese Word Segmentation Bakeoff (Emerson, 2005). We remove all whitespaces from the training set and split the bottom 10% of sentences to create a development set, with the remainder serving as the training set. We use the testing set from MSR without any modifications as our testing set. For Japanese, we use the "Featured Articles" from the Japanese version of Wikipedia, processed by Mori et al. (2019). We retain the splits for training, development, and testing sets as provided by them.

In the English dataset, we compiled a vocabulary from all characters and whitespace in the training and development set. The '<unk>' symbol in the pre-processed corpus was segmented into five tokens. For both Chinese and Japanese, we explored using bytes and characters for vocabulary construction. For the character vocabulary, we included characters appearing at least five times in both training and development sets. Conversely, the byte vocabulary was derived by converting text strings into UTF-8 byte sequences, resulting in a concise byte vocabulary of only 256 tokens. Additionally, we incorporated five special tokens into the vocabulary to mark the beginning and end of the sequence for X and the segment sequence Y_m, along with an <oov> symbol to accommodate tokens beyond the character vocabulary.

3.2 Models

Any model capable of sequential input processing can function as the encoder and decoder within our proposed LSLM framework. This criterion allows us to leverage models with inherent temporal dynamics, without restricting the architecture to a specific type. We have adapted the T5 Transformer (Raffel et al., 2020) as our encoder-decoder due to its robustness in sequence generation tasks. We modified the attention masks in both the encoder and decoder to use causal masking. Detailed model configuration are shown in Appendix A.

To evaluate the effectiveness of our framework in sequence modeling, we compared LSLM with dynamic token pooling (DTP) as proposed by Nawrot et al. (2023). We re-implemented DTP to maintain consistency across variables. For a fair comparison, DTP was configured with the same number of layers and hidden dimension sizes as LSLM. We also compared it to the standard Transformer (GPT-2 (Radford et al., 2019)) without token shortening, using 18 layers and the same hidden dimension sizes as the LSLM model. A dropout rate of 0.1 was applied to the attention and feed-forward layers for all models.

We evaluated the model on the development set at the end of each epoch, saving it if improvement was observed. The best model was then restored for evaluation on the testing set. We noted that the LSLM's loss sometimes exhibited sudden spikes during training, potentially leading the model to sub-optimal convergence. To mitigate this, we monitored LSLM's development loss and restored LSLM to the previous best model if the loss doubled relative to the previous best performance.

	En	Zh(byte)	Ja(byte)
GPT2	1.418	1.785	1.668
DTP			
p=.4	1.416	1.714	1.682
p=.7	1.379	1.722	1.648
LSLM			
DP, p=.4, β=.5	1.506	1.776	1.606
DP, p=.4, β=1	1.555	1.798	1.612
DP, p=.7, β=.5	**1.363**	1.748	1.626
DP, p=.7, β=1	1.390	**1.667***	**1.564***

Table 1: BPC of models trained on English (En), Chinese (Zh), and Japanese (Ja). We denote LSLM that utilize dynamic pooling as 'DP'. Each result represents the average from five different runs. Additionally, p denotes the prior probability that $z \geq 0.5$ in the beta distribution. A statistically significant improvement in BPC compared to the baselines is indicated by an asterisk (*), as determined by a paired Student's t-test ($p < 0.05$).

4 Results and Discussion

Table 1 shows the results for LSLM and baselines on English, Chinese, and Japanese with a byte vocabulary. Each model is evaluated using Bits Per Character (BPC), computed as follows:

$$BPC(X) = -\frac{1}{T}\sum_{t=1}^{T}\log_2 p(x_t),$$

which measures the negative log likelihood of the corpus—the lower, the better. In all languages, the proposed LSLM with Dynamic Pooling (DP) achieves the lowest BPC, outperforming both GPT-2 and DTP. This improvement is particularly noteworthy in Chinese and Japanese, languages characterized by the absence of explicit word boundaries. The reduction in BPC is not only consistent but also statistically significant, highlighting the robustness of LSLM in handling languages with dense character information.

Additionally, we observed negative results when the model is poorly configured. The model performs worse than GPT-2 for English and Chinese when the prior is set to a low value, which is expected since the segments are getting longer.

	LSLM	DTP	GPT2
Zh	**4.677**	4.921	4.837
Ja	3.093	3.119	**3.03**

Table 2: Comparison of three models using character vocabulary: LSLM, DTP, and GPT2, configured as per the best model specifications reported in the previous table. Results are averaged from 5 different runs.

