上海交通大學
Shanghai Jiao Tong University

Byte Latent Transformer: Patches Scale Better Than Tokens

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†⋄, Srinivasan Iyer

FAIR at Meta, Paul G. Allen School of Computer Science & Engineering, University of Washington, University of Chicago

2024-12-12

包书勇

2024-12-26

引言

分词化/tokenize 化，是模型理解自然语言的最小单元。但是分词化存在一些局限性诸如领域/模态敏感性（Dagan等人，2024）、对输入噪声的敏感性（第6节）、缺乏正字法知识（Edman等人，2024）以及多语言不平等、计算开销等问题。

BLT 的核心思想：

- 直接处理原始字节： 与传统的基于词元的模型不同，BLT 直接使用构成文本的最小数字单位-字节进行处理。这从根本上消除了对分词的需求，避免了分词可能带来的误差和局限性。
- 动态 patching： 这是 BLT 的关键创新。它根据文本的复杂度将字节组合成不同长度的“patch”（可以理解为小块）。对于简单的字节序列，BLT 会使用较大的 patch，从而节省计算资源；而对于复杂的区域，则使用较小的 patch，以提高精度。这种动态调整的方式使得 BLT 能够更有效地利用计算资源。

Scaling Trends

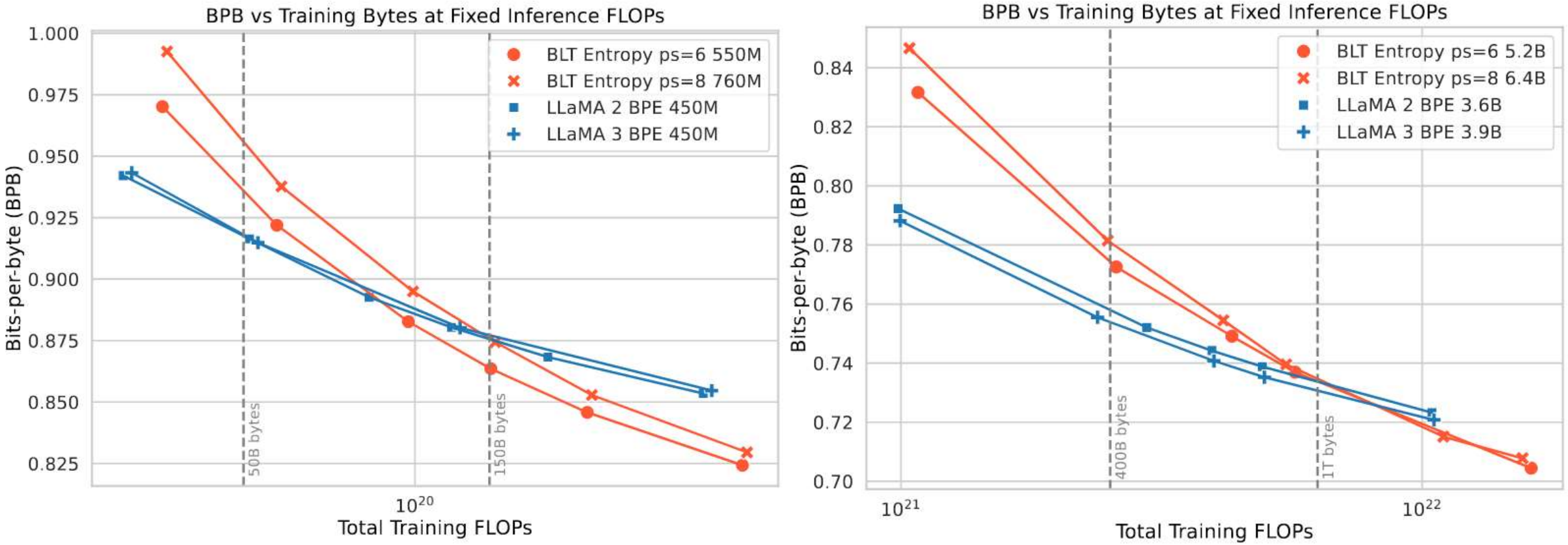


Figure 1 Scaling trends for fixed inference FLOP models (fully) trained with varying training budgets. In token-based models, a fixed inference budget determines the model size. In contrast, the BLT architecture provides a new scaling axis allowing simultaneous increases in model and patch size while keeping the same training and inference budget. BLT patch-size (ps) 6 and 8 models quickly overtake scaling trends of BPE Llama 2 and 3. Moving to the larger inference budget makes the larger patch size 8 model more desirable sooner. Both BPE compute-optimal point and crossover point are indicated with vertical lines.

Scaling Trend

Bits-Per-Byte (BPB) 估计

- Perplexity only makes sense in the context of a fixed tokenizer as it is a measure of the uncertainty for each token.
- Bits-Per-Byte (BPB) 是一种与分词器无关的度量方法，用于评估模型对字节序列的预测不确定性。BPB 的计算公式如下：

$$\mathrm{BPB}(x) = \frac{\mathcal{L}_{CE}(\boldsymbol{x})}{\ln(2) \cdot n_{\mathrm{bytes}}}$$

