Latent Segment Language Models: A Tokenization-Free Approach

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 critical step in most natural language processing (NLP) pipelines. It involves subdividing text into smaller units—commonly words or subwords—for model processing. Historically, word-level tokenization was standard (Mielke et al., 2021), yet it introduced the out-of-vocabulary (OOV) issue: any new or rare word at inference time is absent from the training vocabulary. The typical remedy is to replace these words with a special symbol (<*oov*>), which can degrade model accuracy.

To alleviate OOV issues, subword-based tokenization schemes emerged (Sennrich et al., 2016; Wu et al., 2016; Kudo and Richardson, 2018). These methods split words into smaller units, significantly reducing OOV occurrences. However, techniques like Byte Pair Encoding (BPE) still require a separate learning process for the subword vocabulary. Their simple rule-based design can lead to suboptimal vocabularies, sometimes compromising downstream task performance (Bostrom and Durrett, 2020). In addition, relying on a fixed subword vocabulary may not generalize well to diverse languages and longer sequences.

Motivated by these limitations, recent work has investigated methods to discover linguistic units directly from raw text. Segmental language models (Sun and Deng, 2018) jointly model token sequences and their segmentations, effectively learning Chinese word boundaries in an unsupervised manner. Similarly, some researchers compress sequences of characters into a fixed number of abstract representations (Clark et al., 2022; Behjati and Henderson, 2023), aiming to reduce sequence length and computational overhead. Nevertheless, fixing the number of segments a priori constrains the expressiveness of these approaches.

More flexible solutions have emerged that dynamically adjust character pooling based on learned boundaries. For instance, Nawrot et al. (2023) introduced a boundary predictor to pool variable-sized character segments, showing efficiency gains over fixed-length pooling. Despite these advances, their method—and most neural language models—still requires the decoder to attend to the entire character history. This setup can become computationally inefficient for long sequences, where many characters provide limited contribution to future predictions.

In this work, we propose an *encoder-decoder* architecture that processes character segments instead of the full string at every decoding step. Building on the idea of boundary prediction (Sun and Deng, 2018; Nawrot et al., 2023), we employ a causally masked encoder to detect segmentation boundaries and produce compact segment-level representations. The decoder then focuses on the most relevant segments during cross-attention, effectively reducing the length of the context used for prediction. This design improves latency and retains strong modeling performance, as we show by evaluating on English, Chinese, and Japanese corpora.

Our primary contributions are as follows:

- We propose a language model that jointly learns an internal segmentation of the input, eliminating the need for an external tokenization step.
- We validate our model across multiple languages (English, Chinese, and Japanese), showcasing its adaptability and potential for broader NLP applications.

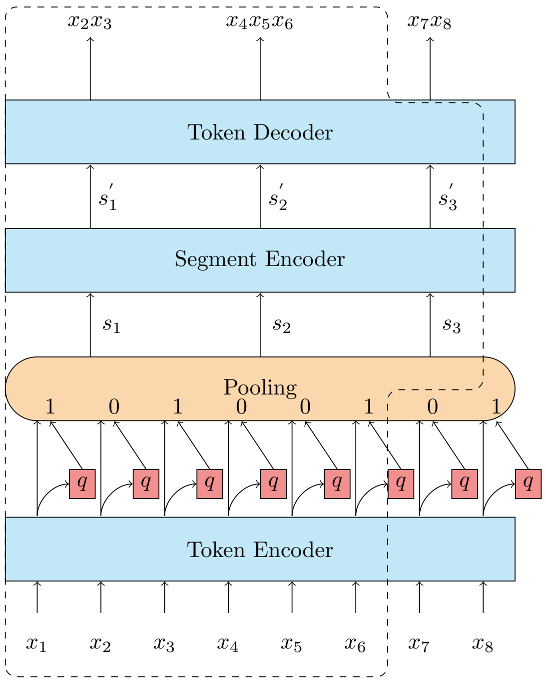

Figure 1: The architecture of the proposed Latent Segment Language Model is illustrated here. x_1 and x_8 represent $< BoS >$ and $< eos >$, respectively. We have not shown $< bos >$ and $< nxt >$ for each output segment in this figure. The segmentation of this sequence is represented as 0s and 1s in 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_7x_8, during inference.

