Byte Latent Transformer: Patches Scale Better Than Tokens

Artidoro Pagnoni, Ram Pasunuru‡, Pedro Rodriguez‡, John Nguyen‡, Benjamin Muller, Margaret Li[1,◇], Chunting Zhou◇, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman[†,2,◇], Srinivasan Iyer†

FAIR at Meta, [1]Paul G. Allen School of Computer Science & Engineering, University of Washington, [2]University of Chicago

‡Joint second author, †Joint last author, ◇Work done at Meta

Presented by Hao Li & Jing Neng Hsu

UNIVERSITY OF TORONTO

Introducing BLT: A Dynamic Approach (in 5 sentences)

This paper introduces the **Byte Latent Transformer (BLT)**, a new large language model (LLM) architecture that **avoids fixed-vocabulary tokenization** by working directly with raw bytes.

BLT addresses these shortcomings by **learning “patches” of bytes**:

1. A **local encoder**
2. A **global “latent” transformer**
3. A **local decoder**

By dynamically allocating more compute for difficult parts of text and fewer steps for predictable ones, BLT **cuts down the overall inference cost** (up to ~50% savings in FLOPs) at parameter scales up to 8B.

The paper’s experiments show improvements:

- **Scaling**
- **Robustness**
- **Generalization**

Overall, the authors argue that **byte-level patching can be a strong alternative to traditional tokenization**, offering improved efficiency, robustness to text noise, and more flexible scaling for large language models.

UNIVERSITY OF TORONTO

The Problem with Traditional Tokenization

Large language models traditionally rely on fixed-vocabulary **tokenization** to preprocess text into subword tokens. This approach, however, has well-known drawbacks: tokenization can make models brittle to domain shifts or noisy input (e.g. typos) and can obscure character-level information. It also introduces biases (e.g. favoring well-represented languages/scripts over others).

Domain Sensitivity

Traditional tokenization creates biases in how strings are compressed, leading to domain and modality sensitivity issues.

Multilingual Inequity

Fixed vocabularies create inherent biases against certain languages, particularly those with different scripts or morphological structures.

Noise Vulnerability

Token-based models struggle with input noise, lacking robustness to character-level variations.

Limited Orthographic Knowledge

Token-based models lack direct access to character-level information, limiting their understanding of spelling and phonology.

UNIVERSITY OF TORONTO

The Promise of Byte-Level Processing

Robustness

Working directly at the **byte level** (i.e. on raw text bytes/characters) could eliminate tokenization issues and make models more **robust and language-agnostic**.

Language Agnostic

Byte-level processing treats all languages equally without favoring well-represented languages or scripts over others.

Character-Level Fidelity

Preserves important character-level information that might be lost during traditional tokenization processes.

In principle, working directly at the **byte level** (i.e. on raw text bytes/characters) could eliminate these issues and make models more **robust and language-agnostic**.

The Computational Challenge

Byte-by-Byte Processing

Processing every character individually creates extremely long sequences

Attention Complexity

Attention mechanisms scale quadratically with sequence length

Feed-Forward Networks

Running huge feed-forward networks at every sequence position becomes the main cost

Computational Waste

Wastes computation on predictable characters (e.g., last letters of common words)

In other words, a byte-level model wastes a lot of computation on *every* character – whether it's an informative one or just the last letter of a very predictable word.

Early Byte-Level Models and Their Limitations

ByT5 (Xue et al., 2022)

Showed the promise of tokenization-free NLP, but scaling such models is challenging due to very long sequences and prohibitive compute costs.

CANINE (Clark et al., 2022)

Used more efficient attention mechanisms to mitigate the computational challenges of byte-level processing.

Prior studies like ByT5 (Xue et al., 2022) showed the promise of *tokenization-free* NLP, but scaling such models is challenging: processing every byte leads to very long sequences and prohibitive compute cost for large transformers.

Xue, Linting, et al. "Byt5: Towards a token-free future with pre-trained byte-to-byte models." Transactions of the Association for Computational Linguistics 10 (2022): 291-306.
Clark, Jonathan H., et al. "Canine: Pre-training an efficient tokenization-free encoder for language representation." Transactions of the Association for Computational Linguistics 10 (2022): 73-91.

MegaByte: A Multiscale Transformer

Fixed-Length Byte Chunks

Processed data in fixed-length byte chunks (patches) to reduce sequence length

Local Decoder

Used a small local decoder to generate the bytes within each chunk

Decoder-Only Causal LLM

Concatenated groups of bytes into larger representations, then used a global transformer

Competitive Performance

Matched a token-based model around the 1B parameter scale on a 400 billion byte dataset

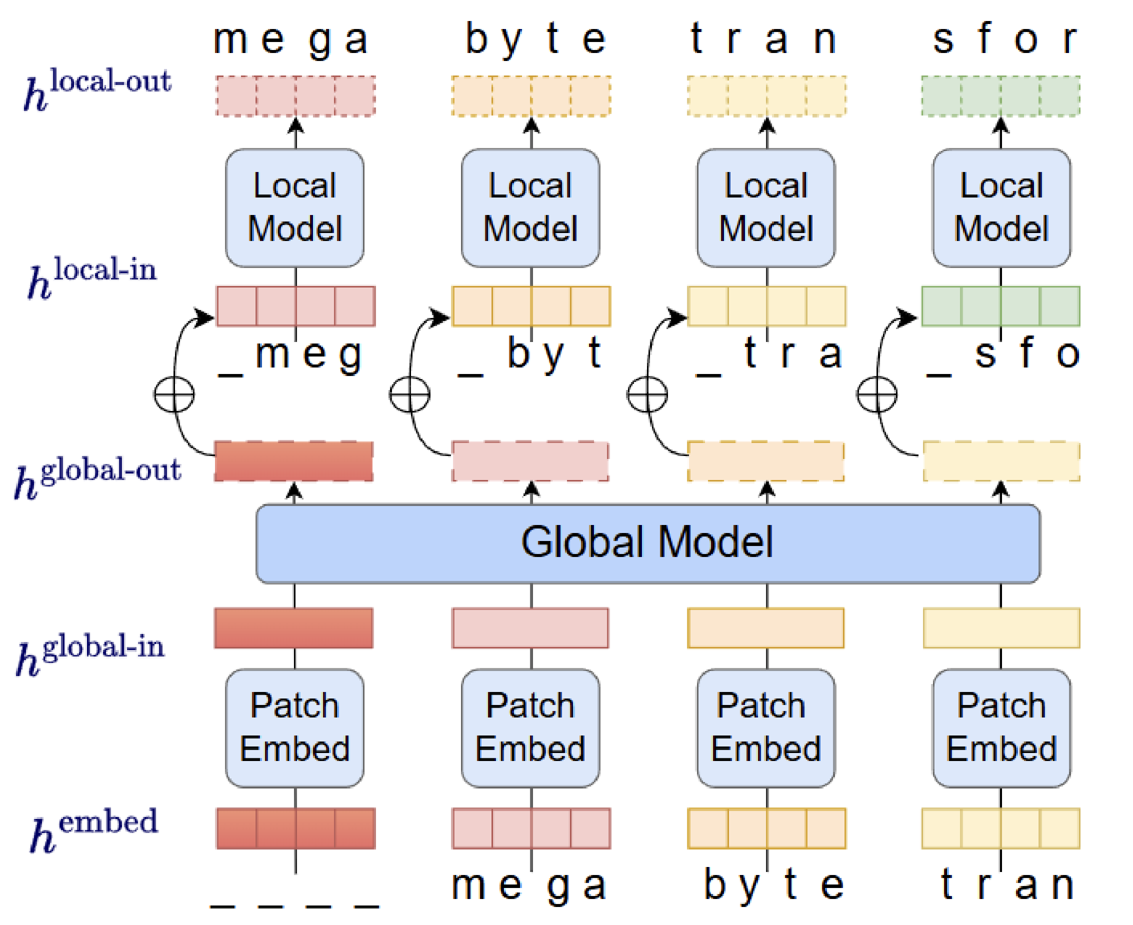


Figure 1. Overview of MEGABYTE with patch size $P = 4$. A small *local* model autoregressively predicts each patch byte-by-byte, using the output of a larger *global* model to condition on previous patches. Global and Local inputs are padded by P and 1 token respectively to avoid leaking information about future tokens.

A particularly influential predecessor of BLT was **MegaByte**, introduced by Yu et al. in 2023 (*Yu, Lili, et al. "Megabyte: Predicting million-byte sequences with multiscale transformers." Advances in Neural Information Processing Systems 36 (2023): 78808-78823.*).

Limitations of Static Patching

Performance Gap

Still lagged behind latest tokenized models

Static Chunking

One-size-fits-all approach was inefficient

Compute Budget Issues

Underperformed when controlling for compute

Lack of Adaptability

Couldn't adjust to text complexity

However, its **static patching** (e.g. always grouping, say, 8 bytes at a time) was a limitation. Follow-up analyses found that a one-size-fits-all chunking still lagged behind the latest tokenized models when controlling for the same compute budget.

Attempted Improvements to MegaByte

Natural Boundary Segmentation

Segmenting on natural boundaries like whitespace

Local Encoder Addition

Introducing a local encoder in addition to the decoder

Domain-Specific Improvements

Better results on certain domains (e.g. code from GitHub and scientific text from arXiv)

1B Parameter Scale

Improvements demonstrated at the 1B parameter scale

Yet, even with these tweaks, the static or heuristic patching strategies were not enough to fully close the gap with state-of-the-art token-based LLMs. This pointed to the need for **a more flexible, learned approach to grouping bytes** – which is exactly the problem the Byte Latent Transformer (BLT) set out to solve.

Introducing the Byte Latent Transformer

Eliminates Fixed Tokenization

Byte Latent Transformer (BLT) is a new byte-level LLM architecture that eliminates fixed tokenization

Adaptive "Byte Patching"

Introduces an adaptive **"byte patching"** mechanism to allocate computation *non-uniformly* across a text sequence

Focus on Unpredictable Parts

Focuses more resources on unpredictable parts of the input and fewer on well-predicted parts

Multi-Module Design

Features local encoder/decoder transformers that interface with a global latent transformer

The core idea is to **allocate computation *non-uniformly* across a text sequence**, focusing more resources on unpredictable parts of the input and fewer on well-predicted parts.