BPB 的优点

- 与分词器无关：

BPB 直接基于字节计算，不依赖于任何分词器，适用于字节级模型和token级模型的比较。

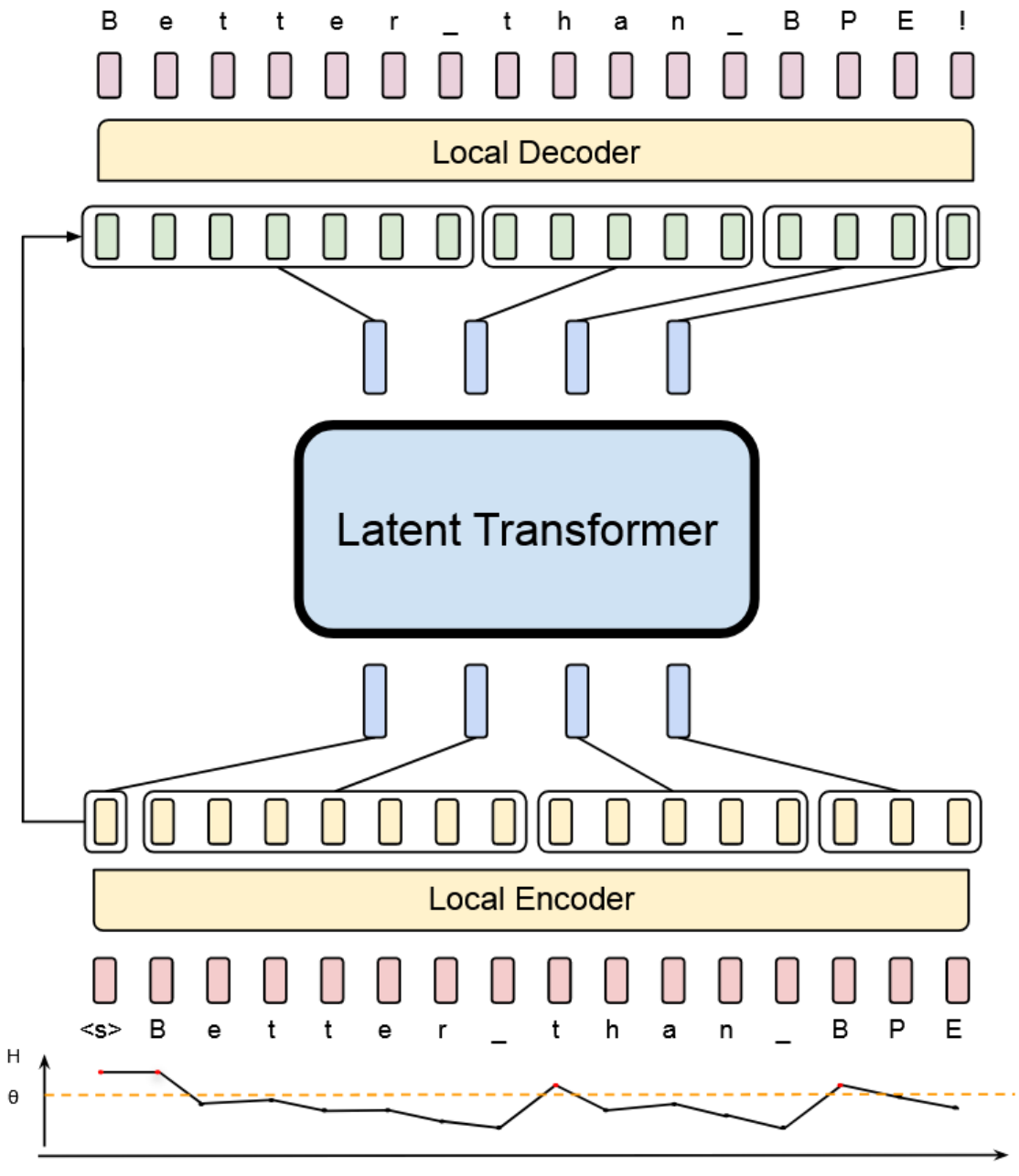
- 统一度量：

BPB 提供了一种统一的度量方法，使得不同模型之间的比较更加公平和直观。

- 适用于多种任务：

BPB 可以用于评估各种字节级任务，如语言建模、文本生成、机器翻译等。

BLT



5. Small Byte-Level Transf. Makes **Next-Byte Prediction**

4. **Unpatching** to Byte Sequence via Cross-Attn

3. Large Latent Transformer **Predicts Next Patch**

2. Entropy-Based Grouping of Bytes Into **Patches** via Cross-Attn

1. Byte-Level Small Transf. Encodes **Byte Stream**

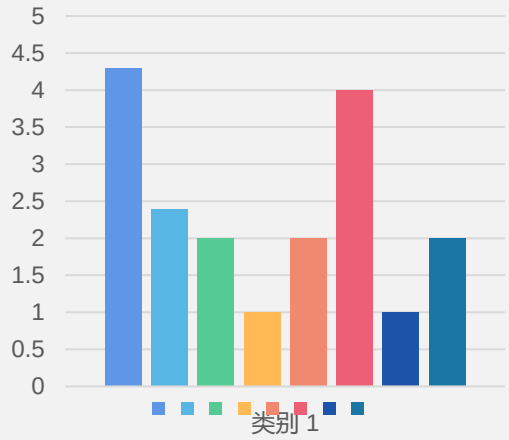
BLT

Transformer

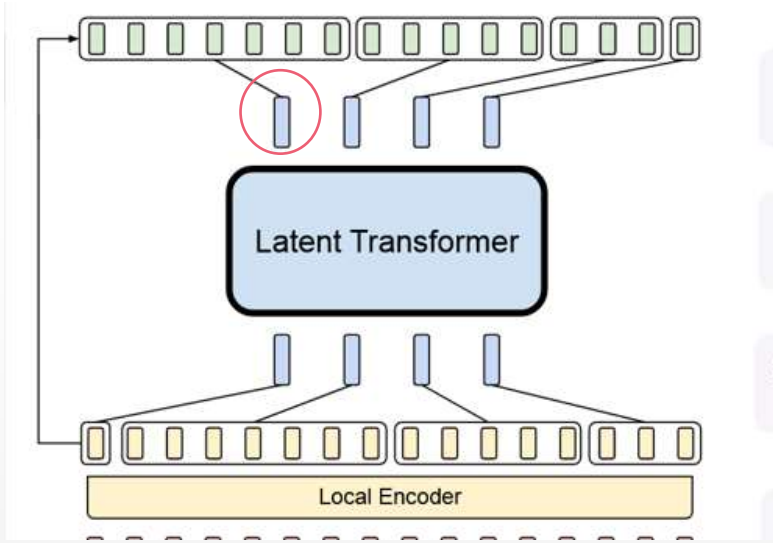
History

Token

Transformer



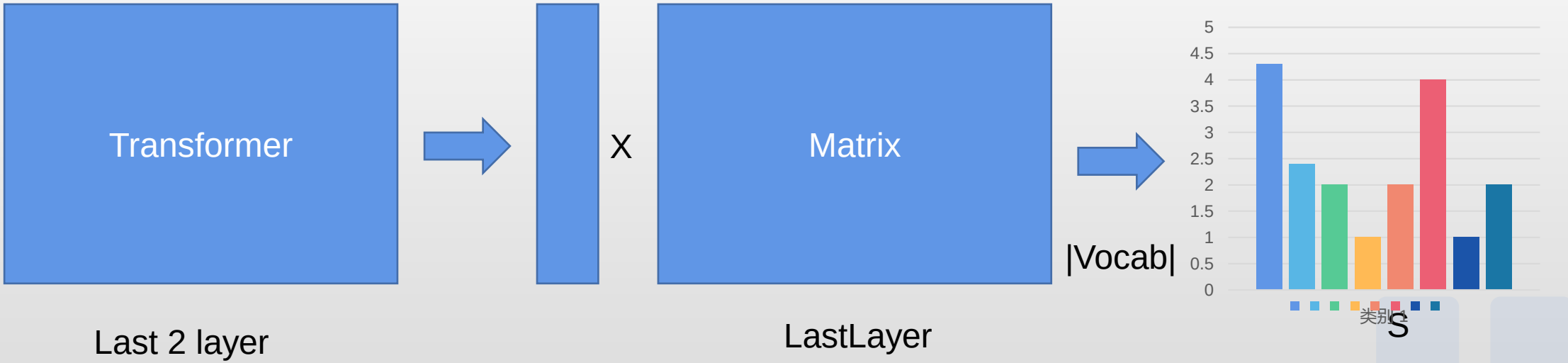
BLT



Transformer

hidden signal

H



Last 2 layer

LastLayer

Tokenization

将这些文本分割成模型可以操作的独立部分。
分词的基本方法

- 字符级分词：将每个字符（包括空格）分割成一个部分，但会导致序列过长，增加计算复杂度。

- 按空格分词：将文本按空格分割成单词，每个单词作为一个token。这种方法在很长一段时间内是标准的做法。

Tokenization

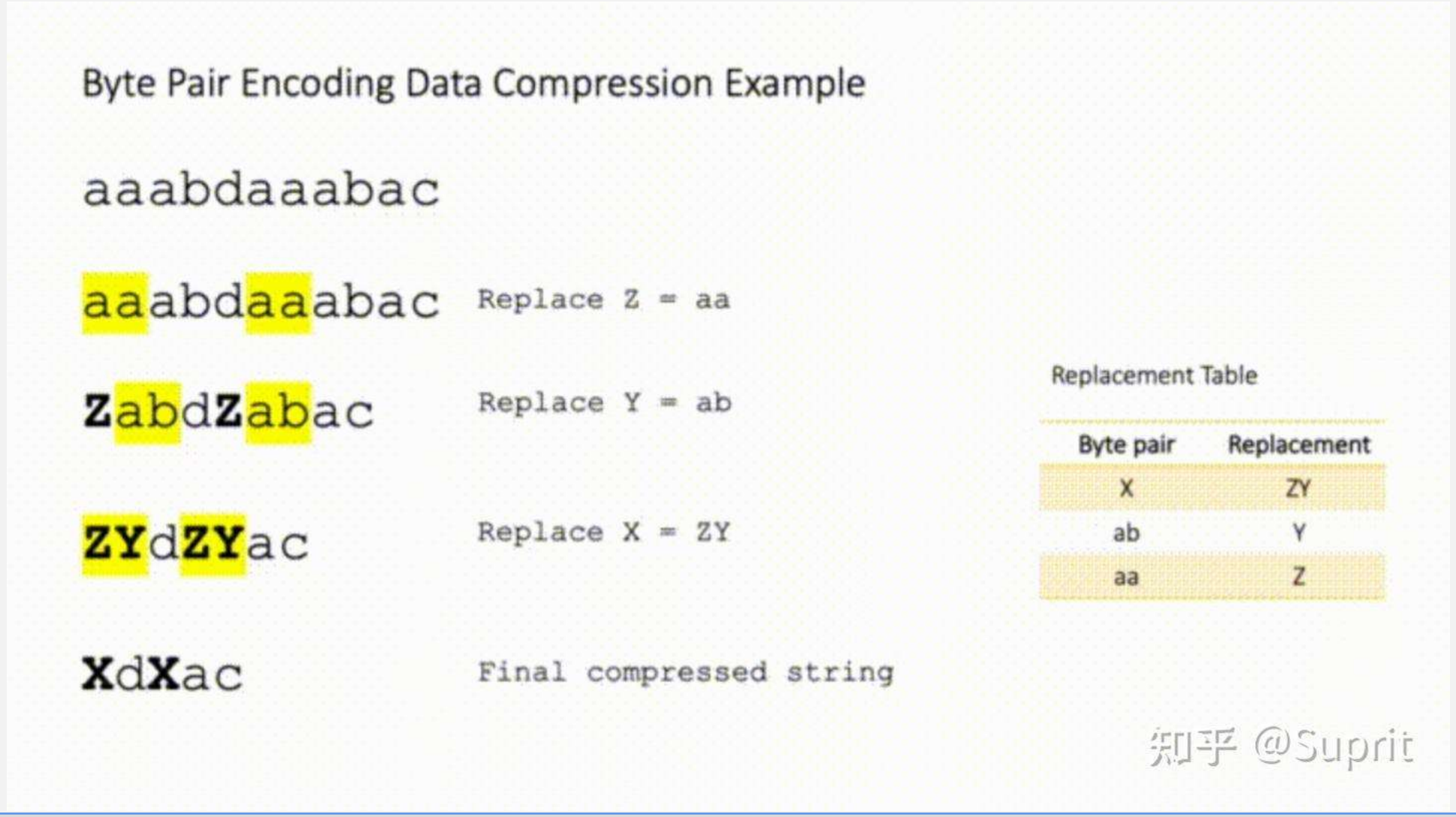
子词分词方法

- Byte Pair Encoding (BPE) 和 WordPiece Encoding：这些方法通过将文本分割为子词（subword）来解决词汇表外问题。

- 基本单元：子词分词方法使用一组基本单元（如字符或字节）来构建所有可能的文本组合。

子词分词方法本质上是一种压缩算法，通过组合频繁出现的字符或字节来减少token数量。

Tokenization



Tokenization

BPE

1. 获取语料库，这样一段话为例： “FloydHub is the fastest way to build, train and deploy deep learning models. Build deep learning models in the cloud. Train deep learning models.”
2. 拆分，加后缀，统计词频：

WORD	FREQUENCY	WORD	FREQUENCY
d e e p </w>	3	b u i l d </w>	1
l e a r n i n g </w>	3	t r a i n </w>	1
t h e </w>	2	a n d </w>	1
