From Characters to Tokens: Dynamic Grouping with Hierarchical BPE

Anonymous ACL submission

Abstract

Subword tokenization methods like Byte Pair Encoding (BPE) are widely used in large language models due to their balance of vocabulary compactness and representational power. However, they suffer from inefficiencies in representing rare words and require large embedding matrices. Character-level models address these issues but introduce performance bottlenecks, particularly in Transformer-based architectures. Recent hierarchical models attempt to merge the benefits of both paradigms by grouping characters into patches, but existing patching strategies either rely on whitespace—limiting applicability to certain languages—or require auxiliary models that introduce new dependencies. In this paper, we propose a dynamic character grouping method that leverages the structure of existing BPE tokenization without requiring additional models. By appending explicit end-of-patch markers to BPE tokens and introducing a second-level BPE compression stage to control patch granularity, our method offers efficient, flexible, and language-agnostic representations. Empirical results demonstrate that our approach matches or exceeds the performance of dynamic entropy- and whitespace-based patching strategies, while maintaining a compact vocabulary.

1 Introduction

Subword tokenization algorithms, particularly Byte Pair Encoding (BPE), have become the de facto standard for text representation in large language models due to their balance between vocabulary compactness and representational flexibility. However, despite their widespread use, subword methods introduce notable limitations. Embedding matrices tied to large vocabularies become parameter inefficient, for which rare words often have bad representations. While BPE provides a degree of compression over raw byte sequences—enhancing computational efficiency, the achievable compression is fundamentally constrained by the vocabulary size. For example, modern tokenizers used in models such as Gemma 2 and LLaMA 3 employ vocabularies of around 250K tokens, which inherently limits the extent to which sequence length can be reduced.

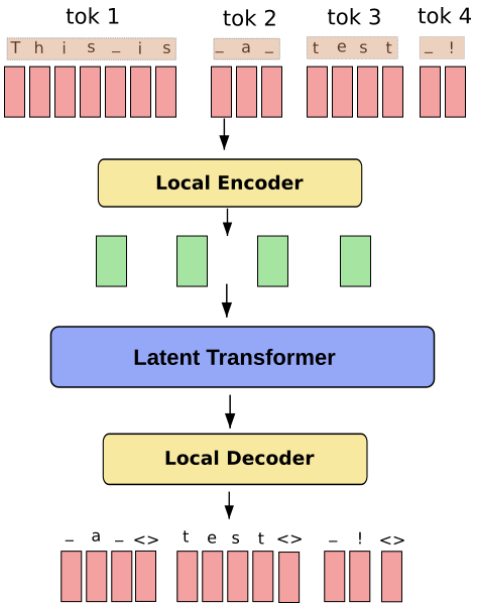

Figure 1: Hierarchical Model: Next patch prediction with autoregressive character prediction per patch. A patch is given by a BPE token."<>" represent end of token.

Character-level models directly address many of the limitations inherent in subword-based tokenization, particularly with regard to rare word handling and parameter efficiency. However, this comes at the cost of runtime performance, especially in quadratic-complexity architectures such as standard Transformers. To bridge this gap, recent work has explored hierarchical representations that aim to combine the flexibility of character-level input with the efficiency of subword models. These approaches typically group characters into

larger units—referred to as "patches"—and learn representations via neural networks rather than embedding tables.

The effectiveness of such models is closely tied to the patching strategy. Naively grouping consecutive characters has been shown to underperform relative to traditional BPE tokenization (Pagnoni et al., 2024; Slagle, 2024). In contrast, dynamic grouping strategies, particularly those that treat whitespace as a delimiter, have achieved competitive or state-of-the-art results. However, whitespace-based segmentation does not generalize to logographic writing systems such as Chinese, where spacing is not semantically meaningful.

One proposed solution involves training a lightweight character-level model to identify patch boundaries based on information-theoretic criteria, such as local entropy. In this setup, new patches are initiated in regions of high entropy (i.e., high surprise). While promising, this approach introduces a dependency on an additional model, which may be sensitive to domain shifts and data variability.