2 Latent Segment Language Models

2.1 Motivation and Overview

Modern NLP often relies on tokenization heuristics—either word-level or subword-level—to process text (Sennrich et al., 2016; Wu et al., 2016; Kudo and Richardson, 2018). However, such approaches can suffer from out-of-vocabulary issues and suboptimal segmentations (Bostrom and Durrett, 2020). To overcome these limitations, we propose the *Latent Segment Language Model* (LSLM), which learns to segment the input sequence on-the-fly. Specifically, the model captures both a token sequence X and a sequence of discrete boundary variables Z, obviating the need for a fixed vocabulary or external tokenizer.

This section first introduces the joint modeling of X and Z, and an illustrative example is provided in Appendix A. We then describe our generative model in detail, explaining how new tokens and boundaries are produced. Finally, we outline how the boundary variable z_t is parameterized and integrated into the model via a causally masked encoder. By the end of this section, we will have established the core design of LSLM before moving on to our pooling strategies (Section 2.5) and training objectives (Section 2.6).

2.2 Model Definition

Let a sequence of T tokens be $X = x_1x_2 \dots x_T$ where each token x_t corresponds to a character or a UTF-8 byte. We define a boundary sequence $Z = z_1z_2 \dots z_T$ where $z_t \in \{0, 1\}$ indicates whether a boundary is introduced between x_t and x_{t+1} (cf. Nawrot et al., 2023). A value $z_t = 1$ signifies that x_t concludes a segment, while $z_t = 0$ indicates continuation. Accordingly, the m-th segment, $Y_m = y_{m,1}y_{m,2} \dots y_{m,L_m}$ consists of consecutive tokens from X, with length L_m determined by the boundaries in Z.

Joint Modeling of X and Z. We define the LSLM to model the probability $p(X, Z)$ jointly, thereby learning where segments should begin or end in a data-driven manner. Formally, we write:

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

where $X_{<t} = x_1x_2 \dots x_{t-1}$ and $Z_{<t} = z_1z_2 \dots z_{t-1}$. Each boundary variable z_t is modeled as a discrete latent variable: $z_t \sim p(z_t; t)$.

Here, $p(x_t \mid X_{<t}, Z_{<t})$ predicts the next token, whereas $p(z_t \mid X_{\leq t}, Z_{<t})$ determines whether the next position introduces a boundary. By jointly learning these distributions, the model captures the intricate interplay between token generation and segment formation.

2.3 Generative Model $P(X, Z)$

Before detailing the boundary parametrization, we first describe how the model generates tokens and segments in an auto-regressive manner. Suppose a boundary is encountered between tokens x_t and x_{t-1}. We then aggregate token representations $h_1, h_2, \ldots, h_{t-1}$ into segment representations $S = s_1 s_2 \ldots s_m$ where $m = \sum_{i=1}^{t-1} z_i$. The token encoder enc_{tok} updates the hidden state for each new token x_t:

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

Next, a segment encoder enc_{seg} produces a contextualized representation s'_m for the entire segment s_m:

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

An auto-regressive decoder then predicts the next token x_t of segment $m+1$, conditioning on the previous segment representations $s'_{0:m}$:

$$o_{m+1,n} = dec(s'_{0:m}, o_{m+1,n-1}, y_{m+1,n-1}),$$

$$x_t = y_{m+1,n} = \mathit{softmax}(\mathbf{W} o_{m+1,n} + \mathbf{b}).$$

Here, $y_{m+1,n-1}$ is either the previously generated token of segment $m+1$ or a special symbol to provide the initial context for the decoder.

Conversely, if $z_t = 0$, the model continues within the current segment m. In both scenarios, the newly generated token x_t is fed back to enc_{tok} to maintain a consistent state.

Segment Markers. Each segment Y_m is generated in an auto-regressive fashion, prepended with a $\langle bos \rangle$ symbol and appended with a $\langle nxt \rangle$ symbol. During inference, the presence of $\langle nxt \rangle$ indicates a boundary transition to the next segment. This mechanism helps the model delineate segments without requiring any external tokenizer.

2.4 Parametrization of z

We now detail how the boundary variables z_t are predicted. The LSLM uses a *causally-masked encoder* that processes one token at a time, appending the new token to a list and simultaneously outputting a boundary prediction. Once a token is flagged as an endpoint, the collected tokens form a complete segment, which is then passed to the segment encoder.