m o d e l s </w>	2	d e p l o y </w>	1
F l o y d h u b </w>	1	B u i l d </w>	1
i s </w>	1	m o d e l s </w>	1
f a s t e s t </w>	1	i n </w>	1
w a y </w>	1	c l o u d </w>	1
t o </w>	1	T r a i n </w>	1

Tokenization

BPE

1. 建立词表，统计字符频率（顺便排个序）：

NUMBER	TOKEN	FREQUENCY	NUMBER	TOKEN	FREQUENCY
1	</w>	24	15	g	3
2	e	16	16	m	3
3	d	12	17	.	3
4	l	11	18	b	2
5	n	10	19	h	2
6	i	9	20	F	1
7	a	8	21	H	1
8	o	7	22	f	1
9	s	6	23	w	1
10	t	6	24	,	1
11	r	5	25	B	1
12	u	4	26	c	1
13	p	4	27	T	1
14	y	3			

Tokenization

BPE

1. 以第一次迭代为例，将字符频率最高的 d 和 e 替换为 de，后面依次迭代：

NUMBER	TOKEN	FREQUENCY	NUMBER	TOKEN	FREQUENCY
1	</w>	24	16	g	3
2	e	16 - 7 = 9	17	m	3
3	d	12 - 7 = 5	18	.	3
4	l	11	19	b	2
5	n	10	20	h	2
6	i	9	21	F	1
7	a	8	22	H	1
8	o	7	23	f	1
9	de	7	24	w	1
10	s	6	25	,	1
11	t	6	26	B	1
12	r	5	27	c	1
13	u	4	28	T	1
14	p	4			
15	y	3			

Tokenization

BPE

1. 更新词表

WORD	FREQUENCY	WORD	FREQUENCY
d e e p </w>	3	b u i l d </w>	1
l e a r n i n g </w>	3	t r a i n </w>	1
t h e </w>	2	a n d </w>	1
m o d e l s </w>	2	d e p l o y </w>	1
F l o y d h u b </w>	1	B u i l d </w>	1
i s </w>	1	m o d e l s </w>	1
f a s t e s t </w>	1	i n </w>	1
w a y </w>	1	c l o u d </w>	1
t o </w>	1	T r a i n </w>	1

Tokenization

BPE

继续迭代直到达到预设的 subwords 词表大小或下一个最高频的字节对出现频率为 1。
如果将词表大小设置为 10，最终的结果为：

```
d e
r n
rn i
rni n
rnin g</w>
o de
ode l
m odel
l o
l e
```

Tokenization

Tokenization的缺点

1. 固定词汇表的限制
问题：传统的分词方法依赖于一个固定的词汇表，所有文本被分割为这个词汇表中的token。这种固定词汇表的方式无法处理词汇表外的单词（Out-of-Vocabulary, OOV）。
影响：当遇到未见过的单词（如新词、专有名词或拼写错误）时，模型无法正确处理，导致性能下降。