In this paper, we address the limitations of character-level and subword-based models by introducing a dynamic character grouping method that avoids the need for training an additional model to determine patch boundaries. Our approach defines each patch as the sequence of characters that constitute a BPE token, effectively repurposing the tokenization process itself as a grouping mechanism. To enable incremental processing, we modify the standard BPE encoder by appending an explicit end-of-patch marker after each token. While prior work has briefly mentioned this idea, to the best of our knowledge, we are the first to provide a detailed empirical evaluation.

To offset the additional cost introduced by appending an end-of-patch symbol to each patch, we introduce a hierarchical BPE algorithm that compresses each character-level patch to a maximum predefined length S. This compression leverages the observation that short n-grams appear frequently across tokens, allowing for effective merging and shorter patch lengths. Consequently, our method can be viewed as performing *dynamic grouping of tokens*, rather than operating solely at the character level.

Empirically, our model outperforms entropy-based patching and achieves comparable performance with whitespace-based dynamic grouping, while maintaining broader applicability across writing systems. Moreover, compared to standard BPE where the vocabulary is a large, our approach yields better efficiency, requiring significantly fewer FLOPs.

2 Methodology

We introduce a hierarchical representation model that enables more effective trade-offs between granularity, sequence length, and run-time efficiency. Given an input sequence of characters $c_1, c_2, \ldots, c_L$, we apply a pre-trained BPE tokeniser to produce a sequence of variable-length subword tokens $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$. Each token is sequence of characters, which we pass to our hierarchical BPE algorithm. Our algorithm takes the sequence of characters and compresses it to a shorter sequence of integers and adds an end of sequence marker. Then the sequence representing the initial BPE token is padded to the maximum length S[1] and passed to the neural network detailed in section 2.2.

2.1 Hierarchical BPE

While representing BPE tokens as sequences of characters which are modelled by local networks effectively leverages syntactic correlations and handles rare tokens, we observe notable computational inefficiencies. An analysis of the GPT-2 tokenizer (a pre-trained BPE tokenizer, reveals that although the longest token spans 93 bytes, the distribution of token lengths is highly skewed, with the majority containing fewer than 15 bytes (figure 2, left). This skew leads to memory spikes in the local models. Additionally, appending a delimiter (e.g., "<>") to each patch increases patch length, further slowing down the decoding process.

To address these issues, we propose a novel hierarchical tokenization strategy that incorporates a secondary BPE algorithm. This second-stage tokenizer operates on character sequences derived from the initial BPE tokens. Specifically, the algorithm identifies all tokens shorter than a predefined threshold S (the maximum patch length), selects the most frequent byte pair among them, and merges the pair into a new symbol. This procedure is repeated until all tokens conform to the length constraint S. The full algorithm in described in figure 1.

The algorithm can be easily understood with an example. Let us assume a pre-trained BPE tok-

[1]One can reduce the amount of padding by concatenating the patches and applying a sliding window approach. A mask can be used to delimit the sequences.

Algorithm 1 Hierarchical BPE with Fixed Patch Size

Require: Maximum patch size S
Require: Tokens $\mathcal{T} = \{\boldsymbol{x}_v^{1:S} \mid v \in \{1, \ldots, V\}\}$
merges $\leftarrow [\,]$
$V' \leftarrow 0$
while $|\mathcal{T}| > 0$ **do**
  pair $\leftarrow$ MOSTFREQPAIR($\mathcal{T}$)
  $V' \leftarrow V' + 1$
  $\mathcal{P} \leftarrow$ MERGE(pair, merges, $\mathcal{T}$)
  for all $t \in \mathcal{P}$ **do**
    if len(t) $\leq S$ **then**
      Remove t from $\mathcal{T}$
    end if
  end for
end while
return V', merges

enizer splits the text "This is a test!" into four tokens as in figure 1. Also, we set $S = 6$. The algorithm then looks at the entire vocabulary and sees that "is" the most frequent pair[2], therefore merging the pair into a new symbol with encoding "257". The representation of "tok1" becomes $(84, 104, 257, 32, 257, 256)$, where 84, 104 and 32 are the ASCII representations of "T", "h" and space. 256 represents the end of patch marker. Since the length of the representation for "tok1" is smaller than or equal to S, this token will not be further compressed.