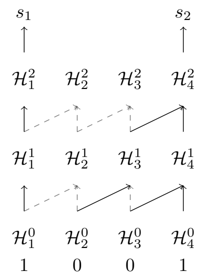

Figure 2: An illustration of 2-hops pooling. Here, a sequence of 4 tokens and its boundary variables Z are shown at the bottom. Each representation $\mathcal{H}$ is obtained by selectively summing source-node representations along the solid arrows.

Boundary Symbols. To unify the start and end of sequences with the segment-based generation, LSLM introduces a unique symbol $\langle BoS \rangle$ at the beginning of X to set the initial context, and an $\langle eos \rangle$ symbol at the end of X. In the boundary sequence Z, an initial *ending* symbol indicates $\langle BoS \rangle$ is already the beginning of the first segment; likewise, we enforce the model to predict an *ending* symbol deterministically upon encountering $\langle eos \rangle$. This procedure simplifies segment creation and terminates decoding once $\langle eos \rangle$ is reached.

Discrete Boundary Variables. Formally, each $z_t \in \{0, 1\}$ is sampled from a Bernoulli distribution:

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

where FFN is a feed-forward network with ReLU activations (Vaswani et al., 2017). In preliminary experiments, we found that summing the output of $FFN(h_t)$ to a scalar before applying the sigmoid produced stable training dynamics.

By predicting z_t jointly with x_t, the LSLM learns an end-to-end mechanism for segmenting and generating text. In the next section (Section 2.5), we will discuss how token representations are aggregated to form segment representations using either N-hops pooling or dynamic pooling. Finally, Section 2.6 will describe how we optimize these discrete boundaries using variational inference and reparameterization techniques.

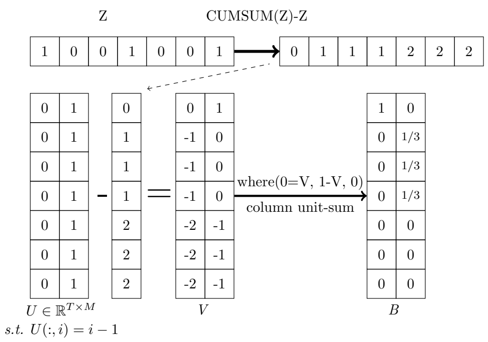

Figure 3: Computing the binary matrix B for dynamic pooling of token representations (adapted from Bhati et al., 2021). Zeros in V are converted to $1 - V$, and non-zero entries become zeros (Nawrot et al., 2023). M denotes the number of segments excluding the one containing $\langle eos \rangle$.

2.5 Pooling Token Representations

After segment boundaries are established (Section 2.3), we need to compute a concise representation for each segment. This step is crucial because the model generates future tokens by conditioning on these segment-level representations rather than solely on individual tokens. Formally, to obtain the representation s_m for a segment terminating at token x_t, we collect the tokens starting from x_t and move backward until reaching the boundary condition $z_{t-m-1} = 1$. As an illustrative example, if a sequence of tokens has been split into two segments by their respective boundaries, we gather the token embeddings in each segment to form distinct segment vectors.

In this work, we investigate two approaches to pooling token representations within each segment. The first, referred to as N-hops, is defined as:

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

where $\mathcal{H}_t^0 = h_t$ denotes the token embedding of x_t. If $z_t = 1$, $\mathcal{H}_t^{n-1}$ remains at the current position; otherwise, we merge $\mathcal{H}_{t-1}^{n-1}$ into $\mathcal{H}_t^n$. By gathering the final $\mathcal{H}_t^N$ for all t such that $z_t = 1$, we obtain a set of segment representations S. A large N aggregates most token embeddings in a segment, offering a richer but more computationally expensive representation; a small N focuses on more recent tokens, providing a lighter approximation.