2. 领域和模态的敏感性
问题：分词方法通常在特定领域或模态上表现良好，但在其他领域或模态上可能失效。例如，BPE分词器在自然语言文本上表现良好，但在代码或数学符号上可能表现不佳。
影响：模型在处理多领域或多模态数据时，性能可能不稳定。

3. 输入噪声的敏感性
问题：分词方法对输入噪声（如拼写错误、大小写变化）非常敏感。例如，随机大小写变化可能导致同一个单词被分割为不同的token。
影响：模型在处理噪声数据时，性能可能显著下降。

Tokenization的缺点

4. 多语言不公平性
问题：分词方法在处理不同语言时可能存在不公平性。例如，某些语言的词汇表可能比其他语言更大或更复杂，导致这些语言的性能较差。
影响：模型在多语言任务中的性能可能不均衡。

5. 计算效率问题
问题：字符级和字节级分词虽然灵活，但序列长度过长，导致计算效率低下。子词分词虽然减少了序列长度，但需要额外的分词步骤，增加了预处理复杂度。
影响：模型的计算效率和运行时间可能受到影响。

6. 分词的不可逆性
问题：分词过程通常是不可逆的，即将文本分割为token后，无法完全恢复原始文本。例如，BPE分词器将单词分割为子词后，原始单词的拼写信息可能丢失。
影响：模型在处理需要精确拼写信息的任务（如拼写纠正）时，性能可能受限。

Tokenization

Retrofitting Large Language Models with Dynamic Tokenization

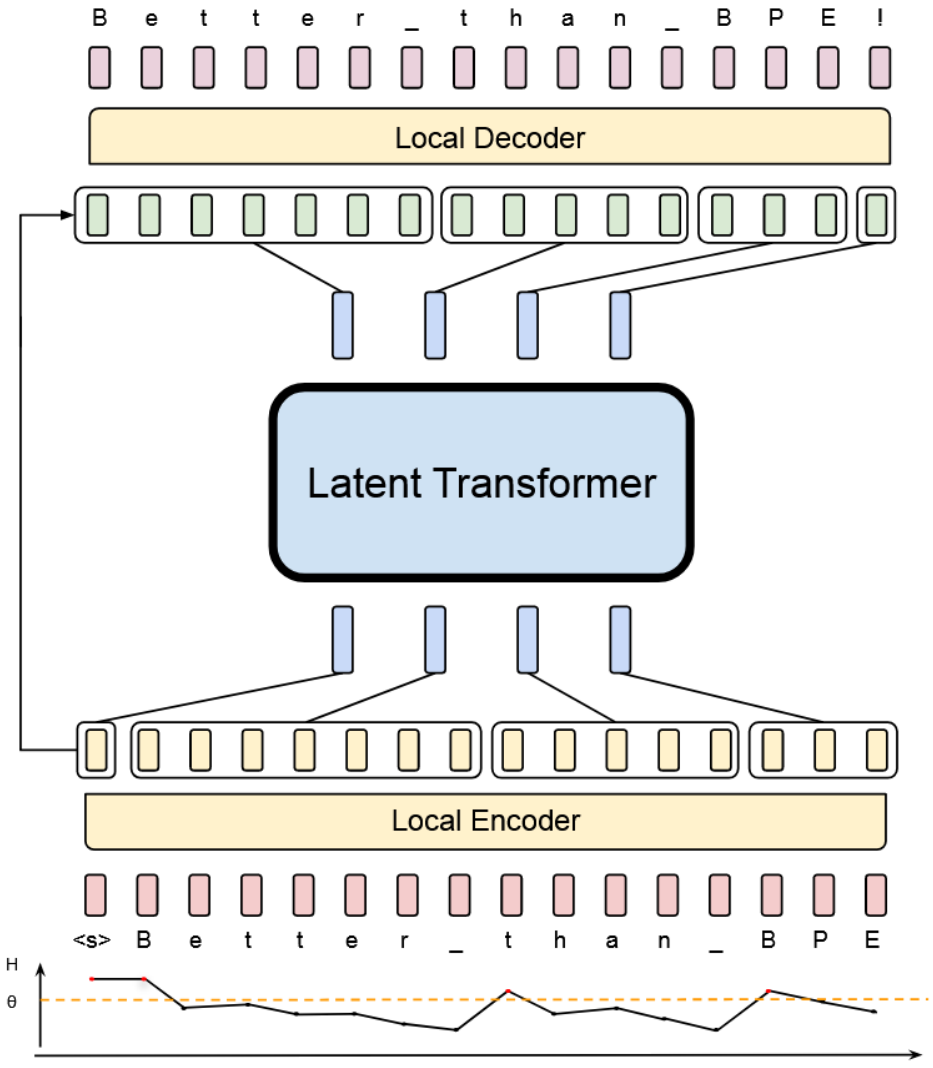
Darius Feher **Ivan Vulić** **Benjamin Minixhofer**
University of Cambridge
{dmf45, bm644, iv250}@cam.ac.uk

针对之前的tokenization的局限性，也有相关工作在进行。
这篇是提出动态分词，根据输入文本动态调整词边界。该方法受字节对编码（BPE）启发，允许在推理时动态合并子词。具体而言，我们在文本批次中合并高频子词序列，并通过预训练的嵌入预测超网络动态生成词嵌入。实验表明，该方法在减少标记序列长度的同时，保持了模型性能。

Language	Original Subword Tokenization	#tokens
English	A sub/stantial im/prove/ment fosters further im/prove/ment/s	12
Swahili	U/bor/esh/aj/i mk/ub/wa una/ku/za u/bor/esh/aj/i za/idi	18
#merges	**Dynamic Tokenization**	**#tokens**
1	A sub/stantial *improve*/ment fosters further *improve*/ment/s	10 (83%)
1	U/*boresh*/aj/i mk/ub/wa una/ku/za u/*boresh*/aj/i za/idi	16 (89%)
2	A sub/stantial *improvement* fosters further *improvement*/s	8 (67%)
2	U/*boreshaj*/i mk/ub/wa una/ku/za u/*boreshaj*/i za/idi	14 (78%)
4	A *substantial improvement* fosters further *improvements*	6 (50%)
11	*Uboreshaji mkubwa unakuza uboreshaji zaidi*	5 (28%)

Table 1: Comparison of static subword vs dynamic tokenization for the same sentences in **English** and **Swahili**. Embeddings for tokens in *blue* are obtained using a hypernetwork (HN) which composes the subword-level embeddings, as highlighted by /. The last row shows the number of merges required to achieve word-level tokenization, serving as a 'compression upper bound' of the proposed approach. The percentages show the fraction of the original token count remaining after merging.