The BLT Architecture

Local Encoder

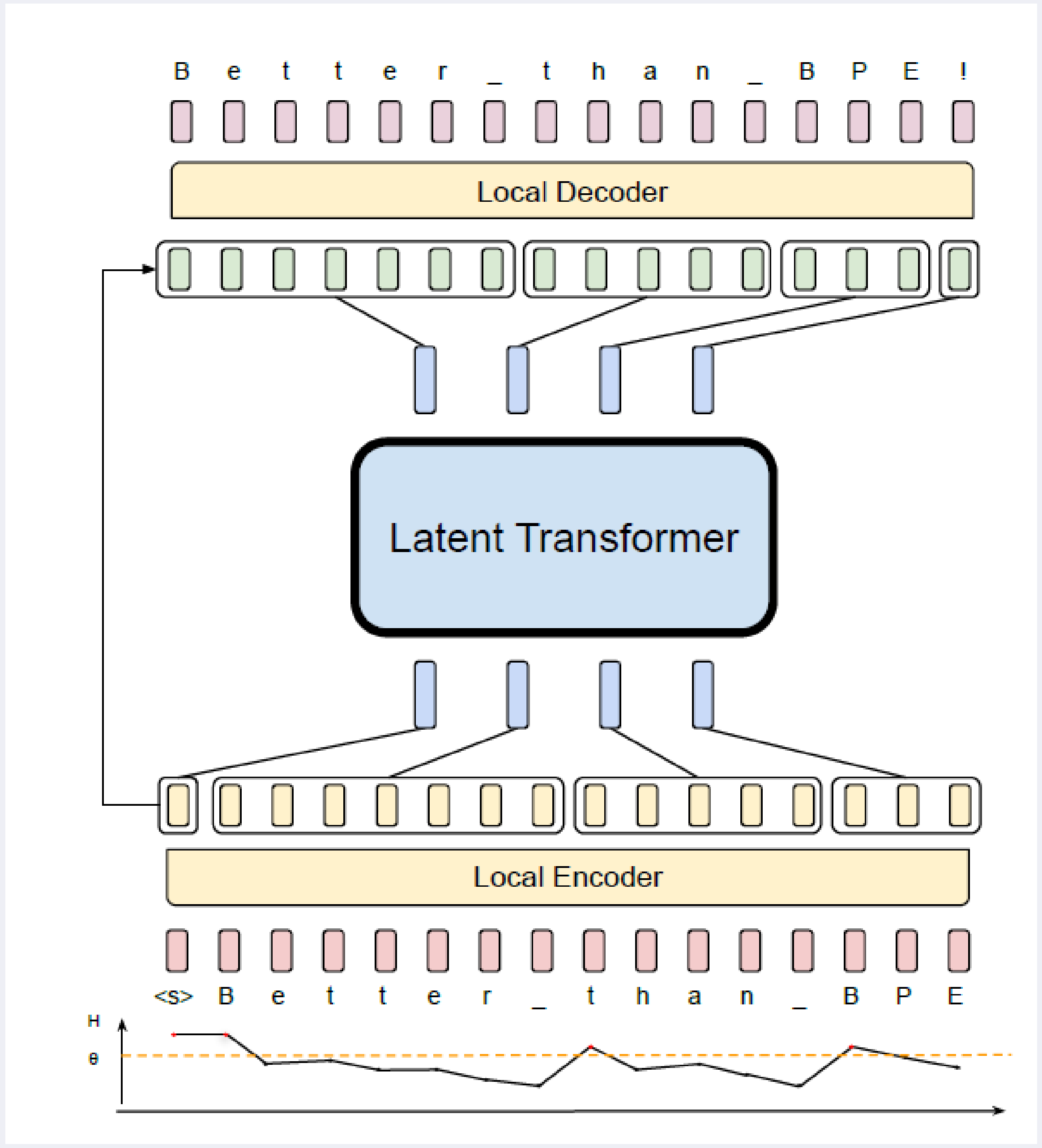
A local encoder groups consecutive bytes into patches based on a small auxiliary model's entropy estimates. Higher-entropy (unpredictable) regions of text get more, smaller patches (and thus more compute steps), while predictable regions can be grouped into fewer, larger patches.

Latent Transformer

A global "latent" transformer then operates on these patch representations—similar to how a standard LLM operates on token embeddings.

Local Decoder

A local decoder finally expands the predicted patch representations back into bytes.



B e t t e r _ t h a n _ B P E !
Local Decoder
Latent Transformer
Local Encoder
<s> B e t t e r _ t h a n _ B P E
H
θ

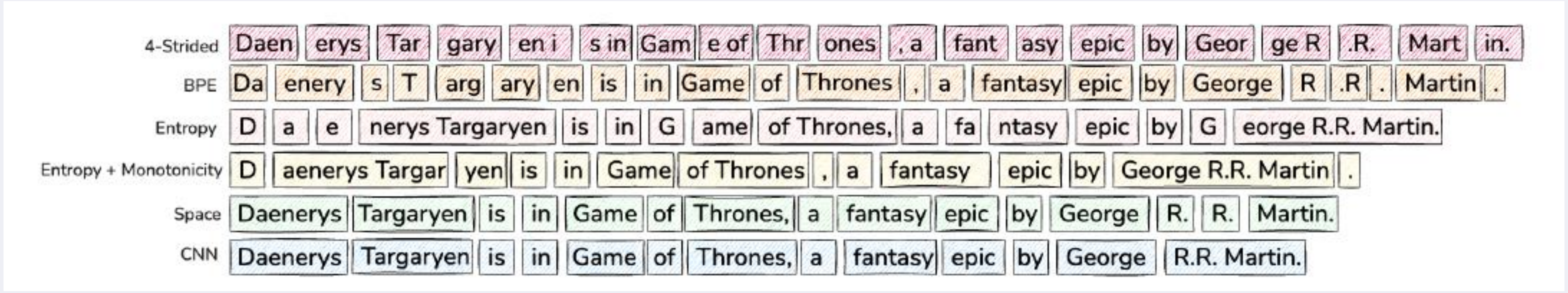
BLT also employs specialized attention connections between the local and global layers (e.g. a cross-attention that allows the latent transformer to peek into finer-grained byte details when needed) and byte n-gram embeddings to enrich the patch representations.

Patching: central to how BLT tackles sequence length

What is Patching?

Patching is the process of **grouping bytes into variable-sized units called "patches"** that serve as the primary units of computation. Unlike tokens, patches have no fixed vocabulary and can be of any length.

Patching Methods Compared


4-Strided: Daen | erys | Tar | gary | en i | s in | Gam | e of | Thr | ones | , a | fant | asy | epic | by | Geor | ge R | .R. | Mart | in.
BPE: Da | enery | s | T | arg | ary | en | is | in | Game | of | Thrones | , | a | fantasy | epic | by | George | R | .R | . | Martin | .
Entropy: D | a | e | nerys Targaryen | is | in | G | ame | of Thrones, | a | fa | ntasy | epic | by | G | eorge R.R. Martin.
Entropy + Monotonicity: D | aenerys Targar | yen | is | in | Game | of Thrones | , | a | fantasy | epic | by | George R.R. Martin | .
Space: Daenerys | Targaryen | is | in | Game | of | Thrones, | a | fantasy | epic | by | George | R. | R. | Martin.
CNN: Daenerys | Targaryen | is | in | Game | of | Thrones, | a | fantasy | epic | by | George | R.R. Martin.

Entropy-Based Dynamic Patching

How It Works

Unlike static chunking, BLT decides patch boundaries on the fly based on the *information content* of the data. It uses a separately trained lightweight **entropy model** that looks at the context and estimates the uncertainty (entropy) of the next byte.

- Low entropy (predictable) → extend current patch
- High entropy (uncertain) → **start a new patch**

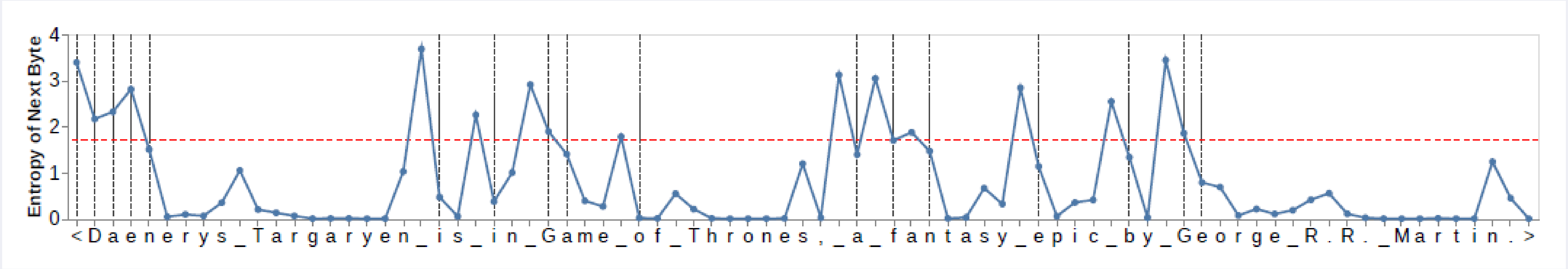
BLT tries to create patches such that each patch has a relatively uniform level of uncertainty/information.

Benefits

"Hard" parts of text (e.g. the first letter of a new word, an unpredictable token, a language/script change, etc.) result in shorter patches (more focus), whereas "easy" stretches (like finishing a common word or a sequence of repeated characters) can be folded into longer patches.

This entropy-guided grouping is dynamic and data-dependent, not fixed by any predetermined rule or vocabulary. It allows the model to **allocate compute adaptively**, focusing its power where needed and skipping ahead when possible.

The entropy of each byte in "Daenerys Targeryen is in Game of Thrones, a fantasy epic by George R.R. Martin."

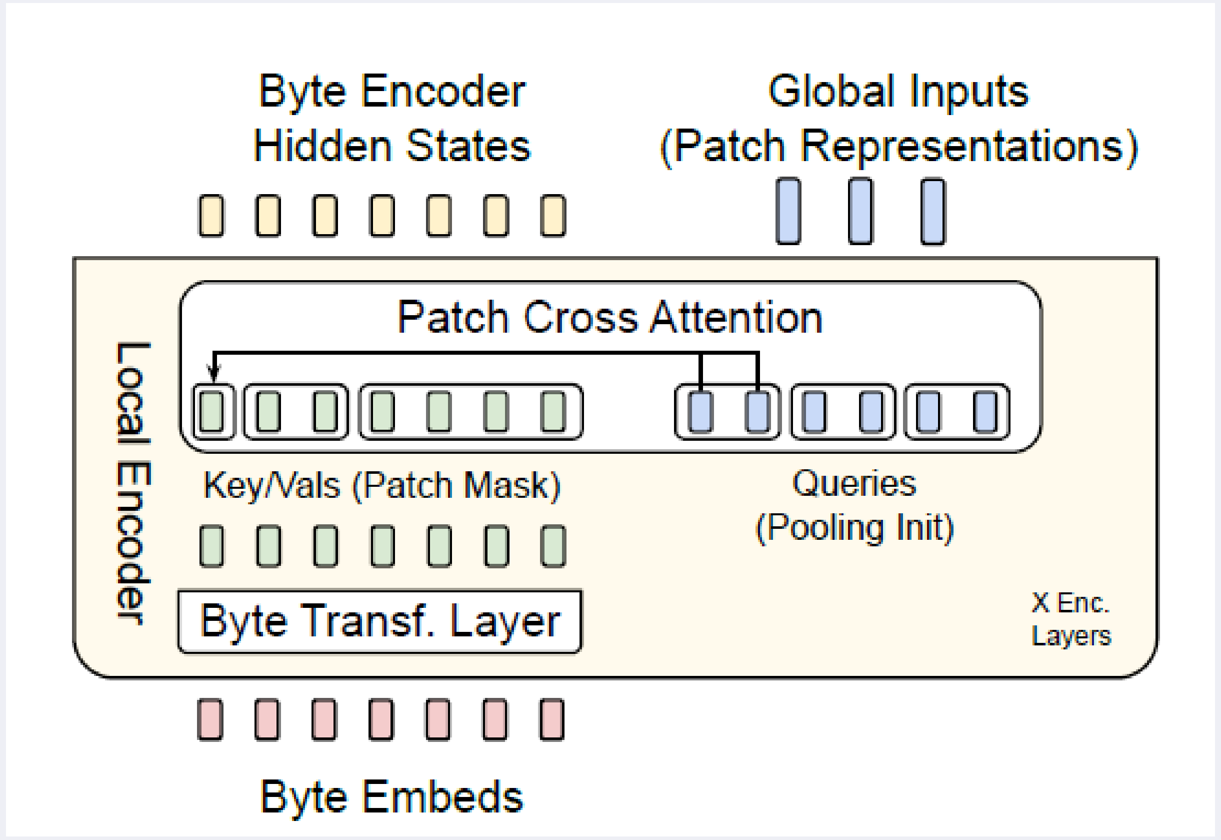


Local Encoder: Creating Robust Representations

The Local Encoder Model is a lightweight transformer-based model with significantly fewer layers than the global model. Its main role is to efficiently map input bytes into expressive patch representations.