While N-hops can theoretically capture all token embeddings in a segment, it may be inefficient when many hops are unnecessary. To address this, we adopt a *dynamic pooling* (DP) approach (Nawrot et al., 2023), which constructs a binary matrix $B \in \mathbb{R}^{T\times M}$ from the boundary sequence Z (Bhati et al., 2021), where M is the number of segments excluding any segment containing <*eos*>. As shown in Figure 3, multiplying $B^\top$ by the matrix of token embeddings $H \in \mathbb{R}^{T\times D}$ yields segment representations: $S = B^\top H$. We then normalize each segment vector by its token count to avoid biases from varying segment lengths. Although these representations can be computed in a single step during training, they are recalculated each time a <*nxt*> symbol appears during inference, ensuring consistency between segment boundaries and their corresponding representations.

2.6 Optimization

Having defined how to generate tokens and compute segment representations, we now discuss how to optimize the LSLM when segment boundaries are not explicitly supervised. Directly marginalizing over all boundary configurations Z becomes intractable for long sequences. Therefore, we adopt *variational inference* (Kingma and Welling, 2014) to approximate the true posterior $p(z \mid X)$ with a variational distribution $q_\phi(z \mid X)$. Our objective is to maximize the Evidence Lower Bound (ELBO):

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

where θ denotes the generative model parameters, and ϕ denotes the inference model parameters. We further assume a $\mathrm{Beta}(a, b)$ prior for $p(z)$ to capture a range of segmentation granularities across languages.

Following Eq. 2, the inference model uses a feed-forward network to estimate the boundary probabilities. We employ Gumbel-Sigmoid reparameterization (Geng et al., 2020) to sample from $q_\phi(z \mid X_{\leq t})$ in a differentiable manner. Specifically,

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

where $\hat{p}_t$ is the pre-sigmoid output of the feed-forward network, and g' and g'' are independent Gumbel noises. We then discretize $\hat{z}_t$ into $z_t \in \{0, 1\}$ via:

$$z_t = \begin{cases} 1, & \text{if } \hat{z}_t \geq 0.5, \\ 0, & \text{otherwise.} \end{cases} \qquad (6)$$

A straight-through estimator (Bengio et al., 2013) allows gradients to flow through this discrete step as though it were continuous.

We optimize θ and ϕ using an interleaved strategy (Li et al., 2020): we update the generative model (θ) for k steps, then update the inference model (ϕ) for one step. Empirically, even a small k suffices to align the model with the desired segmentation prior. We incorporate a hyperparameter β to modulate the KL term, balancing the trade-off between fidelity to the training data and adherence to the prior. This design ensures that improvements in one part of the model benefit the other, ultimately facilitating an end-to-end learning of latent segments.

3 Experimental Setup

3.1 Datasets

We evaluate LSLM on three typologically distinct languages: English (fusional), Chinese (isolating), and Japanese (agglutinative).

English. We use the Penn Treebank (PTB) (Marcus et al., 1993) with preprocessing from Mikolov et al. (2011), retaining the top 10k words and mapping the rest to an <unk> token. We follow Mikolov et al. (2011) for the train, development, and test splits.

Chinese. We use the MSR corpus from the Second International Chinese Word Segmentation Bakeoff (Emerson, 2005). Whitespace is removed, and the bottom 10% of sentences form a development set, with the remaining data serving as the training set. The official MSR testing set is used without modification.

Japanese. We adopt the "Featured Articles" corpus from Japanese Wikipedia, processed by Mori et al. (2019), retaining their train/development/test splits.

To handle characters and bytes, we construct:

- **Character vocabulary:** For Chinese and Japanese, we include characters occurring at least five times in both training and development sets. This vocabulary incorporates five special tokens, comprising the four predefined special tokens along with the <oov> token to handle out-of-vocabulary cases.

- **Byte vocabulary:** For all languages, we encode text into UTF-8 bytes, producing a vocabulary of 256 tokens, supplemented with four special tokens.

For English, we also introduce a character-level vocabulary to capture letters, punctuation, and whitespace. Since the preprocessed PTB includes an <unk> symbol, we split <unk> into five tokens, ensuring consistent granularity with our other vocabularies.

GLUE Tasks. Additionally, we evaluate LSLM on English language understanding tasks in GLUE (Wang et al., 2018). Since GLUE's official test set does not provide gold labels, we split the official development set into two halves for validation and testing. We follow Penedo et al. (2024) and use FineWeb-Edu for pre-training, leveraging the same byte vocabulary for robustness across multiple languages.