Tokenization



5. Small Byte-Level Transf. Makes **Next-Byte Prediction**

4. **Unpatching** to Byte Sequence via Cross-Attn

3. Large Latent Transformer **Predicts Next Patch**

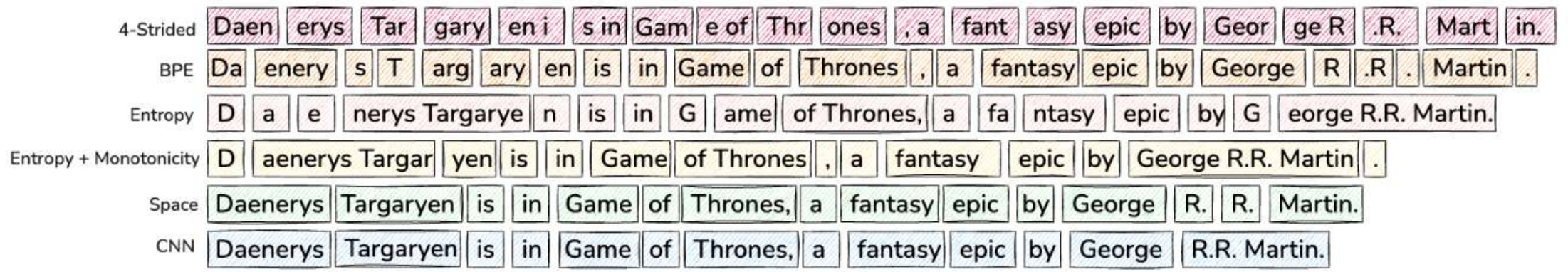
2. Entropy-Based Grouping of Bytes Into **Patches** via Cross-Attn

1. Byte-Level Small Transf. Encodes **Byte Stream**

Patching

Patching

	pros	cons
4-strided	simple	No dynamic compute allocation. Inconsistent patching
space	consistent patching. compute is allocated for each word.	Not good for all languages and domains(such as math) Pre-given patch size
entropy	Dynamic allocation of compute where it is needed	



Patching

Patching

We train a small byte-level auto-regressive language model on the training data for BLT and compute next byte entropies under the LM distribution pe over the byte vocabulary V:

$$H(x_i) = \sum_{v \in \mathcal{V}} p_e(x_i = v | \boldsymbol{x}_{<i}) \log p_e(x_i = v | \boldsymbol{x}_{<i})$$

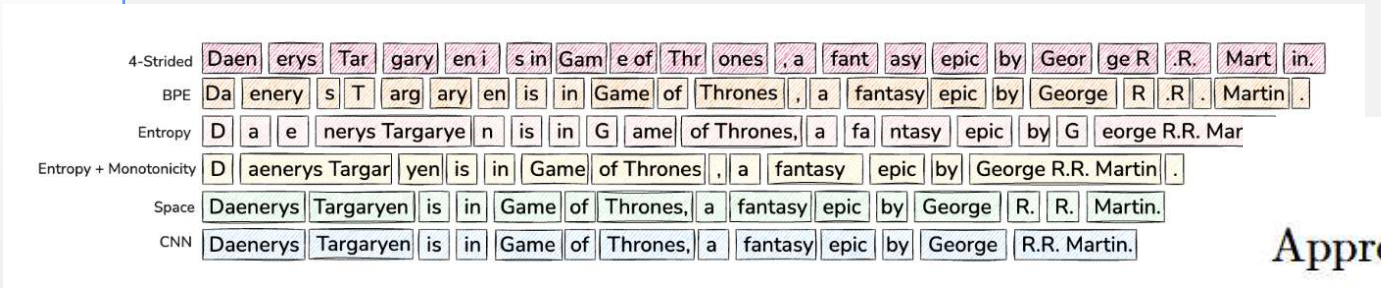
identify patch boundaries given entropies H(xi).
The first, finds points above a global entropy threshold, as illustrated in Figure 4.
The second, identifies points that are high relative to the previous entropy. The second approach can also be interpreted as identifying points that break approximate monotonically decreasing entropy withing the patch.

$$\begin{aligned} &\text{Global Constraint} && H(x_t) > \theta_g \\ &\text{Approx. Monotonic Constraint} && H(x_t) - H(x_{t-1}) > \theta_r \end{aligned}$$

Patch boundaries are identified during a lightweight preprocessing step executed during dataloading. This is different from Nawrot et al. (2023) where classifier is trained to predict entropy-based patch boundaries.

Patching

Patching



$$\text{Global Constraint} \quad H(x_t) > \theta_g$$

$$\text{Approx. Monotonic Constraint} \quad H(x_t) - H(x_{t-1}) > \theta_r$$

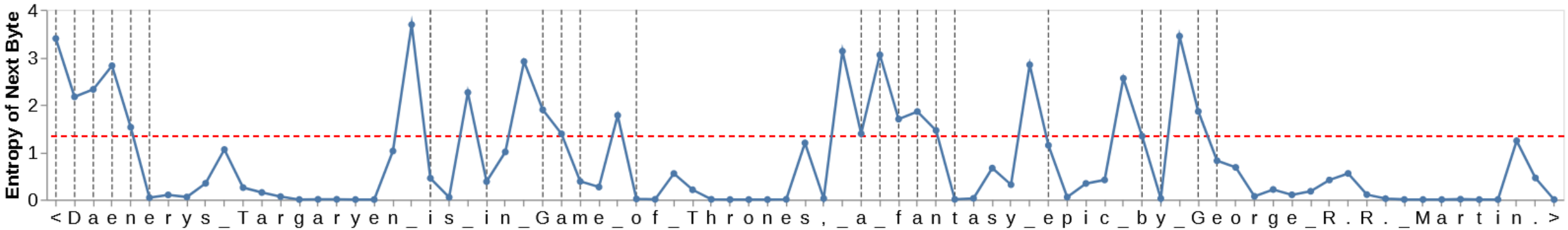


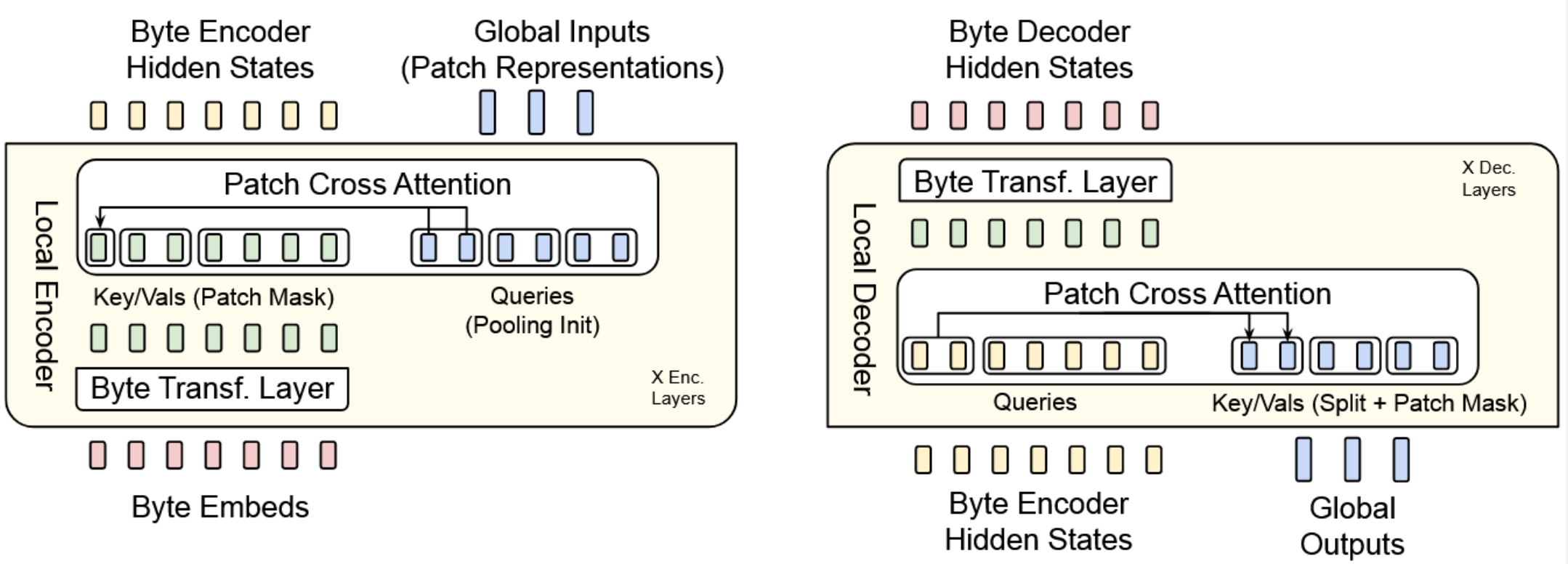
Figure 4 This figure plots the entropy $H(x_i)$ of each byte in "Daenerys Targeryen is in Game of Thrones, a fantasy epic by George R.R. Martin." with spaces shown as underscores. Patches end when $H(x_i)$ exceeds the global threshold θ_g, shown as a red horizontal line. The start of new patches are shown with vertical gray lines. For example, the entropies of "G" and "e" in "George R.R. Martin" exceed θ_g, so "G" is the start of a single byte patch and "e" of a larger patch extending to the end of the named entity as the entropy $H(x_i)$ stays low, resulting in no additional patches.