Our hierarchical BPE framework can also be viewed as a mechanism for *dynamic grouping* over tokens from the vocabulary V'. Concretely, a sequence of tokens $x_t \in V'$ can be grouped into patches using a secondary, pre-trained BPE tokenizer with a much larger vocabulary $V \gg V'$. This approach enables the model to dynamically form higher-order token groupings, capturing richer structure in the data while maintaining computational efficiency.

2.2 Hierarchical Model

We give an overview of the hierarchical model in figure 1. More precisely, the model can be decomposed into a local encoder g_ϕ, a latent transformer f_θ and a local decoder m_ψ. The encoder f_ϕ independently maps all the sub-sequences to a fixed continuous representation:

$$\boldsymbol{e}_t = g_\phi(\boldsymbol{x}_t^{1:S}) \in \mathbb{R}^D \tag{1}$$

[2]If other pairs are more frequent, they are used first.

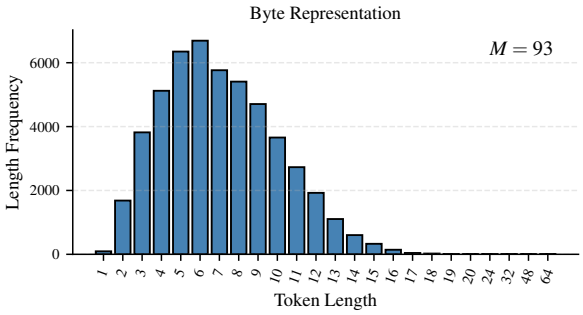

Figure 2: Histogram of lengths for all tokens in the GPT2 tokenizer. The first plot is truncated on the x-axis, from maximum token length $M = 93$.

We subsequently apply a causal *latent transformer* f_θ to the sequence $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_T$, producing the hidden representation:

$$\boldsymbol{h}_t = f_\theta(\boldsymbol{e}_{<t}) \in \mathbb{R}^{D'} \tag{2}$$

Compared to the local encoder and decoder, the latent transformer is substantially larger, dedicating greater computational capacity to modeling the more complex global structure across the sequence of patches. Finally, we pass the hidden representations to a local autoregressive decoder m_ψ, which will predict the sub-sequence representation of the next token:

$$p(x_t^s = c | x_t^{<s}, x_{<t}^{1:S}) = \frac{e^{m_\psi(\boldsymbol{x}_t^{<s}, \boldsymbol{h}_{t-1})[c]}}{\sum_{v \in \mathbb{V}} e^{m_\psi(\boldsymbol{x}_t^{<s}, \boldsymbol{h}_{t-1})[v]}} \tag{3}$$

2.3 Metrics

As our experiments involve both subword and character-level sequence segmentations, it is necessary to normalize evaluation metrics to enable meaningful comparison. To this end, we convert token-level perplexities to bits per byte (BPB), based on the standard definition of information content. Assuming a learned model p_θ and a sequence x of size $|bytes|$, BPB is given by:

$$bpb = \frac{-\log_2 p_\theta(x)}{|bytes|} \tag{4}$$

One can show the above by considering that the total information content $I(x)$ of the test set remains

invariant across segmentation schemes:

$$I(x) = |toks| \times \frac{bits}{token} = |bytes| \times \frac{bits}{byte} \quad (5)$$

$$\Rightarrow \frac{bits}{token} = \frac{|bytes|}{|toks|} \times bpb \quad (6)$$

$$\Rightarrow I(x) = |toks| \frac{|bytes|}{|toks|} \times bpb \Rightarrow bpb = \frac{I(x)}{|bytes|} \quad (7)$$

We can then use the fact that for a data distribution p_D and a model p_θ, the information contained in a sample x is:

$$I(x) = H[p_d, p_\theta] = -\log_2 p_\theta(x) \quad (8)$$

$$\Rightarrow bpb = \frac{-\log_2 P_\theta(x)}{|bytes|} \quad (9)$$

In the above, H refers to entropy, $bpb = \frac{bits}{byte}$, $|toks|$ means the number of tokens in the test set, and $|bytes|$ refers to the number of bytes in the test set.