3.2 Models

We integrate our LSLM framework into two Transformer-based models:

T5-based Encoder-Decoder. We adapt the T5 architecture (Raffel et al., 2020) by applying causal masking in both encoder and decoder self-attention. This maintains autoregressive generation while retaining T5's flexibility for sequence-to-sequence tasks. We compare:

- **LSLM:** Our proposed latent-segmentation approach with dynamic or N-hops pooling.

- **DTP:** The Dynamic Token Pooling baseline by Nawrot et al. (2023), re-implemented to match LSLM's parameter sizes.

- **GPT-2:** A standard autoregressive Transformer (Radford et al., 2019) using 18 layers, without token shortening, but with the same hidden dimensions as LSLM.

All models use a 0.1 dropout on attention and feed-forward layers. We save a checkpoint whenever the development loss improves, and restore the best checkpoint for testing. If LSLM's training loss spikes above twice the previous best, we revert to that earlier checkpoint to prevent divergence.

Mistral Variant for Downstream Tasks. To demonstrate LSLM's adaptability, we also incorporate the Transformer variant by Jiang et al. (2023) for downstream experiments. We add a cross-attention module to the last n layers, treating them as a decoder within the LSLM framework. Pre-training follows the scaling law in Hoffmann et al.

(2022) to optimize FLOPs usage. Details appear in Appendix B.

During fine-tuning on GLUE, we treat each task as a text-to-text problem (Raffel et al., 2020), training the model jointly on multiple tasks to gauge its instruction-following capability. We evaluate on the dev set at each epoch's end, and report test-set performance using the best dev-set epoch. Unless stated otherwise, all models are fine-tuned for 10 epochs.

3.3 Results

Table 1 shows Bits-Per-Character (BPC) on English, Chinese, and Japanese. "DP" denotes LSLM with dynamic pooling; p indicates the mean of the Beta prior. Notably, LSLM with $p = 0.7$ and certain β values achieves improvements over GPT-2 and DTP on Chinese and Japanese, confirming the benefit of learnable segmentation in morphologically diverse languages.

	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 on English (En), Chinese (Zh), and Japanese (Ja). Each entry is the average of five runs. An asterisk (*) indicates a statistically significant improvement over baselines ($p < 0.05$).

4 Results and Discussion

In this section, we present the evaluation of LSLM against baseline models on English, Chinese, and Japanese. We first discuss overall performance using a byte vocabulary (§4.1), then compare character- and byte-level vocabularies (§4.2), and investigate the impact of encoder and decoder size in Appendix (§C).

4.1 Overall Performance

Table 1 reports BPC for each model, where

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

A lower BPC corresponds to better predictive accuracy. Across all three languages, LSLM with DP achieves the lowest BPC, outperforming GPT-2 and DTP. This improvement is especially pronounced in Chinese and Japanese, which lack explicit word boundaries and thus pose greater challenges for token-based models.

Despite these gains, we observe suboptimal results when LSLM is poorly configured. For instance, setting a low prior probability leads to longer segments, negatively impacting English and Chinese performance. This underscores the importance of hyper-parameter tuning, particularly in languages where token length can vary widely.

	LSLM	DTP	GPT-2
Zh	**4.677**	4.921	4.837
Ja	3.093	3.119	**3.03**

Table 2: BPC comparison among LSLM, DTP, and GPT-2 using character vocabularies. Each result is averaged over five runs.

4.2 Character vs. Byte Vocabulary

To assess whether LSLM generalizes across different vocabularies, we also experimented with character-level vocabularies (Table 2). Overall, LSLM shows the ability to handle diverse languages and vocabularies but tends to yield more consistent improvements when using a byte vocabulary. Specifically, we observe a 4.95% and 0.83% relative improvement in Chinese and Japanese, respectively, under character-based settings, whereas the byte-based approach yields 2.74% and 5.09% gains. This discrepancy is especially notable for Japanese, where rare Kanji can cause out-of-vocabulary problems. Byte vocabularies mitigate such issues by encoding each UTF-8 byte directly, offering a more robust representation of languages with large or complex character inventories.

4.3 *N*-hops vs. Dynamic Pooling

Beyond encoder-decoder capacity, we also compare two distinct pooling strategies: N-hops and DP. Unlike DP—which aggregates all token representations—N-hops restricts how many previous representations contribute to each segment. We test three N settings (0, 1, 3), with results shown in Table 3.