A primary departure from the transformer architecture is the addition of a cross-attention layer after each transformer layer, whose function is to pool byte representations into patch representations.

The transformer layers use a local block causal attention mask; each byte attends to a fixed window of preceding bytes that in general can cross the dynamic patch boundaries but cannot cross document boundaries.



Encoder Hash n-gram Embeddings

Creating Expressive Representations

A key component in creating robust, expressive representations at each step is to incorporate information about the preceding bytes. In BLT, this is achieved by modeling both the byte individually and as part of a byte n-gram.

$$g_{i,n} = \{b_{i-n+1}, \ldots, b_i\} \tag{2}$$

for each byte position i and n from three to eight.[4]

Hash Function Approach

BLT introduces hash **n-gram embeddings** that map all byte n-grams via a hash function to an index in an embedding table with a fixed size, for each size n ∈ {3, 4, 5, 6, 7, 8}. The resulting embedding is then added to the embedding of the byte before being normalized and passed as input to the local encoder model, creating a rich representation that captures multi-byte patterns.

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

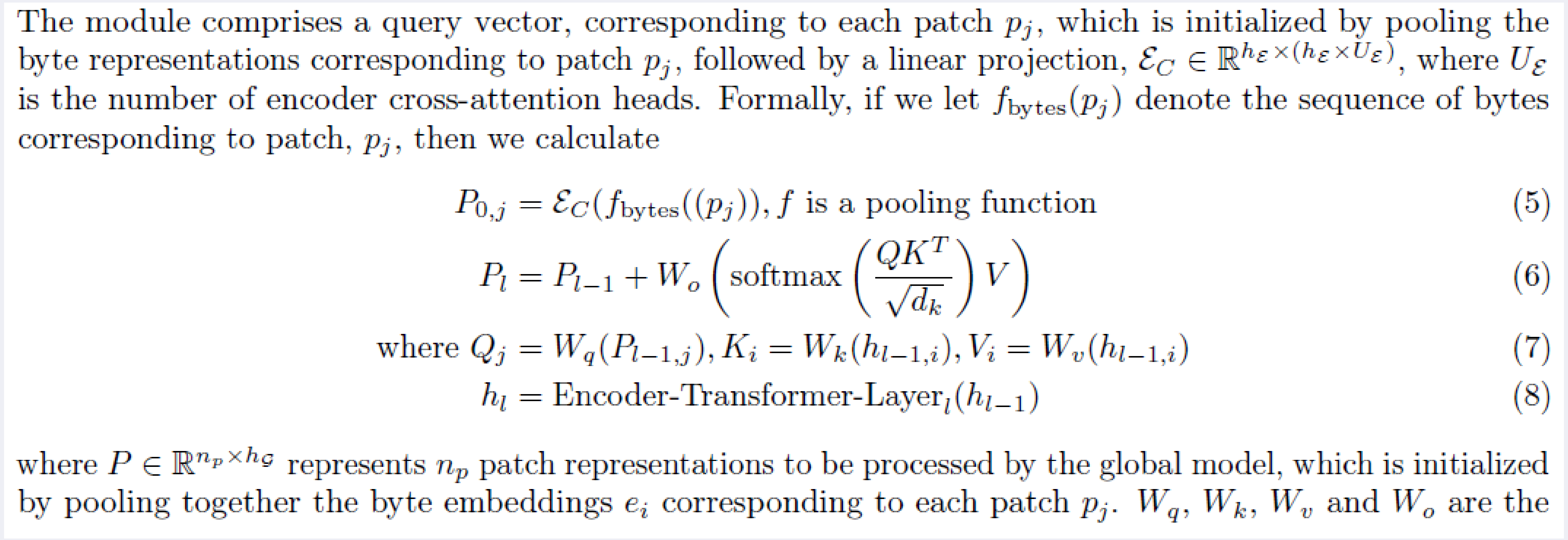
$$\text{where, } \text{Hash}(g_{i,n}) = \text{RollPolyHash}(g_{i,n}) \% |E_n^{hash}| \tag{4}$$

This approach allows the model to efficiently capture n-gram patterns without requiring an exponentially large vocabulary, as would be needed with traditional tokenization approaches.

UNIVERSITY OF TORONTO

Encoder Multi-Headed Cross-Attention

BLT closely follows the input cross-attention module of the Perceiver architecture, with the main difference being that latent representations correspond to **variable patch representations** as opposed to a fixed set of latent representations, and **only attend to the bytes that make up the respective patch**.



The module comprises a query vector, corresponding to each patch p_j, which is initialized by pooling the byte representations corresponding to patch p_j, followed by a linear projection, $\mathcal{E}_C \in \mathbb{R}^{h_\mathcal{E} \times (h_\mathcal{E} \times U_\mathcal{E})}$, where $U_\mathcal{E}$ is the number of encoder cross-attention heads. Formally, if we let $f_{\text{bytes}}(p_j)$ denote the sequence of bytes corresponding to patch, p_j, then we calculate

$$P_{0,j} = \mathcal{E}_C(f_{\text{bytes}}((p_j)), f \text{ is a pooling function} \tag{5}$$

$$P_l = P_{l-1} + W_o \left(\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \right) \tag{6}$$

$$\text{where } Q_j = W_q(P_{l-1,j}), K_i = W_k(h_{l-1,i}), V_i = W_v(h_{l-1,i}) \tag{7}$$

$$h_l = \text{Encoder-Transformer-Layer}_l(h_{l-1}) \tag{8}$$

where $P \in \mathbb{R}^{n_p \times h_\mathcal{G}}$ represents n_p patch representations to be processed by the global model, which is initialized by pooling together the byte embeddings e_i corresponding to each patch p_j. W_q, W_k, W_v and W_o are the

The module comprises a query vector, corresponding to each patch, which is initialized by pooling the byte representations corresponding to the patch, followed by a linear projection. This cross-attention mechanism allows the model to effectively aggregate information from variable-length byte sequences into fixed-dimension patch representations.

Local Decoder: From Patches Back to Bytes

Input Processing

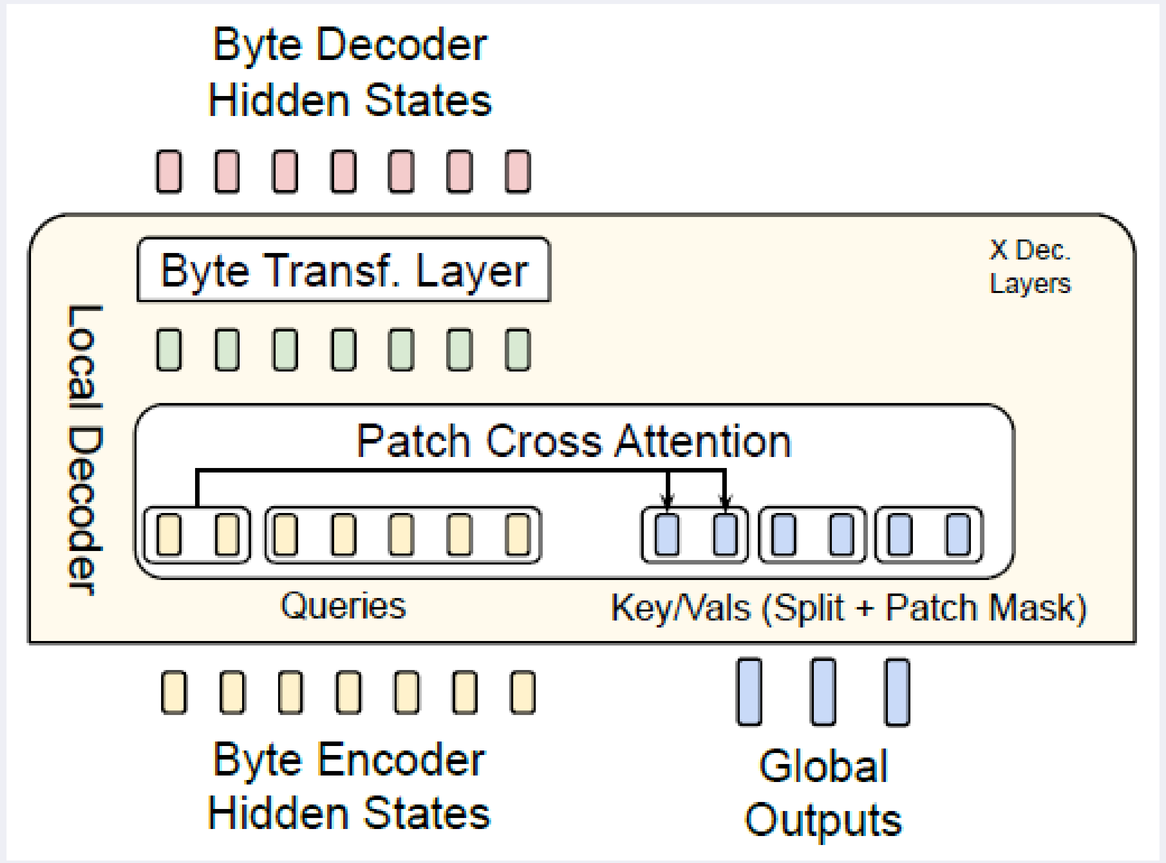
Takes global patch representations and previously decoded bytes as input

Cross-Attention

Applies cross-attention where byte representations are queries and patch representations are keys/values

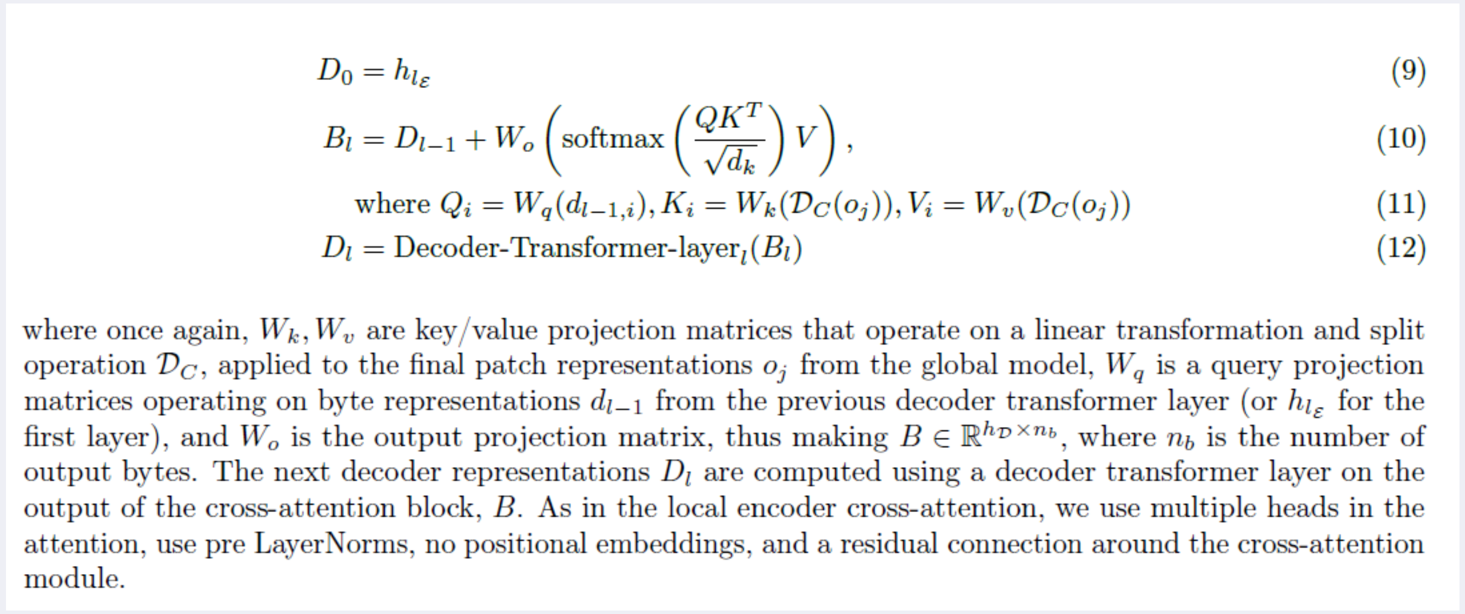
Transformer Layers

Processes the resulting byte sequence through transformer layers



Similar to the local encoder, the local decoder is a lightweight transformer-based model with significantly fewer layers than the global model. It decodes a sequence of global patch representations into raw bytes, predicting a sequence of raw bytes as a function of previously decoded bytes.