Patching

"tokens" to refer to byte-groups drawn from a finite vocabulary determined prior to training as opposed to "patches" which refer to dynamically grouped sequences without a fixed vocabulary.

For both token-based and patch-based models, the computational cost of processing data is primarily determined by the number of steps executed by the main Transformer.In BLT, this is the number of patches needed to encode the data with a given patching function. Consequently, the average size of a patch, or simply patch size, is the main factor for determining the cost of processing data during both training and inference with a given patching function

BLT


Byte Encoder
Hidden States
Global Inputs
(Patch Representations)
Local Encoder
Patch Cross Attention
Key/Vals (Patch Mask)
Queries
(Pooling Init)
Byte Transf. Layer
X Enc.
Layers
Byte Embeds
Byte Decoder
Hidden States
Local Decoder
Byte Transf. Layer
X Dec.
Layers
Patch Cross Attention
Queries
Key/Vals (Split + Patch Mask)
Byte Encoder
Hidden States
Global
Outputs

Encoder Hash n-gram Embeddings

在BLT 中，为了在每个步骤 i生成鲁棒且富有表现力的表示，关键是要结合前几个字节的信息。BLT通过同时建模单个字节bi和字节n-gram来实现这一点。具体来说，BLT 在每个字节位置 i 构建字节n-gram，并通过哈希函数将这些n-gram映射到固定大小的嵌入表中，最终生成增强的字节嵌入。

- 字节n-grams：

BLT将输入字节序列分割为n-grams（n从3到8），公式：

$g_{i,n} = \{b_{i-n+1}, ..., b_i\}$

例如，对于字节序列“help”，3-grams为“hel”,”elp”。

BLT

Encoder Hash n-gram Embeddings

- hash n-gram embeddings：

BLT使用哈希函数将n-grams映射到固定大小的嵌入表中，从而生成n-gram嵌入。
例如，对于3-gram “hel” ,”elp”，计算其哈希值并映射到嵌入表中的某个索引。

$$e_i = x_i + \sum_{n=3,\ldots,8} E_n^{hash}(\text{Hash}(g_{i,n}))$$

$$\text{where, } \text{Hash}(g_{i,n}) = \text{RollPolyHash}(g_{i,n})\%|E_n^{hash}|$$

Given a byte n-gram $g_{i,n} = \{b_{i-n+1}, \ldots, b_i\}$, the rolling polynomial hash of $g_{i,n}$ is defined as:

$$\text{Hash}(g_{i,n}) = \sum_{j=1}^{n} b_{i-j+1} a^{j-1}$$

Where a is chosen to be a 10-digit prime number.

- 增强字节表示：

BLT将n-gram嵌入与字节嵌入结合，生成更丰富的字节表示。
例如，对于字节 “a” ，结合其所在的3-gram “cat” 的嵌入，生成更上下文相关的表示。
BLT中n-grams的优点

Encoder Hash n-gram Embeddings

Ngram Sizes	Per Ngram Vocab	Total Vocab	BPB			
			Wikipedia	CC	Github	Train Dist
-	-	-	0.892	0.867	0.506	0.850
6,7,8	100k	300k	0.873	0.860	0.499	0.842
6,7,8	200k	600k	0.862	0.856	0.492	0.838
3,4,5	100k	300k	0.859	0.855	0.491	0.837
6,7,8	400k	1M	0.855	0.853	0.491	0.834
3,4,5	200k	600k	0.850	0.852	0.485	0.833
3,4,5,6,7,8	100k	600k	0.850	0.852	0.486	0.833
3,4,5	400k	1M	0.844	0.851	0.483	0.832
3,4,5,6,7,8	200k	1M	0.840	0.849	0.481	0.830
3,4,5,6,7,8	400k	2M	**0.831**	**0.846**	**0.478**	**0.826**

Table 8 Ablations on the use of n-gram hash embedding tables for a 1B BLT model trained on 100B bytes. We find that hash n-gram embeddings are very effective with very large improvements in BPB. The most significant parameter is the per-ngram vocab size and that smaller ngram sizes are more impactful than larger ones.

Encoder Hash n-gram Embeddings

- 增强字节表示：

BLT将n-gram嵌入与字节嵌入结合，生成更丰富的字节表示。
例如，对于字节“a”，结合其所在的3-gram“cat”的嵌入，生成更上下文相关的表示。
BLT中n-grams的优点

- 捕捉局部上下文：n-grams能够捕捉字节序列中的局部上下文信息，增强模型的表示能力。
- 计算效率：通过哈希函数，BLT能够高效地生成n-gram嵌入，避免存储所有可能的n-grams。
- 鲁棒性：n-grams能够增强模型对输入噪声（如拼写错误、大小写变化）的鲁棒性。

Encoder Multi-Headed Cross-Attention

在 BLT中，编码器多头交叉注意力模块的设计灵感来自于 Perceiver 架构（Jaegle et al., 2021），但有一个主要区别：BLT 的潜在表示对应于可变的分块表示，而不是固定的潜在表示集。此外，该模块只关注组成相应分块的字节。

$$P_{0,j} = \mathcal{E}_C(f_{\text{bytes}}((p_j)), f \text{ is a pooling function}$$

$$P_l = P_{l-1} + W_o\left(\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\right)$$

$$\text{where } Q_j = W_q(P_{l-1,j}), K_i = W_k(h_{l-1,i}), V_i = W_v(h_{l-1,i})$$

$$h_l = \text{Encoder-Transformer-Layer}_l(h_{l-1})$$

实现

预训练数据集：
Llama 2数据集（包含2T（万亿）token，经过清洗和过滤以提高质量)
BLT-1T数据集(包含1Ttoken，包括Datacomp-LM发布的一部分预训练数据。
model:
The entropy model in our experiments is a byte level language model trained on the same training distribution as the full BLT model.
we use a transformer with 100M parameters, 14 layers, and a hidden dimensionality of 512, and sliding window attention of 512 bytes.