Performance declines as N moves from 3 to 1, likely because more tokens are excluded from their segments. Surprisingly, N=0 recovers performance,

	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 3: LSLM results under different pooling methods. Hyperparameters match the full model unless otherwise noted. Asterisks (*) denote statistically significant BPC improvements over all other variants (paired t-test, $p < 0.05$).

suggesting that self-attention in the token encoder already encodes sufficient historical context, alleviating the need for explicit pooling hops. In practice, we find that N-hops can lead to model collapse, where a boundary is predicted after every token, thus degrading training stability. By contrast, both DP and N=0 avoid this uncertainty, providing more consistent performance.

Interestingly, combining N=3 with a small encoder still yields comparable results to the full model while reducing token-generation latency from 212 ms to 201 ms on a single V100 GPU, a 5.47% improvement. This indicates that, with careful hyperparameter tuning, N-hops can strike a balance between performance and efficiency.

4.4 Performance on Downstream Tasks

So far, our evaluations have focused on the language modeling metric BPC across different configurations. We now investigate whether LSLM's latent segmentation also benefits a broader range of NLP tasks. Table 4 summarizes results on the GLUE benchmark, where we measure Matthews correlation for CoLA, Spearman correlation for STS-B, F1 scores for MRPC and QQP, and accuracy for the remaining tasks. An average score across tasks is also reported.

In both Base and Medium models, LSLM achieves a higher overall average than DTP. Notably, LSLM reaches the highest accuracy on QNLI (Base and Medium), SST-2 (Base), STS-B (Base), and MRPC (Medium), implying that jointly learning segmentation and token prediction is especially useful for these classification or matching tasks.

Scaling models from Base to Medium yields improvements for all approaches, though to varying degrees. Meanwhile, LSLM shows only marginal gains in STS-B, and its CoLA performance remains lower than the other models. A targeted fine-tuning on CoLA partially addresses this gap: LSLM improves to 18.63 compared to 21.71 for DTP, indicating that its lower multi-task performance stems partly from domain shift and limited training data. Overall, these results suggest that while LSLM's segmentation approach can enhance certain downstream tasks, careful task-specific or fine-grained tuning may be required to close gaps in low-data settings.

5 Related Work

In this section, we review two main lines of research that are closely related to our work: *Segmentation Models*, which focus on learning subword or morpheme-level boundaries, and *Pooling Token Representations*, which aim to reduce sequence length by aggregating character-level information. We also highlight key differences between these methods and our proposed approach.

5.1 Segmentation Models

Several notable segmentation-based approaches have emerged in recent years. He et al. (2020) introduced a machine translation model where the target sentences are segmented via dynamic programming encoding (DPE). DPE is learned by marginalizing over multiple segmentation hypotheses of the target, given a BPE dictionary and the source sentence. In a similar vein, Kawakami et al. (2019) proposed a segmental neural language model (SNLM) that represents context as a character sequence, generating each segment either character by character or via a single lookup from a lexical memory built from training n-grams.

Further exploring data-driven segmentation, Meyer and Buys (2022) developed a model that learns subword boundaries on Nguni languages, inspired by SNLM-like architectures. Sun and Deng (2018) introduced a different approach that marginalizes segmentation with a fixed maximum segment length, enabling the discovery of meaningful Chinese words from raw characters when minimal gold segmentations are provided. More recently, Behjati and Henderson (2023) proposed a variant of slot attention (Locatello et al., 2020) to cluster characters into morpheme-like slots, training a Transformer decoder to reconstruct the original sequence from these slots.