Cross-Attention Mechanism



$$D_0 = h_{l_\mathcal{E}} \tag{9}$$

$$B_l = D_{l-1} + W_o \left(\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \right), \tag{10}$$

$$\text{where } Q_i = W_q(d_{l-1,i}), K_i = W_k(\mathcal{D}_C(o_j)), V_i = W_v(\mathcal{D}_C(o_j)) \tag{11}$$

$$D_l = \text{Decoder-Transformer-layer}_l(B_l) \tag{12}$$

where once again, W_k, W_v are key/value projection matrices that operate on a linear transformation and split operation $\mathcal{D}_C$, applied to the final patch representations o_j from the global model, W_q is a query projection matrices operating on byte representations d_{l-1} from the previous decoder transformer layer (or $h_{l_\mathcal{E}}$ for the first layer), and W_o is the output projection matrix, thus making $B \in \mathbb{R}^{h_\mathcal{D} \times n_b}$, where n_b is the number of output bytes. The next decoder representations D_l are computed using a decoder transformer layer on the output of the cross-attention block, B. As in the local encoder cross-attention, we use multiple heads in the attention, use pre LayerNorms, no positional embeddings, and a residual connection around the cross-attention module.

Decoder Cross-Attention

In the decoder, the roles are reversed: byte representations are the queries, and patch representations are the keys and values.

This allows the model to decode patch representations back into bytes, completing the end-to-end byte-level processing pipeline.

UNIVERSITY OF TORONTO

Differences from Earlier Byte-Level Approaches

Feature	MegaByte	BLT
Patch Sizing	Fixed length	Dynamic, entropy-driven
Input Processing	Concatenation of byte vectors	Learned encoding transformer
Local Modules	Decoder only	Encoder and decoder
Attention Mechanisms	Standard	Enhanced with cross-attention
Memory	None	Byte-sequence memory
Performance at Scale	Lags behind token models	Matches token models

These innovations collectively allow BLT to *match* the modeling power of token-based transformers – something static patch models struggled with beyond small scales.

Experimental Setup

Pre-training Datasets

- Llama 2 dataset: 2 trillion tokens from various public sources
- BLT-1T: A new dataset with 1 trillion tokens from public sources, including a subset of Datacomp-LM

Neither dataset includes any data from Meta products or services.

Entropy Model

A byte-level language model trained on the same training distribution as the full BLT model, with **100M parameters**, 14 layers, and a hidden dimensionality of 512.

Context Length Equalization

To maintain the same average context length and avoid giving larger patch sizes unfair advantage, the number of bytes in each batch remains constant in expectation.

The experiments were carefully designed to compare BLT with tokenization-based models with particular attention to not give BLT any advantages from possibly using longer sequence contexts.

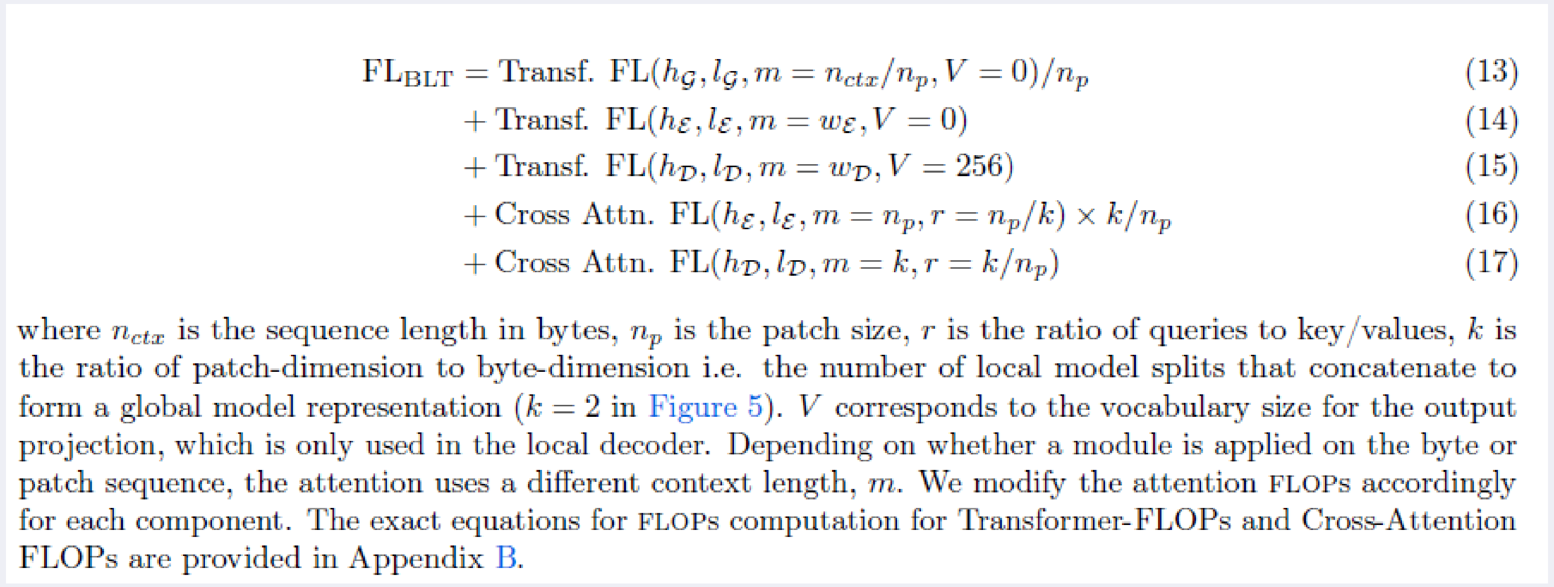
UNIVERSITY OF TORONTO

Experimental Setup

FLOPs Estimation

To compute flops per byte for BLT models, the flops for the local encoder transformer, the global latent transformer, and the local decoder transformer are added together with the cross-attention blocks in the encoder and the decoder.

A notable difference from standard approaches is that the input embedding layer is implemented as an efficient lookup instead of a dense matrix multiplication, therefore becoming a 0-flop operation.

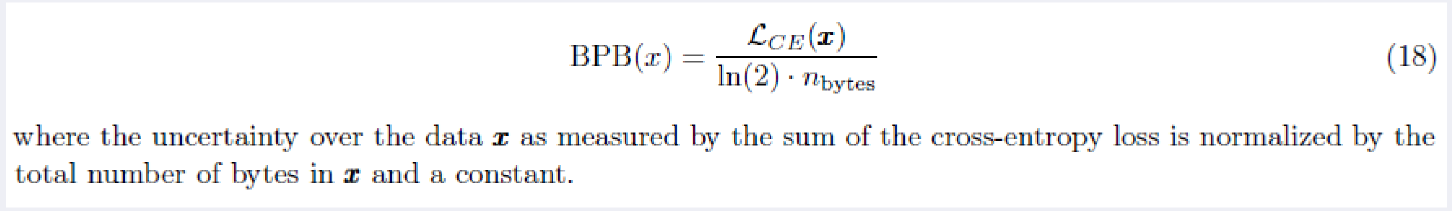


$$
\begin{aligned}
\mathrm{FL}_{\mathrm{BLT}} &= \text{Transf. FL}(h_{\mathcal{G}}, l_{\mathcal{G}}, m = n_{ctx}/n_p, V = 0)/n_p && (13)\\
&+ \text{Transf. FL}(h_{\mathcal{E}}, l_{\mathcal{E}}, m = w_{\mathcal{E}}, V = 0) && (14)\\
&+ \text{Transf. FL}(h_{\mathcal{D}}, l_{\mathcal{D}}, m = w_{\mathcal{D}}, V = 256) && (15)\\
&+ \text{Cross Attn. FL}(h_{\mathcal{E}}, l_{\mathcal{E}}, m = n_p, r = n_p/k) \times k/n_p && (16)\\
&+ \text{Cross Attn. FL}(h_{\mathcal{D}}, l_{\mathcal{D}}, m = k, r = k/n_p) && (17)
\end{aligned}
$$

where n_{ctx} is the sequence length in bytes, n_p is the patch size, r is the ratio of queries to key/values, k is the ratio of patch-dimension to byte-dimension i.e. the number of local model splits that concatenate to form a global model representation ($k = 2$ in Figure 5). V corresponds to the vocabulary size for the output projection, which is only used in the local decoder. Depending on whether a module is applied on the byte or patch sequence, the attention uses a different context length, m. We modify the attention FLOPs accordingly for each component. The exact equations for FLOPs computation for Transformer-FLOPs and Cross-Attention FLOPs are provided in Appendix B.

Bits-Per-Byte (BPB) Estimation

When comparing byte and token-level models, BPB is used as a **tokenizer-independent version of perplexity**:



$$\mathrm{BPB}(x) = \frac{\mathcal{L}_{CE}(\boldsymbol{x})}{\ln(2) \cdot n_{\text{bytes}}} \qquad (18)$$

where the uncertainty over the data $\boldsymbol{x}$ as measured by the sum of the cross-entropy loss is normalized by the total number of bytes in $\boldsymbol{x}$ and a constant.

This allows for fair comparison between models with different tokenization schemes.