实现细节

Entropy Model Context

使用 BLT模型结合熵模型（Entropy Model）时的一些优化细节，特别是在处理重复性较强的结构化内容（例如多项选择题任务）时如何避免“熵漂移”（Entropy Drift）以及如何提高推理效率。以下是关键内容的分解与解释：

核心问题

重复性内容导致熵漂移（Entropy Drift）：

- 在多项选择题等任务中，内容往往具有高度重复性（例如答案选项中的重复短语）。当直接使用熵模型计算时，这种重复性会导致熵值（Entropy）偏低。
- 偏低的熵值可能导致模型在重复短语部分产生非常大的“补丁”（Patches），即大块的文本区域被认为是低熵区域。这可能提高推理效率，但对于推理质量（尤其是推理逻辑性和可解释性）可能不利。

解决方法

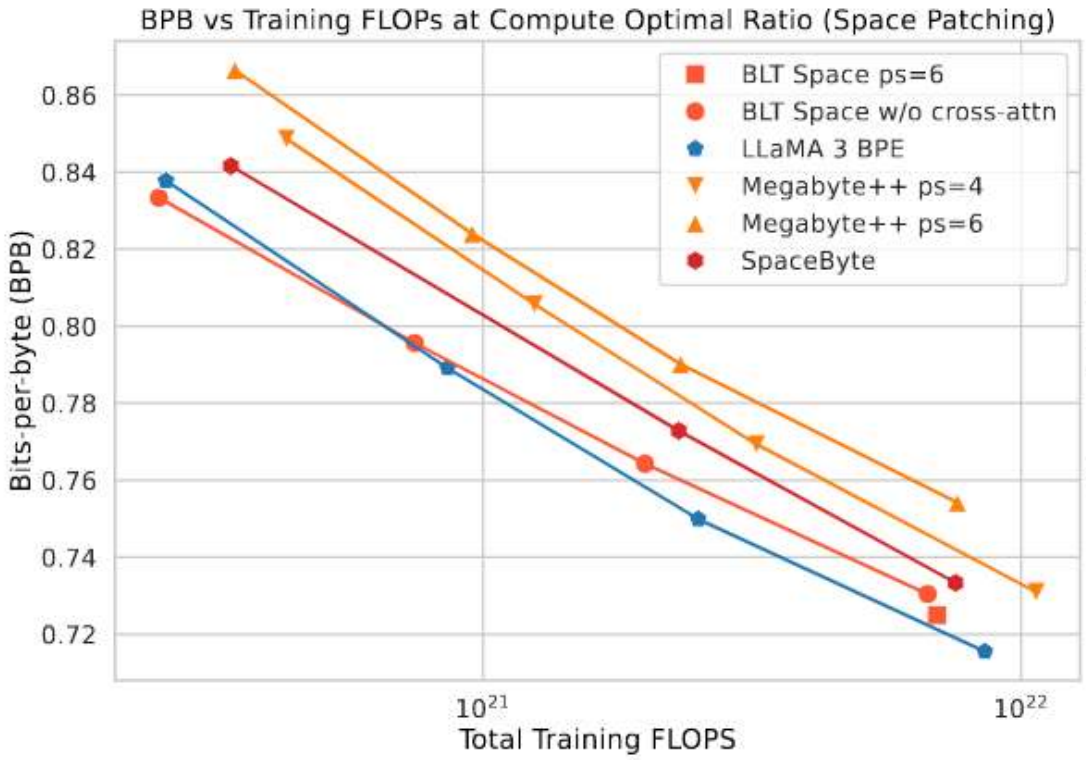
重置熵上下文

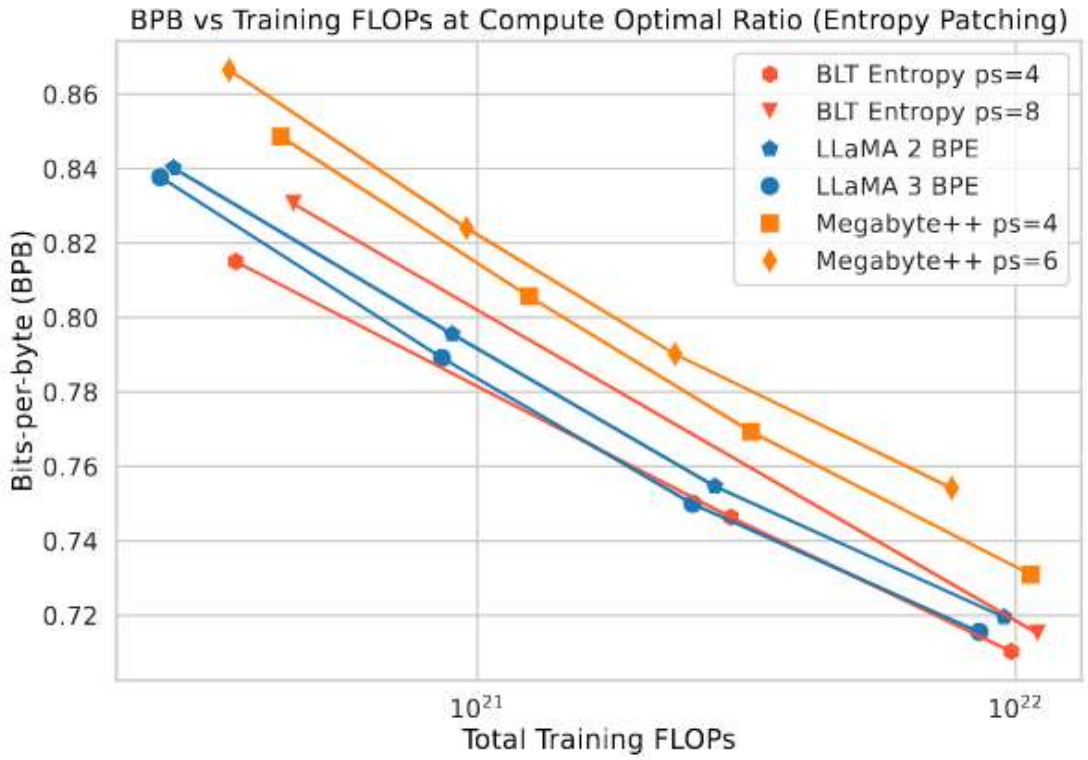
熵阈值（Entropy Threshold）的保持

实现细节

Figure 9 An example of default entropy-based patching with global threshold during inference on MMLU. Green denotes the prompt, Blue denotes the few-shot examples, and red denotes the question to be answered. Note that the size of the patches for the repeated phrases in the answer choices is much larger, which means that the global model is invoked significantly fewer times than its tokenizer-based counterpart, with this inference patching scheme.

实验结果





BLT matches or outperforms BPE models in scaling trends, validating its effectiveness in compute-optimal regimes.

Larger BLT patch sizes (6–8 bytes) improve performance and reduce inference FLOPs by up to 50%.

While models with larger patch sizes start with lower performance (e.g., at 1B parameters), they surpass BPE models at larger scales (e.g., 7B parameters).