Finally, we define **token fertility** as $F = \frac{|\text{tokens}|}{|\text{words}|}$, which quantifies the degree of compression applied to the *global sequence* on which the *latent model* f_θ operates. When fertility is less than 1, the latent model processes shorter sequences than the original word-level input, leading to increased computational efficiency.

3 Experiments

3.1 Experimental Setup

Our experiments target both language modeling and downstream tasks such as question answering, using the evaluation benchmark introduced by Gao et al. (2024). For language modeling, we use the first two chunks of the SlimPajama dataset (Soboleva et al., 2023), which notably contain Chinese characters interspersed with English text. We evaluate two model sizes: a smaller 123M-parameter baseline and a larger 359M-parameter model, each trained on token counts ranging from 2.5B to 15B. While model sizes vary, the Latent Transformer architecture remains fixed across all experiments. The only architectural modification involves the embedding layers, which are substituted with local models when applicable. A summary of hyperparameter configurations is provided in Table 1. We also release our code with all the experimental configuration[3]. Our experiments were run on 4 A100, and take between 4 hours to 24 hours to run, depending on the experiment.

[3]Anonymised git url.

Table 1: Model hyperparameters for 359M and 123M parameter LLMs

Hyperparameter	Medium	Small
Latent Layers	24	12
Latent Hidden Dim.	1024	768
Latent FFN	2816	2048
Latent Heads	16	12
Local Hidden Dim	512	512
Local FFN	512	512
Local Heads	8	8
Enc/Dec. Layers	3	3
Learning Rate	6e-4	4e-4

3.2 Baseline Comparison

We begin by comparing our method against four alternative text encoding approaches. The first baseline employs standard Byte Pair Encoding (BPE), implemented using the GPT-2 tokenizer. The second is a character-level model trained on sequences of length 8192. We further evaluate two dynamic patching methods: space-based and entropy-based grouping. For both, we adopt the codebase and experimental setup from Pagnoni et al. (2024), including the same pre-trained entropy model. Notably, these patching baselines incorporate additional parameters due to the use of hash embeddings in their local encoders—components that our model does not include. However, we do not introduce any cross-attention layers, ensuring that the local encoder architecture remains consistent across all experiments.

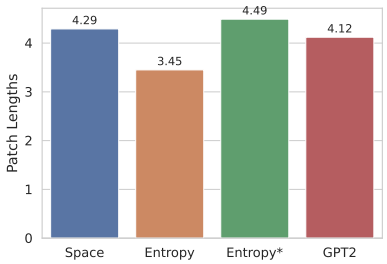

Figure 3: Average Patch Lengths for SlimPajama dataset. "*" represents the unbounded entropy model.

In the space-based setting, we enforce a maximum patch size of 6. For the entropy-based method, we explore two configurations: one with the same maximum patch size of 6, and another with no patch size constraint, allowing the model to determine boundaries solely based on local entropy.

Interestingly, although the maximum patch size is set to 6, the *actual average patch length* on the SlimPajama dataset is only *4.29* for the space-based method. This can be attributed to the prevalence of short words; by enforcing an upper bound, the average is naturally skewed toward shorter sequences.

For the entropy-based method, the average patch length is *3.45* in the bounded setting and *4.49* in the unbounded setting. These relatively short patch lengths may be explained by a domain mismatch, as the entropy model was not trained on the SlimPajama dataset. We consider this comparison fair, as our method, based on dynamic grouping using the GPT-2 tokenizer, is also not specifically adapted to SlimPajama. For reference, the GPT-2 tokenizer produces an average patch length of *4.12* characters per token on this dataset. After introducing the end-of-patch marker and applying our second stage BPE with a maximum patch size of 10, the average patch length becomes *4.13*. Figure3 summarises the average patch lengths for all the grouping strategies. The average patch length has a direct effect on the number of FLOPs, since it affects the length seen by the big latent model f_θ.