Character vs Byte Vocabulary Next, we conducted experiments using a character vocabulary to assess whether LSLM generalizes across different vocabularies. The results are reported in Table 2. These results suggest that LSLM is capable of generalizing to different languages and vocabularies. A thorough search of hyper-parameters could benefit both LSLM and DTP, particularly for the Japanese model using a character vocabulary.

In terms of relative improvement, using a byte vocabulary grants LSLM consistent improvement across two languages over the DTP baseline, compared to using a character vocabulary. This results in a 4.95% and 0.83% relative improvement for Chinese and Japanese, respectively, with a character vocabulary, as compared to 2.74% and 5.09% relative improvement with a byte vocabulary.

The improvement of the Japanese model is more significant when transitioning from a character to a byte vocabulary. This improvement is attributable to the composition of the Japanese writing system, which consists of Hiragana, Katakana, and Kanji. Hiragana and Katakana together comprise a total of 96 characters. Kanji, characters adapted from Chinese, are more numerous, and some rare Kanji suffer from the OOV issue. Byte vocabulary significantly mitigates the OOV problem, particularly for Japanese, by efficiently encoding rare Kanji, which makes it more advantageous for Japanese than for Chinese. These findings underscore the effectiveness of LSLM when equipped with byte vocabulary, demonstrating not only a capacity for language generalization but also a notable performance advantage over traditional character vocabulary. The improvement in BPC suggests that byte vocabulary could offer a more robust approach for handling diverse linguistic structures.

	En	Ch(byte)	Ja(byte)
Full model	**1.363***	**1.667***	**1.564**
Small encoder	1.442	1.772	1.572
Small decoder	1.415	1.824	1.718
Both small	1.459	1.806	1.658

Table 3: LSLM performance with small token encoder and decoder configurations compared to the full model. Asterisks (*) indicate statistically significant improvements in BPC over all variants, determined by a paired Student's t-test ($p < 0.05$).

Effects on Sizes of Encoder/Decoder Although LSLM pools the token representations into context representations with shorter lengths, the token encoder and decoder still carry out computations proportional to the sequence length. To amortize the cost of operating on character/byte sequences, we can configure the token encoder and decoder with fewer parameters. For a smaller token encoder and decoder configuration, we set the number of layers to 2, the hidden dimension size to 128, and split the attention into 2 heads. In these experiments, residual connections are omitted when the sizes of the hidden dimensions differ. As shown in Table 3, the performance of both English and Chinese models deteriorates when the token encoder or decoder is under-parameterized. In contrast, the small encoder variant for the Japanese model performs on par with the full model.

Observing the performance decline with smaller decoder configurations, we hypothesize this is due to the decoder's diminished capacity to utilize the encoder's contextual information effectively, compounded by the smaller decoder's challenges in modeling long sequences. This aligns with the discussion on negative results previously highlighted.

The reason why performance doesn't degrade as much for the small encoder variant of the Japanese model could be due to the fact that Japanese is an agglutinative language, where words contain multiple morphemes concatenated together, each adding a new layer of meaning. In contrast, for English and Chinese, a more complex inference model is needed to handle their respective linguistic complexities. Specifically, in English, a word can convey different meanings in a sentence, often through changes at the end or beginning of the word. In Chinese, meanings are often indicated through word order or auxiliary words. This suggests that the inference model, which is conditioned on the output of the token encoder, could more easily learn to segment Japanese than English and Chinese, even when it is under-parameterized.

	En	Zh(byte)	Ja(byte)
3hops	1.389	1.701	1.586
1hop	1.395	1.834	1.699
0hop	1.398	1.777	1.634
3hops+ Small encoder	**1.376**	**1.661***	**1.581**

Table 4: LSLM results for two pooling methods. Hyperparameters are configured to be the same as full model. Asterisks (*) indicate statistically significant improvements in BPC over all variants, determined by a paired Student's t-test ($p < 0.05$).

***N*-hops vs DP** While DP pools all token representations without any redundant computation, we hypothesize that some of the token representations can be omitted. To investigate this effect, we experiment with N-hops using three N values (0, 1, 3) and present the results in Table 4. We observe that the performance degrades as N changes from 3 to 1, which is expected since more tokens are excluded from their segment. However, the performance improves as N becomes 0. This improvement can be attributed to the attention mechanism of the token encoder, allowing past tokens to contribute to the representation of the current token. This suggests that for 0-hop, the token pooling is integrated into the attention computation inside the token encoder, avoiding the uncertainty associated with pooling at N>0, where it's unclear when tokens will be included in the pooling or remain unaffected. Unlike the scenarios with N>0, DP and 0-hop do not face this issue of uncertainty, ensuring more stable training. When training with N-hops, we also observe more instances of model collapsing, where the model trivially predicts a boundary between every token. Finally, it is noteworthy that the small encoder variant with 3-hops pooling not only achieves performance comparable to the full model but also improves efficiency in token generation, reducing the latency from 212 ms to 201 ms on a single V100 GPU, a 5.47% improvement.