实验结果

	Llama 3 1T Tokens	BLT-Space 6T Bytes	BLT-Entropy 4.5T Bytes
Arc-E	77.6	75.4	**79.6**
Arc-C	**53.3**	49.8	52.1
HellaSwag	79.1	79.6	**80.6**
PIQA	80.7	**81.1**	80.6
MMLU	**58.1**	54.8	57.4
MBPP	40.2	37.6	**41.8**
HumanEval	31.1	27.4	**35.4**
Average	60.0	58.0	**61.1**
Bytes/Patch on Train Mix	4.4	**6.1**	4.5

Table 1 Comparison of FLOP-matched BLT **8B** models trained on the BLT-1T dataset comprising high-quality tokens of text and code from publicly available sources, with baseline models using the Llama 3 tokenizer. BLT performs better than Llama 3 on average, and depending on the patching scheme, achieves significant FLOPs savings with a minor reduction in performance.

Llama 2	Llama 3	Entropy ps=6	Entropy ps=8	Inference FLOPs	Compute Optimal (Bytes)	Crossover (Bytes)
470m	450m	610m (1.2x)	760m (1.6x)	3.1E8	50B	150B
3.6B	3.9B	5.2B (1.3x)	6.6B (1.7x)	2.1E9	400B	1T

Table 2 Details of models used in the fixed-inference scaling study. We report non-embedding parameters for each model and their relative number compared to Llama 2. We pick model sizes with equal inference FLOPs per byte. We also indicate BPE's compute-optimal training data quantity and the crossover point where BLT surpasses BPE as seen in Figure 1 (both expressed in bytes of training data). This point is achieved at much smaller scales compared to many modern training budgets.

实验结果

Task	Prompt	Llama 3	BLT
Substitute Word	Question: Substitute " and " with " internet " in " She went to the kitchen and saw two cereals. ". Answer:	She went to the kitchen and saw two cereals.	She went to the kitchen internet saw two cereals.
Swap Char	Question: Swap " h " and " a " in " that ". Answer:	that	taht
Substitute Char	Question: Substitute " a " with " m " in " page ". Answer:	-	pmge
Semantic Similarity	Question: More semantically related to " are ": " seem ", " acre ". Answer:	acre	seem
Orthographic Similarity	Question: Closer in Levenshtein distance to " time ": " timber ", " period ". Answer:	period	timber
Insert Char	Question: Add an " z " after every " n " in " not ". Answer:	znotz	nzot

Figure 7 Output responses from Llama 3 and BLT models for various tasks from CUTE benchmark. BLT model performs better on sequence manipulation tasks compared to the tokenizer-based Llama 3 model. Note that few-shot examples are not shown in the above prompts to maintain clarity.

实验结果

拼写反转任务等任务表现出色

	Llama 3 (1T tokens)	Llama 3.1 (16T tokens)	BLT (1T tokens)
HellaSwag Original	79.1	80.7	**80.6**
HellaSwag Noise Avg.	56.9	64.3	**64.3**
- AntSpeak	45.6	61.3	**57.9**
- Drop	53.8	57.3	**58.2**
- RandomCase	55.3	65.0	**65.7**
- Repeat	57.0	61.5	**66.6**
- UpperCase	72.9	76.5	**77.3**
Phonology-G2P	11.8	18.9	**13.0**
CUTE	27.5	20.0	**54.1**
- Contains Char	0.0	0.0	**55.9**
- Contains Word	55.1	21.6	**73.5**
- Del Char	34.6	34.3	**35.9**
- Del Word	**75.5**	84.5	56.1
- Ins Char	7.5	0.0	**7.6**
- Ins Word	**33.5**	63.3	31.2
- Orthography	43.1	0.0	**52.4**
- Semantic	65	0.0	**90.5**
- Spelling	1.1	-	**99.9**
- Spelling Inverse	30.1	3.6	**99.9**
- Substitute Char	0.4	1.2	**48.7**
- Substitute Word	16.4	6.8	**72.8**
- Swap Char	2.6	2.4	**11.5**
- Swap Word	20.1	4.1	**21**

Table 3 We compare our 8B BLT model to 8B BPE Llama 3 trained on 1T tokens on tasks that assess robustness to noise and awareness of the constituents of language (best result bold). We also report the performance of Llama 3.1 on the same tasks and underline best result overall. BLT outperforms the Llama 3 BPE model by a large margin and even improves over Llama 3.1 in many tasks indicating that the byte-level awareness is not something that can easily be obtained with more data.

实验结果

语言翻译任务表现出色

Language	Language → English		English → Language	
	Llama 3	BLT	Llama 3	BLT
Arabic	22.3	24.6	10.4	8.8
German	41.3	42.0	29.8	31.2
Hindi	20.7	20.9	7.8	7.2
Italian	34.0	33.9	24.4	26.2
Vietnamese	31.2	31.0	28.4	23.7
Thai	17.9	18.1	10.5	7.7
Armenian	1.7	6.3	0.6	0.9
Amharic	1.3	3.1	0.4	0.5
Assamese	2.7	5.4	0.8	1.6
Bengali	4.7	12.7	1.7	4.1
Bosnian	36.0	37.3	16.9	19.6
Cebuano	18.2	20.6	5.8	9.1
Georgian	1.7	7.4	1.0	2.5
Gujarati	2.0	5.8	1.0	2.2
Hausa	5.75	5.9	1.2	1.3
Icelandic	16.1	17.9	4.8	5.3
Kannada	1.6	3.9	0.7	1.7
Kazakh	5.6	7.0	1.0	2.6
Kabuverdianu	20.3	20.9	5.1	6.8
Khmer	4.4	9.5	0.8	0.8
Kyrgyz	4.6	5.1	0.9	2.0
Malayalam	1.8	3.5	0.7	1.4
Odia	1.6	2.7	0.8	1.1
Somali	5.0	5.0	1.1	1.4
Swahili	10.1	12.0	1.4	2.3
Urdu	9.3	9.5	2.0	1.4
Zulu	4.7	5.0	0.6	0.5
Overall Average	12.1	**14.0**	5.9	**6.4**

Table 4 Performance of 8B BLT and 8B Llama 3 trained for 1T tokens on translating into and from six widely-used languages and twenty one lower resource languages with various scripts from the FLORES-101 benchmark (Goyal et al., 2022).

Limitations and possible improvements

Many experiments were conducted on models up to 1B parameters, and architectural choices might evolve as BLT scales to 8B parameters and beyond, reaching further performance improvements.

Existing transformer libraries are optimized for tokenizer-based architectures. While BLT uses efficient components like FlexAttention, its implementation is not yet fully optimized and could benefit from further refinement.

BLT currently relies on a separately trained entropy model for patching. Learning patching in an end-to-end manner is identified as a promising direction for enhancing performance.

总结

- 不同于传统 Transformer 直接在 token 级别处理输入,BLT 则是将输入转换为字节序列
- 然后通过一个编码器将字节序列映射到一个潜在空间(latent space)
- 在潜在空间中使用 Transformer 进行处理
- 最后通过解码器将结果映射回原始空间

Reference

1. Byte Latent Transformer: Patches Scale Better Than Tokens
2. Retrofitting Large Language Models with Dynamic Tokenization

上海交通大学
Shanghai Jiao Tong University

汇报结束 感谢聆听！

同时预祝大家2025新年快乐！！！！

包书勇

2024-12-26