Aside from architectural innovations, researchers have also examined how to robustly evaluate segmentation. Ghinassi et al. (2023), for exam-

Model	MNLI Acc.	QNLI Acc.	SST-2 Acc.	RTE Acc.	QQP F1	CoLA Mcc.	STS-B Spear.	MRPC F1	Avg.
Base (100M)									
LSLM	**70.48**	**79.01**	**85.77**	**63.50**	**80.53**	1.43	**75.05**	81.04	**67.10**
DTP	69.71	78.13	84.86	59.12	79.09	**3.55**	74.09	**81.25**	66.22
Medium (400M)									
LSLM	73.95	**81.43**	84.63	**66.42**	83.01	**9.18**	75.05	**84.59**	**69.78**
DTP	**75.06**	80.04	**87.15**	65.69	**84.19**	1.34	**78.04**	83.54	69.38

Table 4: Performance of models on GLUE tasks. The numbers in parentheses indicate the approximate number of parameters in each model.

ple, highlighted potential biases in common metrics such as Pk (Beeferman et al., 1999) and compared various model architectures and sentence encodings for linear text segmentation. Their work underscores the challenges in fairly benchmarking segmentation performance, calling for more comprehensive baselines and metrics.

5.2 Pooling Token Representations

Another line of research focuses on reducing sequence length by pooling character-level representations into more compact forms. Since characters often carry less information individually than words, pooling can significantly cut down on computational cost. For instance, CANINE (Clark et al., 2022) uses a convolutional layer to compress character sequences, then upsamples the shorter representations back to the original length to facilitate sequence tagging. CHARFORMER (Tay et al., 2022) adopts a gradient-based subword tokenization, computing each character embedding as a weighted sum over multiple stride-based subword candidates.

Among these pooling-oriented methods, the work by Nawrot et al. (2023) is most similar to ours. They employ two encoders—one operating on raw token representations, and another for contextualization—coupled with an auxiliary loss to avoid trivial segmentation (i.e., predicting a boundary for every token). Unlike their model, we do not restore the sequence to its original length after pooling, focusing instead on a more direct segment-level representation that avoids extra upsampling steps.

Overall, these segmentation and pooling strategies demonstrate the diverse ways in which NLP systems can learn boundaries or reduce sequence length. Our approach draws on elements of both lines of work—dynamic segmentation and token pooling—while introducing an encoder-decoder framework that jointly models boundaries and token generation without re-expanding pooled representations.

6 Conclusion and Future Directions

We have introduced a novel language model, LSLM, that segments an input sequence and pools tokens within each segment to improve both perplexity and latency. In particular, we explore two distinct pooling methods: Dynamic Pooling (DP) for fine-grained representation and N-hops for a faster, coarse-grained strategy. Experiments on English, Chinese, and Japanese language modeling benchmarks show that LSLM effectively predicts future tokens, including scenarios where the encoder is under-parameterized. In such cases, combining a smaller encoder with N-hops achieves performance comparable to our best full model while offering reduced token-generation latency. Furthermore, our model generalizes well to different vocabularies; when tested on Chinese and Japanese characters, LSLM outperforms the DTP baseline.

Overall, these findings demonstrate the viability of joint segmentation and token pooling for efficient sequence modeling. By incorporating a strong inductive bias in the inference model—one that guides segmentation toward meaningful boundaries—LSLM is able to reduce computational overhead without compromising predictive accuracy.

Future Directions. Going forward, we plan to investigate decision-tree-based segmentation, which may more effectively capture morphological structures and mitigate model-collapsing behaviors. We also intend to expand LSLM to additional languages and domains, further evaluating its capacity for generalization and applicability in broader natural language processing tasks.

7 Limitations

Despite the promising results achieved by LSLM, several limitations warrant attention.

Training Cost and Memory Usage. Compared to GPT-2, training LSLM incurs higher computational overhead due to its encoder-decoder architecture. Each segment triggers a new decoding step, requiring gradients to be back-propagated for every segment. This leads to increased time and memory consumption, posing challenges for large-scale or resource-constrained environments.

Hyperparameter Sensitivity. LSLM introduces multiple new hyperparameters (e.g., boundary priors, pooling strategies) that can be difficult to tune. Poor configurations may cause model collapse, as noted in the results section, necessitating extensive experimentation and fine-tuning. Automated hyperparameter optimization techniques (e.g., Bayesian optimization or evolutionary strategies) could potentially mitigate this issue and reduce the manual search burden.

Scalability and Large Models. It remains unclear how well LSLM scales to significantly larger models or massive datasets. While recent language models often exhibit emergent abilities at larger scales, our experiments have been limited to relatively small-scale settings. Evaluating LSLM in conjunction with larger parameter budgets and more extensive corpora would clarify whether its segmentation and pooling mechanisms can maintain effectiveness under substantial growth in model and data size.