5 Related Work

Segmentation Models Several notable approaches have emerged in recent years. He et al. (2020) proposed training a machine translation model where the target sentences are segmented using dynamic programming encoding (DPE).

DPE is learned by marginalizing out different segmentations of the target sentence, given the BPE dictionary and source sentence. Similarly, Kawakami et al. (2019) proposed a segmental neural language model (SNLM) where the context is represented as a sequence of characters, and the generation of each segment is either character-by-character from a decoder or a single draw from a lexical memory compiled from n-grams of the training corpus.

In a similar vein, Meyer and Buys (2022) developed a model that can learn subword segmentations on four Nguni languages, comparable to SNLM. Sun and Deng (2018) introduced an approach that marginalizes the segmentation of a sentence with each segment having a fixed maximum length. This model can discover meaningful Chinese words from a character sequence, given the gold segmentation data of the development set. Unlike these previous works, Behjati and Henderson (2023) proposed a variant of slot attention (Locatello et al., 2020) which can learn to cluster characters into morpheme-like slots. Their model is trained to reconstruct the original sequence with a transformer decoder given the slots.

Recent studies have also focused on improving the evaluation and comparison of segmentation models. For example, Ghinassi et al. (2023) highlighted the difficulties in evaluating text segmentation models and the potential biases introduced by commonly used metrics such as Pk. They provided a comprehensive comparison of architectural and sentence encoding strategies, offering a more robust set of baseline results for future developments in linear text segmentation.

Pooling Token Representations Another series of works focuses on pooling token representations into shorter intermediate representations to reduce computations. These works usually target character-based sequences, as the information each token carries is less dense compared to word-based sequences. For example, CANINE (Clark et al., 2022) adapts a convolution layer to reduce the number of sequence positions, then restores the shortened representations back to their original length by duplicating each representation, enabling sequence prediction and tagging tasks. CHARFORMER (Tay et al., 2022) proposes a gradient-based subword tokenization approach where each character representation is a weighted sum of subword representations, obtained by mean pooling over the character embeddings with various stride sizes.

The most recent work closely related to ours was conducted by Nawrot et al. (2023). Similar to our approach, they employ two encoders in their model: one for processing token representations and another for contextualized representations. The primary architectural difference between our model and theirs is that we do not upsample representations to the original length. Moreover, they augment their training process with an auxiliary loss to prevent the trivial solution of predicting each token as a boundary.

6 Conclusion

We proposed a language model capable of segmenting a sequence of tokens and pooling the tokens within each segment to enhance performance in terms of latency and model perplexity. Specifically, our model employs token pooling using either a fine-grained method, DP, or a more coarse-grained but faster method, N-hops. Experiments conducted on language modeling benchmarks in English, Chinese, and Japanese demonstrate the effectiveness of our proposed model in predicting the next token. Furthermore, we evaluated a variant of the model with fewer parameters in the encoder and found that it can achieve model perplexity comparable to the best-performing model when combined with N-hops pooling, additionally offering the benefit of reduced latency between token generations. Our model also shows its effectiveness in handling diverse vocabularies. In experiments with Chinese and Japanese characters as the vocabulary, our model outperforms the DTP baseline.

In summary, our experiments demonstrate the ability of LSLM to segment sequences effectively, resulting in lower perplexity and improved computational efficiency. These findings enhance our understanding of segment language models, underscoring the importance of incorporating a strong inductive bias within the inference model.

Future work could explore segmentation through decision trees, which presents a promising avenue for allowing the model to uncover morphological structures more efficiently and potentially mitigate issues of model collapsing. Additionally, expanding the model to other languages and domains could provide deeper insights into its generalizability and applicability.

7 Limitations

There are several limitations to LSLM that warrant discussion. First, training takes more time and memory than GPT-2 due to the employment of an encoder-decoder architecture. Specifically, the decoder component initiates a new text generation process for each segment, and gradients need to be back-propagated from every segment during training. This results in higher computational costs and memory usage.