FLOPs Calculation. We estimate the total number of FLOPs in a forward pass based on the average patch length, the local encoder/decoder models, and the latent global model:

$$F = T \cdot \text{Tr}(p, D_{\text{enc}}, L_{\text{enc}}, V = 0) \quad (10)$$
$$+ T \cdot \text{Tr}(p, D_{\text{dec}}, L_{\text{dec}}, V = V') \quad (11)$$
$$+ \text{Tr}(T, D_G, L_G, V = 0) \quad (12)$$

In the above, p denotes the average patch length, and T is the number of latent tokens, i.e., $T = Y/p$, where Y is the total input length in bytes. D_x represents the hidden dimension, L_x the number of layers, and V the vocabulary size. The function $\text{Tr}(\cdot)$ refers to the standard FLOPs computation for a Transformer model as defined in Hoffmann et al. (2022), with the note that embedding operations are assumed to have zero FLOPs.

Results. Table 2 shows that our approach outperforms all baselines in this experimental setup. Notably, the entropy-patching method yields the weakest results, likely due to a domain mismatch between the entropy model's training data and the SlimPajama dataset. While the unbounded entropy model is more efficient, the longer patch lengths require the local models to learn more complex representations with limited capacity, leading to a significant drop in performance. Imposing a maximum patch length of 6 improves results, albeit at a higher FLOPs cost, but the performance still lags behind our method.

[4]We enforce the maximum space size to be 6, resulting in a fertility higher than 1.

Model	FLOPs↓	Fertility↓	Params	BPB↓
Entropy-Patch*	509	1.41	351M	1.24
Entropy-Patch	668	1.83	351M	1.20
Space-Patch	534	1.45[4]	351M	1.14
Char - Level	4214	4.5	85M	1.16
BPE	562	1.51	123M	1.16
BPE-Patch	554	1.51	99.7M	**1.11**

Table 2: Comparison between our model for S = 10, BPE tokenisation, Space and Entropy patching for 128M models trained on 13500 steps. FLOPs are provided in billions. "*" represents the unbounded model. We report mean over multiple runs on the test dataset. Maximum standard deviation across experiments: ± 0.007.

The space-based patching model is the most efficient among the baselines and performs reasonably well. Our model achieves the best overall performance, with a higher computational cost. However, it offers a crucial advantage: unlike space-based methods, it generalizes to languages that do not use whitespace as a word separator.

The character-level model uses fewer parameters but suffers from poor runtime efficiency and lower predictive performance. Interestingly, our approach also surpasses standard BPE, despite using fewer parameters. This suggests that the local encoders in our architecture are capable of learning richer token-level representations than those provided by a fixed embedding matrix.

3.3 Increasing the training time.

We also examine whether the performance gains of our model persist throughout training. To this end, we compare our approach against character-level modeling as well as space- and entropy-based patching, across different numbers of training steps. We use the same model configurations as in Table 2. As illustrated in Figure 4, the BPE-patching method consistently outperforms all baselines as training progresses. Notably, the structure-level representation appears to improve over time, indicating that, in some cases, employing a more granular model—despite its higher computational cost—may lead to better long-term performance.

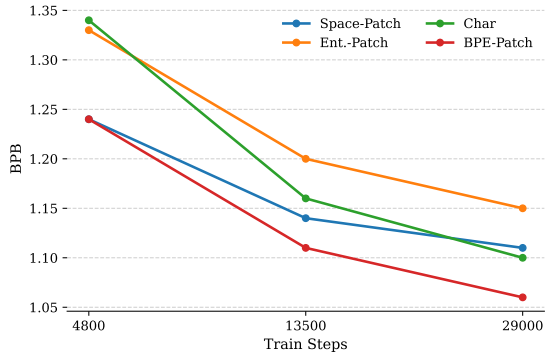

Figure 4: Test performance for different amounts of training.

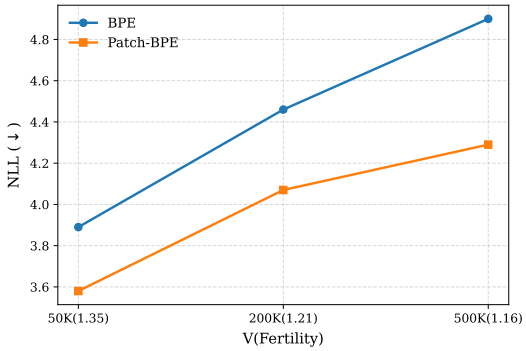

Figure 5: NLL on the test data for varying vocabulary sizes.