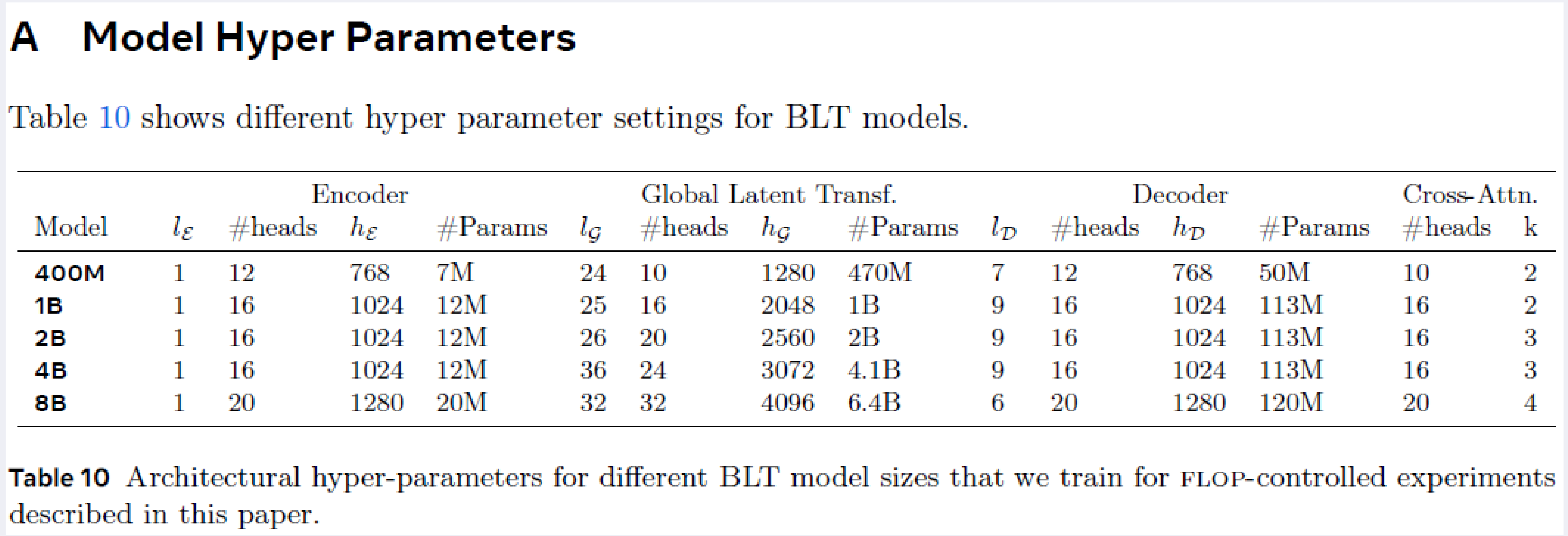
Experimental Setup

Transformer Architecture Hyperparameters

For all transformer blocks in BLT, the architecture **largely follows that of Llama 3**, using the SwiGLU activation function in feed-forward layers, rotary positional embeddings (RoPE) with θ = 500000 only in self-attention layers, and RMSNorm for layer normalization. Flash attention is used for all self-attention layers with fixed-standard attention masks, and Flex Attention for cross-attention layers with dynamic patch-dependent masks.

BLT-Specific Hyperparameters

The authors train BLT models ranging from 400M to 8B parameters. These models adopt a fixed learning rate of $4{\times}10^{-4}$ with AdamW, a 2000-step linear warm-up, and a cosine decay schedule to zero.



A Model Hyper Parameters

Table 10 shows different hyper parameter settings for BLT models.

	Encoder				Global Latent Transf.				Decoder				Cross-Attn.	
Model	$l_{\mathcal{E}}$	#heads	$h_{\mathcal{E}}$	#Params	$l_{\mathcal{G}}$	#heads	$h_{\mathcal{G}}$	#Params	$l_{\mathcal{D}}$	#heads	$h_{\mathcal{D}}$	#Params	#heads	k
400M	1	12	768	7M	24	10	1280	470M	7	12	768	50M	10	2
1B	1	16	1024	12M	25	16	2048	1B	9	16	1024	113M	16	2
2B	1	16	1024	12M	26	20	2560	2B	9	16	1024	113M	16	3
4B	1	16	1024	12M	36	24	3072	4.1B	9	16	1024	113M	16	3
8B	1	20	1280	20M	32	32	4096	6.4B	6	20	1280	120M	20	4

Table 10 Architectural hyper-parameters for different BLT model sizes that we train for FLOP-controlled experiments described in this paper.

UNIVERSITY OF TORONTO

Scaling Trends: Parameter Matched Compute Optimal

Using the Llama 2 dataset, various compute-optimal BPE and BLT models were trained across four different sizes, ranging from 1B to 8B parameters. The BPE models were trained using the optimal ratio of model parameters to training data, as determined by Llama 3.

BLT models either match or outperform their BPE counterparts, and this trend holds as model size and flops scale. To the best of our knowledge, **BLT is the first byte-level Transformer architecture to achieve matching scaling trends with BPE-based models at compute optimal regimes**.

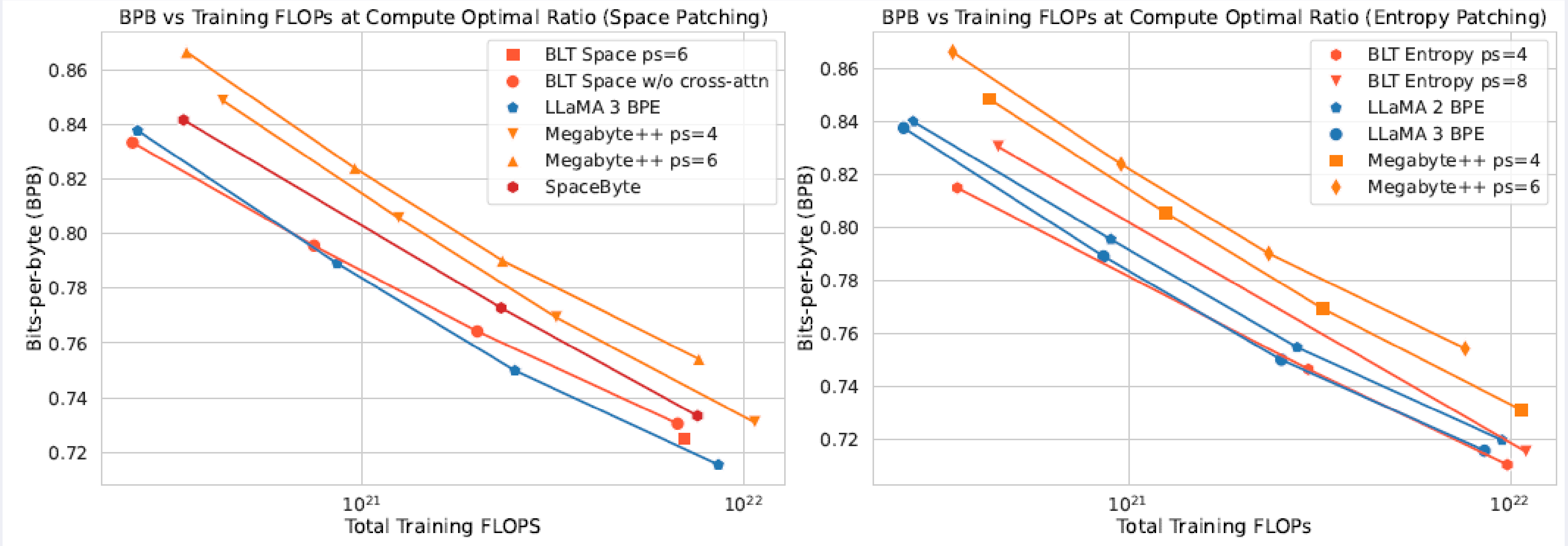
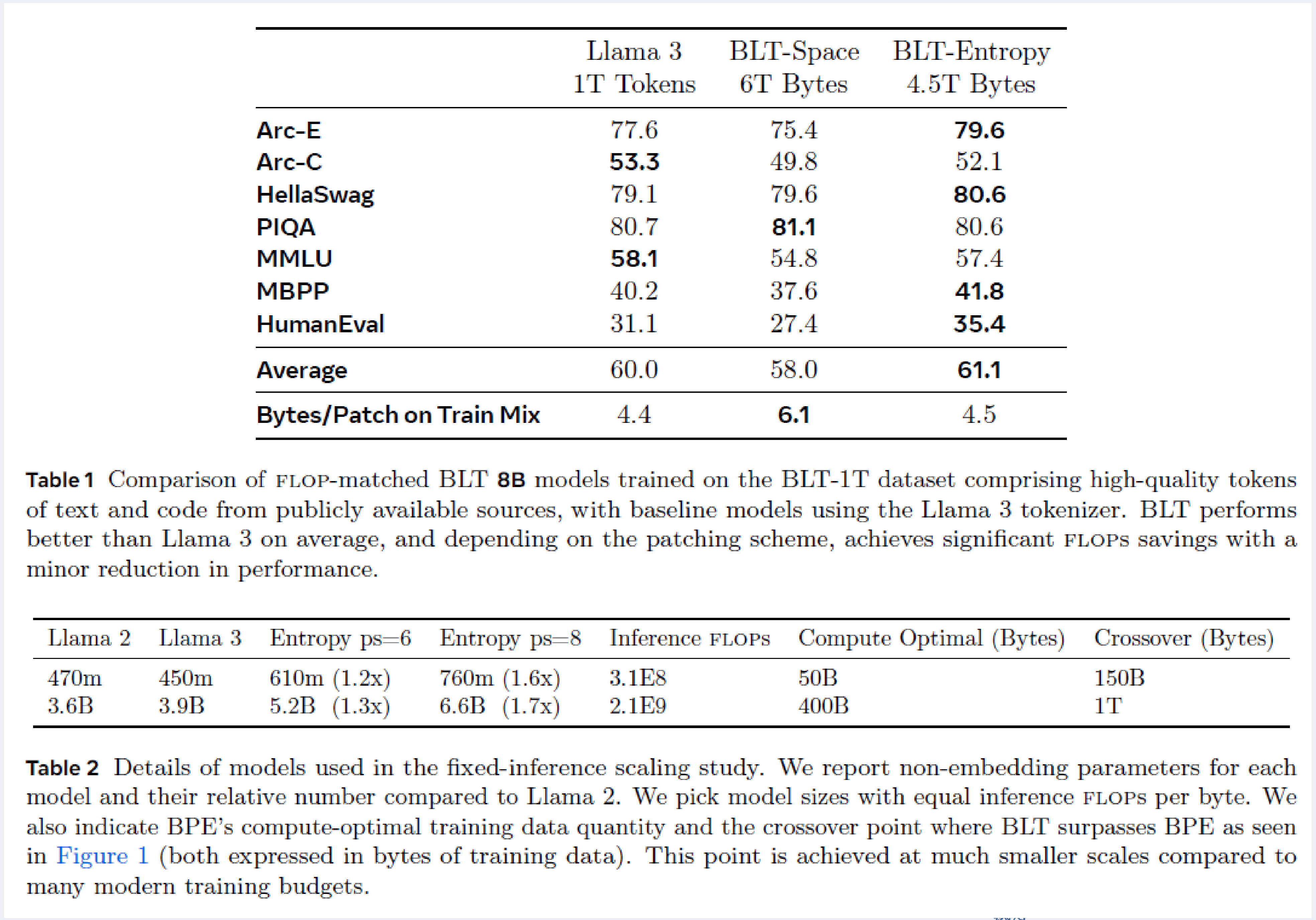


Figure 6 Scaling trends for BLT models with different architectural choices, as well as for baseline BPE token-based models. We train models at multiple scales from 1B up to 8B parameters for the optimal number of tokens as computed by Dubey et al. (2024) and report bits-per-byte on a sample from the training distribution. BLT models perform on par with state-of-the-art tokenizer-based models such as Llama 3, at scale. PS denotes patch size. We illustrate separate architecture improvements on space-patching (**left**) and combine them with dynamic patching (**right**).

UNIVERSITY OF TORONTO

Scaling Trends: Beyond Compute Optimal Task Eval

The authors train an 8B-parameter BLT model beyond its compute-optimal ratio on a high-quality 1T-token dataset and evaluate it against token-based Llama 3 on reasoning and coding tasks (ARC, HellaSwag, PIQA, MMLU, MBPP, HumanEval). The **BLT-Entropy variant matches or surpasses Llama 3 in four out of seven tasks, indicating that patch-based byte modeling can outperform fixed-vocabulary tokenization at scale.** Meanwhile, BLT-Space—though less accurate—reduces inference costs by grouping more bytes into each patch.



	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.

UNIVERSITY OF TORONTO

Scaling Trends: Patches Scale Better Than Tokens

With BLT models, it's possible to **simultaneously increase model size and patch size** while maintaining the same training and inference flop budget and keeping the amount of training data constant. This is a **unique feature of patch-based models** which break free of the efficiency tradeoffs of fixed-vocabulary token-based models.