Second, LSLM introduces several new hyperparameters, which can be challenging to tune. Poorly configured LSLMs can result in model collapsing, as discussed in the results section. This hyperparameter sensitivity requires extensive experimentation and fine-tuning, which can be resource-intensive. Future work could explore automated hyperparameter optimization techniques to mitigate this issue.

Third, it is unclear whether LSLMs scale well with larger model parameters or data sizes. Recent advancements in language models have demonstrated emergent abilities by scaling both data and parameters significantly. However, our experiments have been conducted only with small-scale data and parameters. We have not yet evaluated the performance of LSLM with large-scale datasets or larger model configurations. Future research should investigate the scalability of LSLM by experimenting with larger datasets and model sizes.

Additionally, we have not considered fine-tuning the LSLMs for downstream tasks, which is an important step in aligning the models with human needs. Fine-tuning could potentially improve the model's performance on specific applications, such as sentiment analysis or machine translation. Evaluating LSLM's performance on various downstream tasks would provide a more comprehensive understanding of its practical utility and effectiveness.

In summary, while LSLM shows promise in improving token segmentation and pooling, addressing these limitations is crucial for advancing its applicability and performance in real-world scenarios. Future research should focus on optimizing the training process, exploring scalability, and fine-tuning the model for specific downstream tasks.

References

Melika Behjati and James Henderson. 2023. Inducing meaningful units from character sequences with dynamic capacity slot attention. *Transactions on Machine Learning Research*.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *Preprint*, arXiv:1308.3432.

Saurabhchand Bhati, Jesús Villalba, Piotr Żelasko, Laureano Moro-Velázquez, and Najim Dehak. 2021. Segmental Contrastive Predictive Coding for Unsupervised Word Segmentation. In *Proc. Interspeech 2021*, pages 366–370.

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.

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.

Thomas Emerson. 2005. The second international Chinese word segmentation bakeoff. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*.

Xinwei Geng, Longyue Wang, Xing Wang, Bing Qin, Ting Liu, and Zhaopeng Tu. 2020. How does selective mechanism improve self-attention networks? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2986–2995, Online. Association for Computational Linguistics.

Iacopo Ghinassi, Lin Wang, Chris Newell, and Matthew Purver. 2023. Lessons learnt from linear text segmentation: a fair comparison of architectural and sentence encoding strategies for successful segmentation. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pages 408–418, Varna, Bulgaria. INCOMA Ltd., Shoumen, Bulgaria.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2020. Dynamic programming encoding for subword segmentation in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3042–3051, Online. Association for Computational Linguistics.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2019. Learning to discover, ground and use words with segmental neural language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6429–6441, Florence, Italy. Association for Computational Linguistics.

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational bayes. In *2nd International*

Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.

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, Brussels, Belgium. Association for Computational Linguistics.

Xian Li, Asa Cooper Stickland, Yuqing Tang, and Xiang Kong. 2020. Deep transformers with latent depth. In *Advances in Neural Information Processing Systems*, volume 33, pages 1736–1746. Curran Associates, Inc.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. 2020. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, pages 11525–11538. Curran Associates, Inc.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Francois Meyer and Jan Buys. 2022. Subword segmental language modelling for nguni languages. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6636–6649, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531.

Shinsuke Mori, Hirotaka Kameko, and Akira Ogawa. 2019. Wikitext-JA: A Japanese WikiText Language Modeling Dataset. `https://nlp.accms.kyoto-u.ac.jp/wikitext-ja`.

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.

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

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.

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.

Zhiqing Sun and Zhi-Hong Deng. 2018. Unsupervised neural word segmentation for Chinese via segmental language modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4915–4920, Brussels, Belgium. 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. 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.

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*.

A Model Hyper-parameters

In all experiments except the ablation study, we employed a 14-layer Transformer encoder. Four layers function as the character encoder, while the remaining 10 layers serve as the context encoder, processing the pooled representations. The decoder is a 4-layer Transformer operating on segmented sequences Y_m. It has access to all previous segment representations $s'_{0:m-1}$ for cross-attention computation. Unless specified otherwise, the hidden dimension of each Transformer layer is 512, and the intermediate feed-forward dimension is 2048. Attention is split into eight heads in the context encoder and four heads in both the character encoder and decoder.

Models were trained for 125,000 steps using the AdamW optimizer with a batch size of 64, a learning rate of 3e-4, 10,000 warm-up updates, and weight decay of 1e-4. Training data was divided into equal-length sequences, disregarding sentence boundaries, with chunk sizes of 150 for English and 256 for Chinese and Japanese.