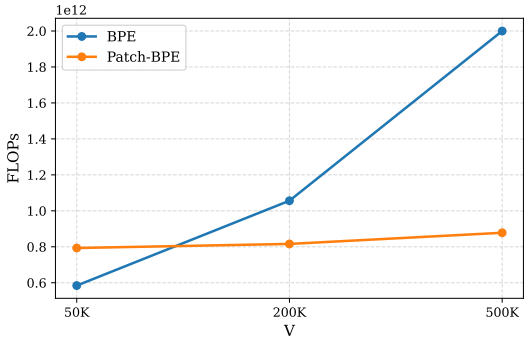

Figure 6: Flops of a forward pass.

3.4 Evaluation on Chinese language

We evaluate our method on the Skypile dataset (Wei et al., 2023), a large-scale Chinese corpus. Small models are trained on a next-byte prediction task for 4,800 steps, and the results are presented in Table 3. While space-based splitting yields a relatively high patch size, this is due to the use of additional heuristics beyond simple space separation. In contrast, the low patch size observed with the entropy-based tokenizer can be attributed to its neural grouping model, which was not trained on this corpus. Our method achieves the largest patch size—resulting in the lowest FLOPs—while also demonstrating strong performance on this dataset.

Model	**Patch Size↑**	**BPB↓**
Space-Patch	3.06	1.24
Entropy-Patch	1.66	1.27
BPE-Patch (LLaMA3)	3.62	**1.20**

Table 3: Comparison of different patching strategies on the Skypile dataset. *LLaMA3* refers to the LLaMA3 tokenizer used in the first-stage BPE. *Patch size* indicates the average number of bytes per patch under each grouping method.

3.5 Increasing the vocabulary

As previously discussed, increasing the vocabulary size V improves compression of the sequence input to the global latent model f_θ. Shorter sequences lead to faster runtime, benefiting both the quadratic attention and MLP layers. However, this introduces a tradeoff: as V increases, the local models must encode more information using fewer tokens, which can strain their capacity. To examine this effect, we conduct an ablation study on vocabulary size. Specifically, we train three SentencePiece tokenizers with vocabulary sizes of 50K, 200K, and 500K, and compare the performance of the standard BPE approach with our model in each setting.

Figure 5 shows that the overall performance, measured by the Negative Log Likelihood (NLL), decreases as vocabulary size increases. However, BPE-patching consistently outperforms the standard BPE approach across all vocabulary sizes. Moreover, our approach demonstrates better scalability with increasing vocabulary size. This behaviour is expected, as the local encoder and decoder in our model generalize more effectively to rare words. In contrast, as the vocabulary size grows, the number of rare tokens increases, leaving many entries in the embedding matrix untrained or unused.

Furthermore, we compare the floating point operations (FLOPs) required by our model to those of the standard BPE approach. The key architectural difference lies in the use of local encoder and decoder modules in our model. To compute the FLOPs for a full forward pass across all layers, we follow the methodology described in Section3.2.

Figure 6 illustrates that our approach scales more efficiently in terms of FLOPs as the vocabulary size increases. This advantage arises from the large

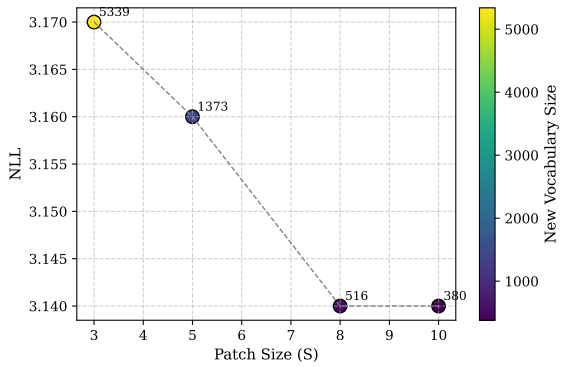

Figure 7: Ablation on the patch size (S) for a small hierarchical model.

embedding matrix in the standard BPE approach, which incurs significant computational overhead during logits computation. In contrast, our method avoids this bottleneck by replacing the embedding matrix with local encoder-decoder modules.