Longer patch sizes save compute, which can be reallocated to grow the size of the global latent transformer, because it is run less often. Fixed inference scaling studies show that **BLT models achieve better scaling trends** than tokenization-based architectures for both inference flop classes tested.

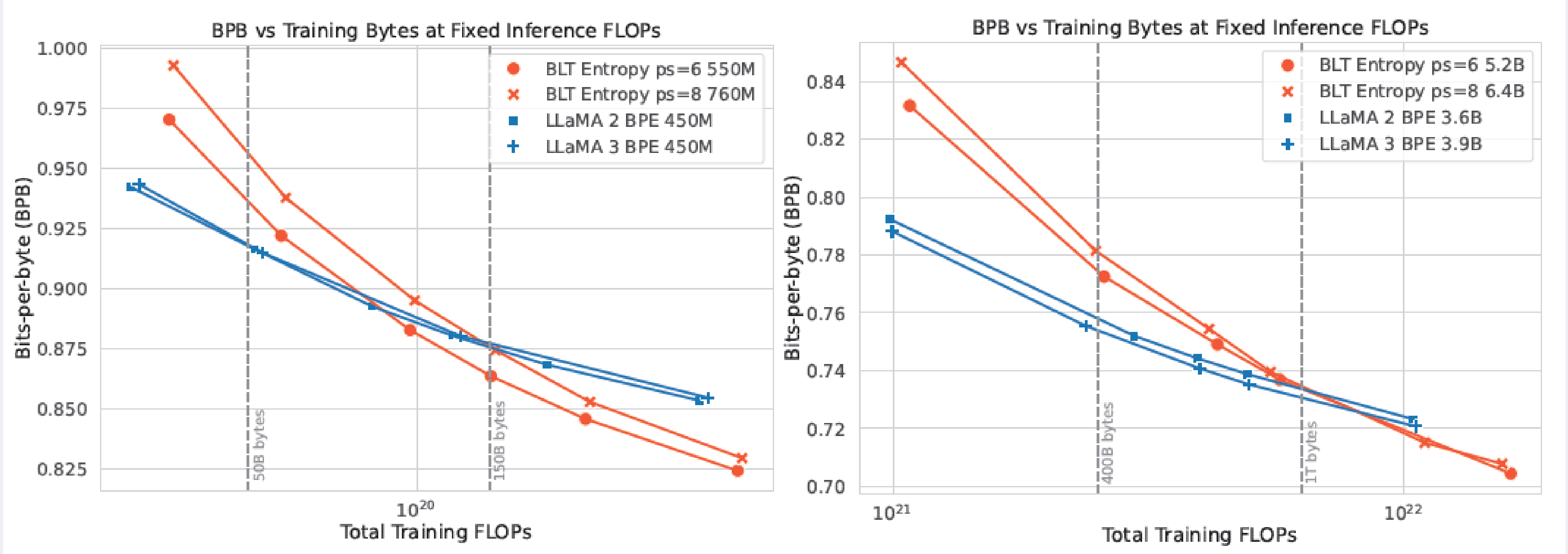


Figure 1 Scaling trends for fixed inference FLOP models (fully) trained with varying training budgets. In token-based models, a fixed inference budget determines the model size. In contrast, the BLT architecture provides a new scaling axis allowing simultaneous increases in model and patch size while 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.

UNIVERSITY OF TORONTO

Byte Modeling Improves Robustness: Character-Level Tasks

Noise Data

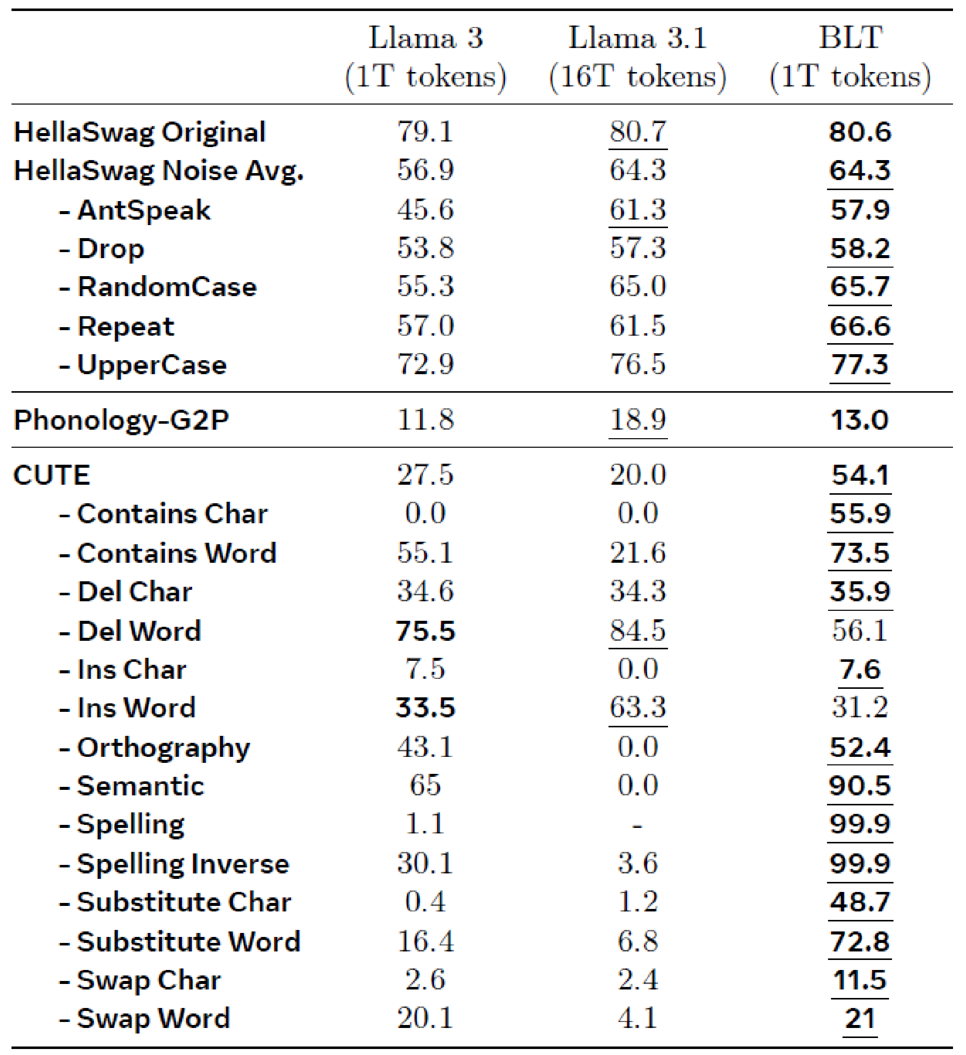
The authors introduce random casing, repeated characters, and uppercase transformations to standard classification tasks (like HellaSwag) to simulate real-world “noisy” text. BLT remains robust under such distortions, **outperforming a Llama 3 baseline** by clear margins. Its byte-level processing preserves orthographic nuances that typical tokenizers fail to capture.

Phonology

To test fine-grained letter-sound knowledge, they evaluate a grapheme-to-phoneme (G2P) task. BLT exhibits **stronger accuracy than token-based models**, reflecting its direct character access. This detailed orthographic modeling is beneficial in languages or tasks requiring precise letter-wise reasoning.

CUTE

CUTE requires local character edits—like inserting symbols or swapping letters—where subword-based models often struggle. BLT excels in these cases because it **treats every character as a first-class input**. Its success on spelling manipulations and orthographic edits underscores the benefits of a purely byte-level approach.



	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.

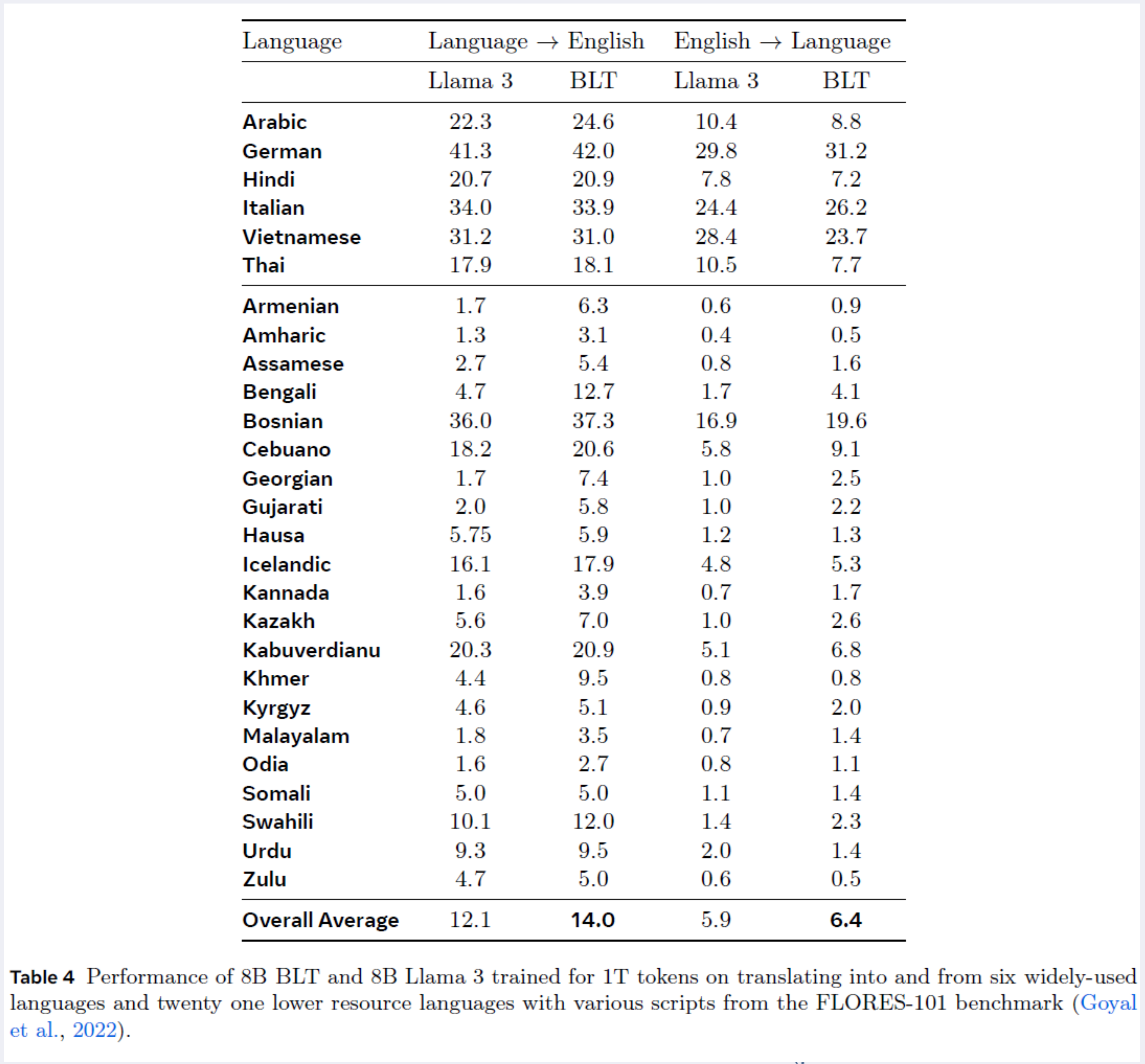
UNIVERSITY OF TORONTO

Byte Modeling Improves Robustness:

Character-Level Tasks

Low-Resource Machine Translation

The authors assess BLT on a suite of lower-resource language pairs from the FLORES-101 benchmark. In many cases, **BLT achieves higher BLEU scores than a comparable Llama 3 baseline**, especially for languages with complex scripts or limited training data. By avoiding a fixed subword vocabulary, BLT's byte-level modeling yields more flexible, granular representations, resulting in better coverage and improved translation quality for underrepresented languages.