Lack of Downstream Fine-tuning. Finally, we have not yet explored applying LSLM to downstream tasks (e.g., sentiment analysis or machine translation). Fine-tuning on specific applications could offer deeper insights into LSLM's practical advantages, particularly regarding how learned segments might enhance domain adaptation or task-specific performance. A thorough evaluation on multiple downstream benchmarks would help ascertain the model's full utility.

Summary. In summary, while LSLM demonstrates promise in improving token segmentation and pooling, these limitations must be addressed to realize its potential in real-world scenarios. Future research should focus on optimizing training efficiency, examining scalability under larger settings, and fine-tuning the model for targeted NLP tasks. Such efforts will help advance LSLM's applicability and performance across a broader spectrum of language processing challenges.

References

Doug Beeferman, Adam L. Berger, and John D. Lafferty. 1999. Statistical models for text segmentation. *Mach. Learn.*, 34(1-3):177–210.

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.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *CoRR*, abs/2203.15556.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

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.

Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *CoRR*, abs/2406.17557.

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.

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, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

A Illustrative Example

For instance, consider a short sequence of four tokens $\{x_1, x_2, x_3, x_4\}$. If the boundary variables are $\{z_1 = 0,\ z_2 = 1,\ z_3 = 0,\ z_4 = 1\}$, then we form two segments:

$$Y_1 = \{x_1, x_2\}, \quad Y_2 = \{x_3, x_4\}.$$

Here, $z_2 = 1$ ends the first segment at position 2, and $z_4 = 1$ closes the second segment at the end of the sequence. Such flexibility allows the model to discover meaningful segmentations without relying on any predefined subword rules.

B 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 segment 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 segment 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.

For the downstream tasks, the Base model consists of 12 layers: two layers as the character encoder, eight layers as the segment encoder, and the final two layers as the character decoder. Each Transformer layer has a hidden dimension of 768 and an intermediate feed-forward dimension of 3072. To ensure optimal utilization of training FLOPs, we follow the scaling law studied by Hoffmann et al. (2022) for pre-training. With this configuration, the model was trained for 19k steps. The Medium model consists of 24 layers: six layers each for the character encoder and decoder, and 12 layers for the segment encoder. Each Transformer layer has a hidden dimension of 1024 and an intermediate feed-forward dimension of 4096. This model was trained for 67k steps. Both models were trained with a batch size of 128 sequences. Each sequence consists of 1024 bytes. The learning rate was set to 3e-4 for the Base model and 2.5e-4 for the Medium model. Both models had 2,000 warm-up steps and a weight decay rate of 0.1.

	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 5: BPC with smaller encoder or decoder vs. the full LSLM. Asterisks (*) indicate significant gains over all smaller variants (Student's t-test, $p < 0.05$).

C Impact of Encoder-Decoder Sizes

While LSLM effectively pools token representations into shorter segments, its token encoder and decoder must still process each token in the sequence. To explore possible efficiency gains, we reduce their parameter sizes and assess the effect on performance. Specifically, we set the number of layers to 2, the hidden dimension to 128, and split the attention into 2 heads. Residual connections are omitted whenever dimension mismatches occur.

Table 5 presents a comparison under this smaller configuration. We observe that both English and Chinese experience performance drops when the encoder or decoder is under-parameterized, suggesting these languages benefit more from larger model capacity. In contrast, the Japanese model with a small encoder remains on par with the full model, indicating that its agglutinative morphology, in which words are composed of multiple morphemes, is easier to segment and model even with fewer parameters.

We hypothesize that the smaller decoder struggles to leverage the encoder's contextual signals, especially in languages requiring extensive morphological or syntactic analysis. This aligns with earlier observations of negative results: a reduced decoder can diminish the model's ability to handle long sequences effectively. For Japanese, each morpheme contributes incremental meaning, enabling an under-parameterized inference module to segment the language with less performance loss compared to English or Chinese.

	Latency (ms)
GPT2	212
DTP	
p=.4	174
p=.7	214
LSLM	
Full model, DP, p=.4	**165**
Small encode, 3hops, p=.7	201
Full model, DP, p=.7	212

Table 6: Latency (in milliseconds) for different models and configurations.