3.6 Ablation on S

The patch size S does not affect the length of the sequence processed by the latent transformer f_w; that is, global sequence compression is determined solely by the BPE tokenizer. However, S has a direct impact on the speed and memory requirements of the local transformer. In general, shorter patches are preferable—particularly during inference, which is performed autoregressively—due to reduced computational load. The tradeoff is that smaller patches limit the ability of the local transformer to model higher-level linguistic structures. Being able to provide this trade-off is part of our contrinution.

We experiment with the small model, and only vary S in algorithm 1. We note that for patches smaller than S we just use padding. The hyperparameters are the same as used in Table1. Figure 7 shows that the optimal maximum patch size in this setting is 8. For lower values, we see a modest performance drop, while for $S = 10$, the NLL stays the same.

3.7 Downstream Tasks

To further verify our model we conduct experiments on question answering tasks. First, we pretrain a medium model on the next token prediction task, for 15B tokens. For the other baselines, we use the same hyperarameters as defined in Table1, alongside a maximum patch size $S = 10$. Then we perform zero-shot evaluations on:

- **Commonsense Reasoning (0-shot)**: HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2019), WinoGrande (Sakaguchi et al., 2019), and ARC-e/-c (Clark et al., 2018).
- **Broad Context Understanding**: Lambada-OpenAI(Paperno et al., 2016).
- **Popular Aggregated Results**: MMLU (Hendrycks et al., 2021).

To obtain the scores we simply calculate the log-likelihood per character in each patch and select the answer with the highest log-likelihood. Table 4 shows that we outperform the standard BPE approach and the entropy based patching, while performing comparable to space patching in most benchmarks.

We also evaluate the language modelling capacity of the medium model by measuring the BPB score. The results in Table 4 show that our model improves with scale (better BPB score than the small model) and also outperforms the other baselines at larger sizes.

4 Related Work

Model-based dynamic grouping. Numerous prior works have explored hierarchical models, particularly those involving dynamic grouping of characters into patches. The Byte-Level Transformer (BLT) (Pagnoni et al., 2024) employs a lightweight model trained on next-character prediction to determine patch boundaries. A new patch is initiated either when the entropy is high or when the current patch exceeds a predefined maximum length. This approach eliminates the need for an explicit end-of-patch token but introduces an auxiliary model solely for patching. In contrast, our method avoids the need for such a model and handles the overhead introduced by end-of-patch tokens via a second-stage BPE compression, which effectively reduces the average patch size.

Other works, such as Nawrot et al. (2023), explore dynamic pooling through learnable models. While they also incorporate groupings informed by subword tokenizers, a key distinction is that they explicitly train a model to replicate the behavior of a BPE tokenizer—an essential step in the absence of explicit patch boundary markers. Earlier research has also attempted character-level compression using convolutional layers prior to applying a global Transformer (Clark et al., 2022; Salesky et al., 2021; Rust et al., 2023).

Model	Param	BPB ↓	MMLU acc ↑	Hella acc_norm ↑	Lmb. acc ↑	ARC-c acc_norm ↑	ARC-e acc ↑	Wino. acc ↑	Piqa acc ↑
BPE	359M	1.02	23	31.3	29.89	**23.63**	38.05	50.5	61.43
Entropy-Patch	575M	1.10	23.07	32.91	29.4	20.9	36.6	48.8	59.6
Space-Patch	575M	1.03	**23.24**	**35.9**	**32.8**	21.5	39.02	52.5	**64.2**
BPE-Patch	323M	**0.98**	22.9	34.6	32.5	22.1	**40.6**	**53.4**	63.3

Table 4: **Common Academic Evaluation Benchmarks** comparison between standard embedding matrix and our learnable embeddings. The individual task performance is measured via zero-shot.

Space-based grouping. Space patching is a well-studied method, with several recent works exploring its effectiveness at medium to large scales, including Neitemeier et al. (2025), Slagle (2024), Sun et al. (2023), and Thawani et al. (2023). The latter introduces a minor modification to the hierarchical architecture, where the local encoder and decoder utilize more than one hidden representation per patch. Prior to space patching, grouping of consecutive characters was explored by YU et al. (2023).

Token Grouping. Recent work has also investigated the consecutive grouping of tokens (Ho et al., 2024). This is related to our approach in that our hierarchical BPE tokenization can be interpreted as a form of dynamic grouping, where tokens are composed from a much smaller vocabulary V'.