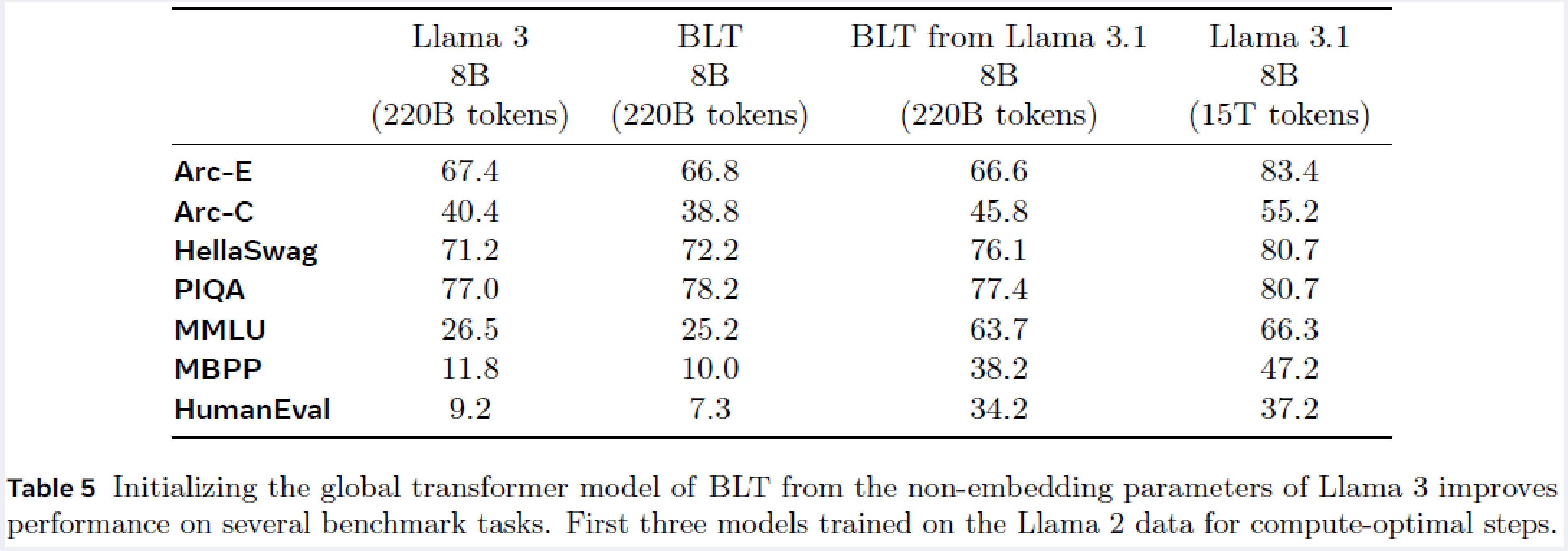


Language	Language → English		English → Language	
	Llama 3	BLT	Llama 3	BLT
Arabic	22.3	24.6	10.4	8.8
German	41.3	42.0	29.8	31.2
Hindi	20.7	20.9	7.8	7.2
Italian	34.0	33.9	24.4	26.2
Vietnamese	31.2	31.0	28.4	23.7
Thai	17.9	18.1	10.5	7.7
Armenian	1.7	6.3	0.6	0.9
Amharic	1.3	3.1	0.4	0.5
Assamese	2.7	5.4	0.8	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).

Byte Modeling Improves Robustness:
Training BLT from Llama 3

The authors explore initializing BLT's large global transformer with the parameters of an existing Llama 3.1 model (excluding embeddings), while the new local encoder and decoder components start from scratch. They then update the global transformer at a lower learning rate than the local components. The results show that this **partial transfer improves BLT's performance** on several tasks (including MMLU), compared to training BLT from scratch, and **reduces the total cost of training a byte-level model**. Nonetheless, further work is needed to fully match or exceed all task performance of the original Llama 3.1 baseline, implying that factors like data mixtures and hyperparameters could play an important role in optimizing this model-initialization procedure.



	Llama 3 8B (220B tokens)	BLT 8B (220B tokens)	BLT from Llama 3.1 8B (220B tokens)	Llama 3.1 8B (15T tokens)
Arc-E	67.4	66.8	66.6	83.4
Arc-C	40.4	38.8	45.8	55.2
HellaSwag	71.2	72.2	76.1	80.7
PIQA	77.0	78.2	77.4	80.7
MMLU	26.5	25.2	63.7	66.3
MBPP	11.8	10.0	38.2	47.2
HumanEval	9.2	7.3	34.2	37.2

Table 5 Initializing the global transformer model of BLT from the non-embedding parameters of Llama 3 improves performance on several benchmark tasks. First three models trained on the Llama 2 data for compute-optimal steps.

UNIVERSITY OF TORONTO

Ablations: Entropy Model Hyperparameters

The authors vary the size and context window of the small byte-level language model that produces next-byte entropy estimates for patching. Model sizes range from as little as ~1 million to 100 million parameters, and the context window (the byte span each entropy prediction sees) varies from 64 to 512.

The findings show that **larger entropy models and longer context windows** do improve BLT's downstream performance—particularly for lower-parameter BLT models—though returns diminish beyond roughly 50 million parameters and a 512-byte window. This suggests that **moderate-sized entropy models can adequately guide patch boundaries** without overly inflating the overall training cost.

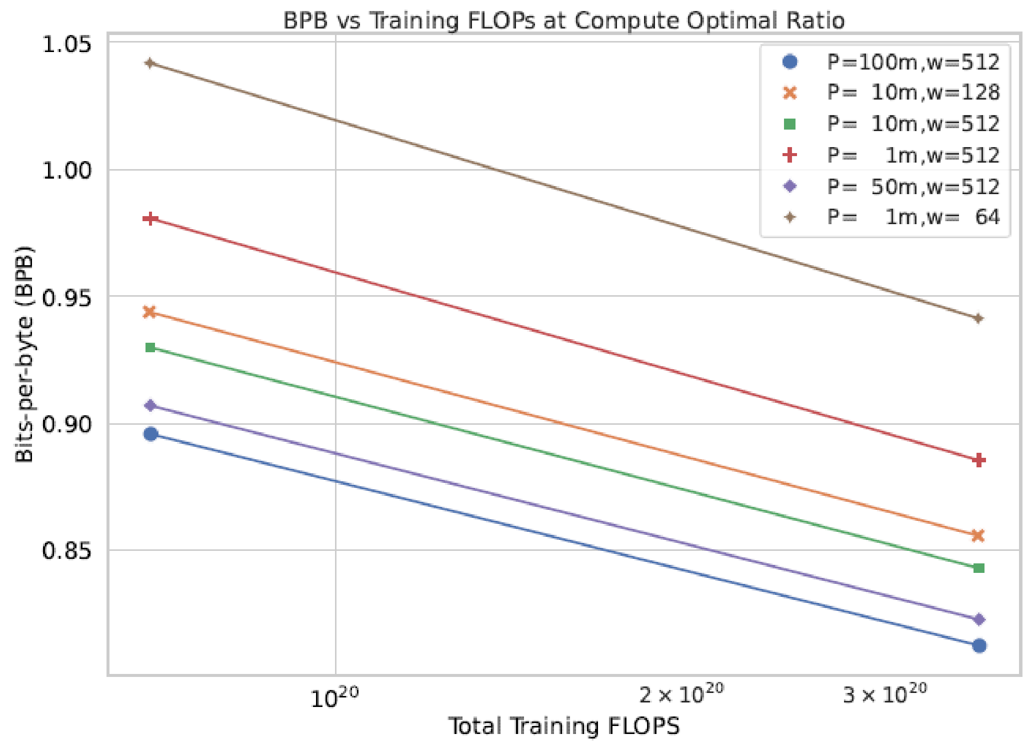


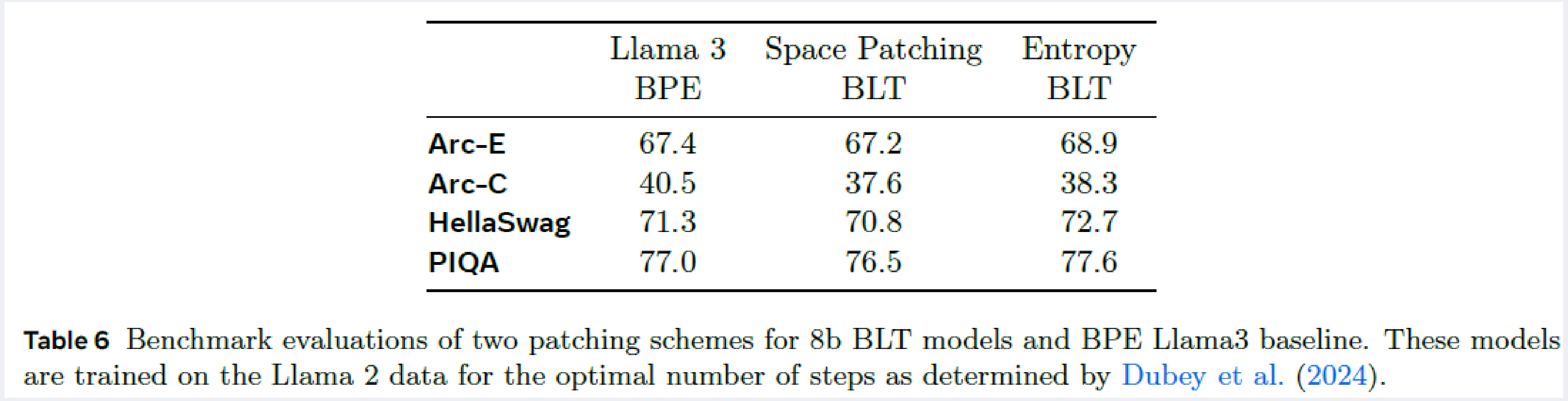
Figure 8 Variation of language modeling performance in bits-per-byte (bpb) with training FLOPs for 400m and 1b BLT models patched with entropy models of different sizes and context windows. Both dimensions improve scaling performance, with diminishing returns beyond 50m parameter entropy models with a context of 512 bytes.

Ablations: Types of Patching

The authors compare four methods for grouping bytes into patches:

Strided Patching: Splitting bytes at a fixed stride (e.g., every 4 or 6 bytes).
Space Patching: Creating new patches whenever a space character appears.
BPE Patching: Using a subword tokenizer (e.g., Llama 3's BPE) to define patch boundaries.
Entropy-Based Patching: Dynamically placing boundaries where next-byte entropy is high.

They ensure each method sees the same average context length—so no approach gains an unfair advantage by processing more (or fewer) bytes overall. Results show **dynamic (entropy) patching generally achieves the best trade-off** between compute savings and performance, with space patching providing a simpler alternative that still outperforms basic strided patching in many cases.