5 Conclusion

In this work, we presented a dynamic character grouping method that leverages BPE token boundaries to define patches, eliminating the need for an auxiliary neural network to determine patch segmentation. By augmenting the BPE process with an explicit end-of-patch token and introducing a second-stage compression step, we control patch granularity while maintaining a compact vocabulary. This design bridges the gap between character-level flexibility and subword-level efficiency, resulting in a hierarchical representation that generalizes beyond whitespace-delimited languages. Furthermore, our method can also be interpreted as a dynamic token grouping strategy. The empirical results demonstrate improved efficiency and performance over entropy-based patching and standard BPE, with competitive results relative to space-based grouping. These findings highlight the effectiveness of token-level dynamic grouping as a lightweight yet expressive alternative to conventional tokenization strategies in large language models.

Limitations

While our method demonstrates strong performance across several benchmarks, there are key limitations worth highlighting.

First, we do not evaluate our approach in large-scale regimes involving models with billions of parameters. It remains an open question how well token-level dynamic grouping scales in such settings, where training dynamics, memory constraints, and optimization challenges may differ substantially.

Finally, our experiments are limited to the GPT-2, LLaMA3 pre-trained tokenizer and SentencePiece tokenizers trained in-house. While this setup allows us to evaluate our method in a controlled and reproducible manner, future research should investigate the impact of using other widely adopted pre-trained tokenizers—such as those used in Gemma, or PaLM—to assess the generality of our approach across different tokenizer vocabularies.

References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. PIQA: Reasoning about Physical Commonsense in Natural Language. *Preprint*, arXiv:1911.11641.

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.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *Preprint*, arXiv:1803.05457.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h,

Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The Language Model Evaluation Harness.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. *Preprint*, arXiv:2009.03300.

Namgyu Ho, Sangmin Bae, Taehyeon Kim, Hyunjik Jo, Yireun Kim, Tal Schuster, Adam Fisch, James Thorne, and Se-Young Yun. 2024. Block Transformer: Global-to-Local Language Modeling for Fast Inference. *Preprint*, arXiv:2406.02657.

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, and 3 others. 2022. Training Compute-Optimal Large Language Models. *Preprint*, arXiv:2203.15556.

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)*, page 6403–6417. Association for Computational Linguistics.

Pit Neitemeier, Björn Deiseroth, Constantin Eichenberg, and Lukas Balles. 2025. Hierarchical autoregressive transformers: Combining byte- and word-level processing for robust, adaptable language models. In *The Thirteenth International Conference on Learning Representations*.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. 2024. Byte Latent Transformer: Patches Scale Better Than Tokens. *Preprint*, arXiv:2412.09871.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context. *Preprint*, arXiv:1606.06031.

Phillip Rust, Jonas F. Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. 2023. Language modelling with pixels. *Preprint*, arXiv:2207.06991.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Preprint*, arXiv:1907.10641.

Elizabeth Salesky, David Etter, and Matt Post. 2021. Robust open-vocabulary translation from visual text representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7235–7252, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kevin Slagle. 2024. SpaceByte: Towards Deleting Tokenization from Large Language Modeling. *Preprint*, arXiv:2404.14408.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Li Sun, Florian Luisier, Kayhan Batmanghelich, Dinei Florencio, and Cha Zhang. 2023. From Characters to Words: Hierarchical Pre-trained Language Model for Open-vocabulary Language Understanding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3605–3620, Toronto, Canada. Association for Computational Linguistics.

Avijit Thawani, Saurabh Ghanekar, Xiaoyuan Zhu, and Jay Pujara. 2023. Learn Your Tokens: Word-Pooled Tokenization for Language Modeling. *Preprint*, arXiv:2310.11628.

Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, Chenxia Li, Liu Yang, Xilin Luo, Xuejie Wu, Lunan Liu, Wenjun Cheng, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, and 11 others. 2023. Skywork: A More Open Bilingual Foundation Model. *Preprint*, arXiv:2310.19341.

LILI YU, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. 2023. MEGABYTE: Predicting Million-byte Sequences with Multiscale Transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? *Preprint*, arXiv:1905.07830.