	Llama 3 BPE	Space Patching BLT	Entropy BLT
Arc-E	67.4	67.2	68.9
Arc-C	40.5	37.6	38.3
HellaSwag	71.3	70.8	72.7
PIQA	77.0	76.5	77.6

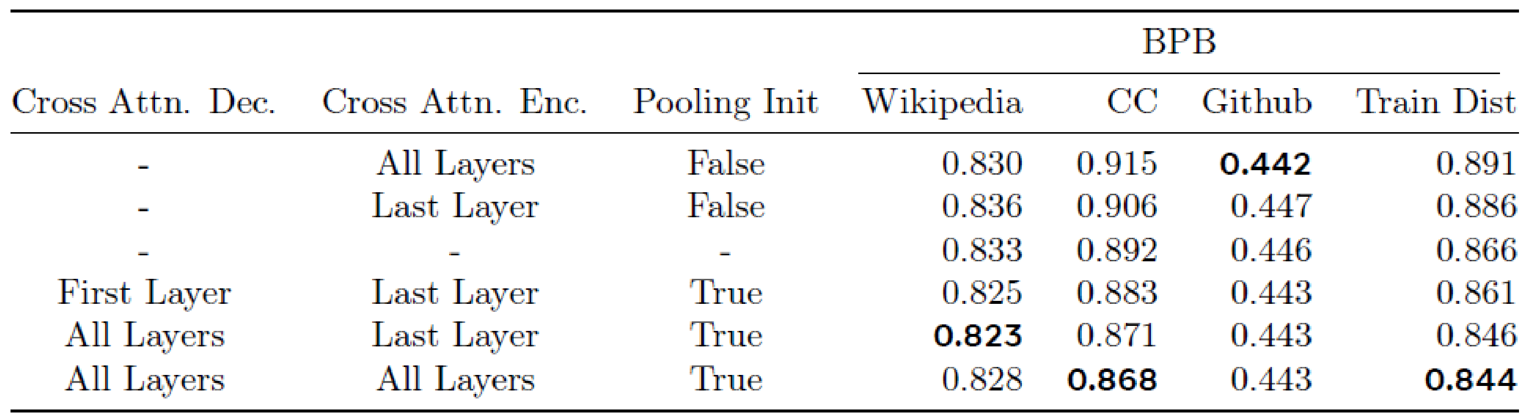
Table 6 Benchmark evaluations of two patching schemes for 8b BLT models and BPE Llama3 baseline. These models are trained on the Llama 2 data for the optimal number of steps as determined by Dubey et al. (2024).

Ablations: Cross-Attention

The authors examine the design of cross-attention blocks both in the local encoder and the local decoder. They test how many layers should include cross-attention and how the initial “query” is formed (e.g., from a pooled byte embedding vs. a single learned vector). Their findings show that:

Cross-Attention in the Decoder consistently yields the largest gains, allowing byte-level representations to attend directly to the global patch embeddings.
Pooling the encoder’s byte representations to initialize patch queries outperforms a single learned query.
Increasing the number of cross-attention layers produces diminishing returns, but at least one or two such layers in both encoder and decoder helps capture key local-global interactions.

While each additional cross-attention layer adds overhead, it improves the alignment between byte embeddings and latent patch representations enough to be worthwhile.



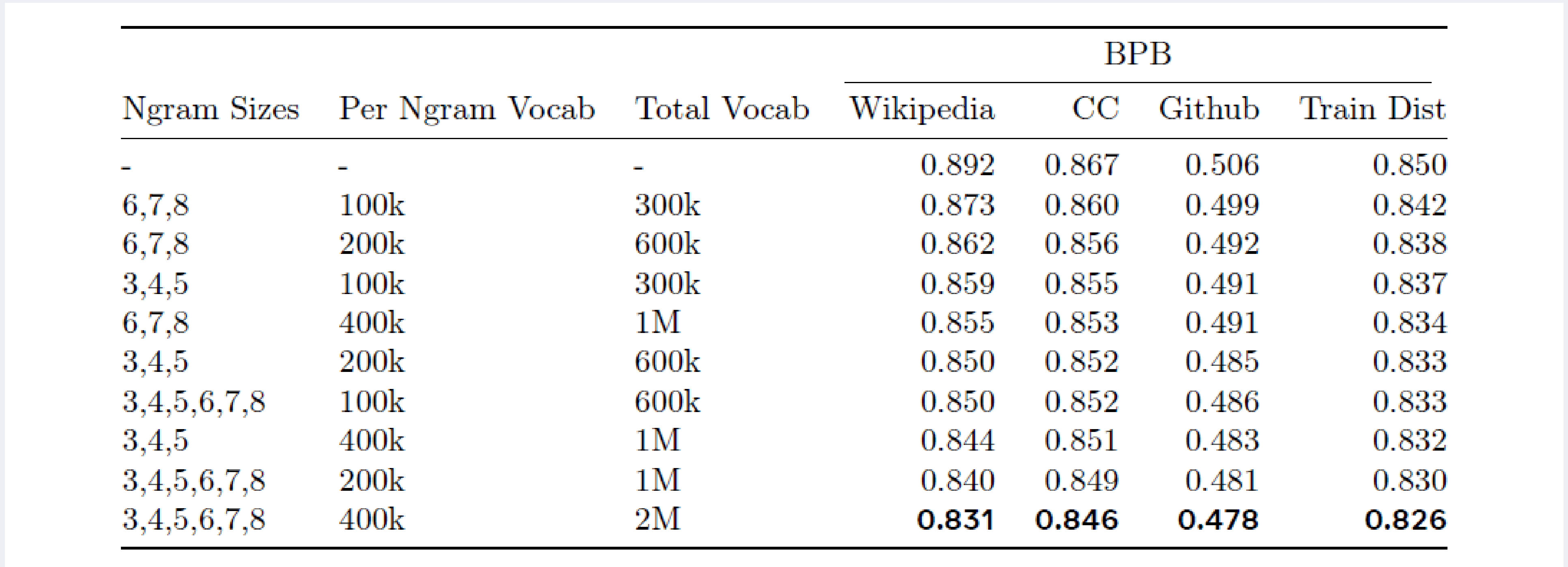
			BPB			
Cross Attn. Dec.	Cross Attn. Enc.	Pooling Init	Wikipedia	CC	Github	Train Dist
-	All Layers	False	0.830	0.915	**0.442**	0.891
-	Last Layer	False	0.836	0.906	0.447	0.886
-	-	-	0.833	0.892	0.446	0.866
First Layer	Last Layer	True	0.825	0.883	0.443	0.861
All Layers	Last Layer	True	**0.823**	0.871	0.443	0.846
All Layers	All Layers	True	0.828	**0.868**	0.443	**0.844**

Table 7 Ablations on the use of Cross Attention for a 1B BLT model trained on 100B bytes. We report bits-per-byte (bpb) on different datasets. We also report bpb on a random sample of the training data (denoted as Train Dist.) The Cross Attn. Enc. and Dec. columns denote which transformer layers the cross-attention block is applied after (or before for the decoder) in the local encoder and decoder respectively.

Ablations: n-gram Hash Embeddings

The authors investigate adding "hash-based n-gram embeddings" (3-grams through 8-grams) on top of each byte's embedding. They explore varying the vocabulary size (e.g., 300 k vs. 2 M hash entries) and different n-gram ranges. Across all tested BLT models, including n-gram embeddings **consistently lowers bits-per-byte error**, most noticeably on structured data like GitHub code or Wikipedia text.

Although larger vocabularies generally yield better results, the returns diminish beyond roughly one to two million entries. Smaller n-gram spans (e.g., 3–5) also tend to boost performance more than only using larger n-grams (6–8). Overall, these hash embeddings appear crucial for capturing local patterns in raw bytes.

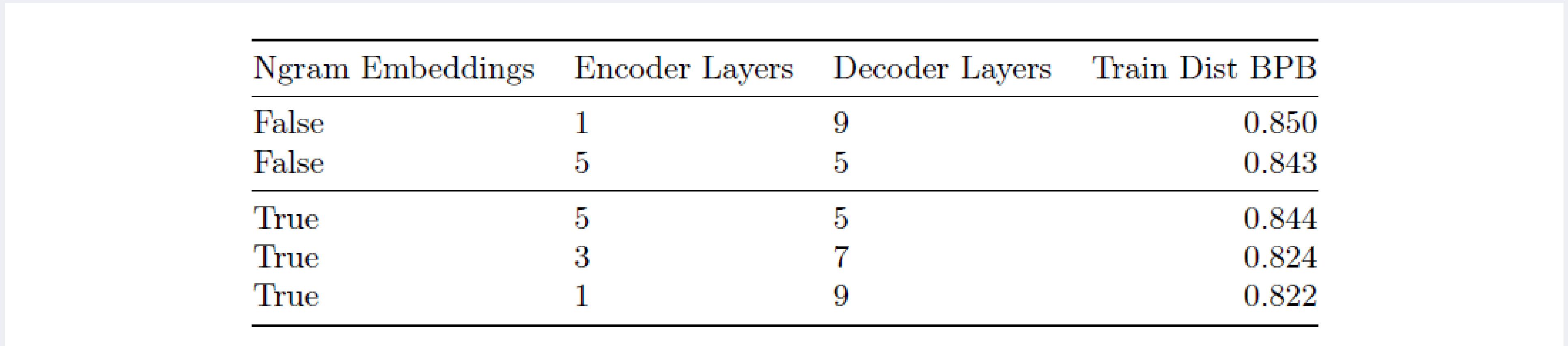


			BPB			
Ngram Sizes	Per Ngram Vocab	Total Vocab	Wikipedia	CC	Github	Train Dist
-	-	-	0.892	0.867	0.506	0.850
6,7,8	100k	300k	0.873	0.860	0.499	0.842
6,7,8	200k	600k	0.862	0.856	0.492	0.838
3,4,5	100k	300k	0.859	0.855	0.491	0.837
6,7,8	400k	1M	0.855	0.853	0.491	0.834
3,4,5	200k	600k	0.850	0.852	0.485	0.833
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.

Ablations: Local Model Hyperparameters

The authors vary how many layers to allocate to the local encoder versus the local decoder, and how large each component's hidden dimension should be. They discover that **a minimal encoder (often a single layer) paired with a deeper decoder (e.g., 9 layers) performs well**, especially when combined with hash-based n-gram embeddings. More encoder layers can help slightly, but the **largest improvements come from a stronger decoder**, which handles byte-to-patch alignment and output generation more effectively.



Ngram Embeddings	Encoder Layers	Decoder Layers	Train Dist BPB
False	1	9	0.850
False	5	5	0.843
True	5	5	0.844
True	3	7	0.824
True	1	9	0.822

Table 9 When paired with hash n-gram embeddings, a light-weight local encoder is sufficient. More layers can then be allocated to the decoder for the same cost.

Limitations and Future Work

Scaling Laws

The optimal ratio of data to parameter sizes may differ for BLT compared to BPE-level transformers. Future work could calculate specific scaling laws for BLT, potentially leading to even more favorable scaling trends.

Implementation Efficiency

Existing transformer libraries are optimized for tokenizer-based architectures. While theoretical flop matched experiments were presented, implementations may not yet be at parity with tokenizer-based models in terms of wall-clock time.

End-to-End Patching

While BLT uses a separately trained entropy model for patching, learning the patching model in an end-to-end fashion could be an interesting direction for future work.

Byte-ifying Existing Models

Further work on "byte-ifying" tokenizer-based models may uncover methods that not only retain the benefits of byte-level processing but also push performance beyond that of these tokenizer-based models without training them from scratch.

UNIVERSITY OF TORONTO

Conclusion and Impact

Performance Parity

BLT achieves performance parity with tokenization-based models at scale, demonstrating the viability of byte-level modeling for large language models.

Inference Efficiency

Dynamic patching enables up to 50% reduction in inference costs while maintaining competitive performance on standard benchmarks.

New Scaling Dimension

BLT unlocks a new dimension for scaling by allowing simultaneous increases in model and patch size within a fixed inference budget.

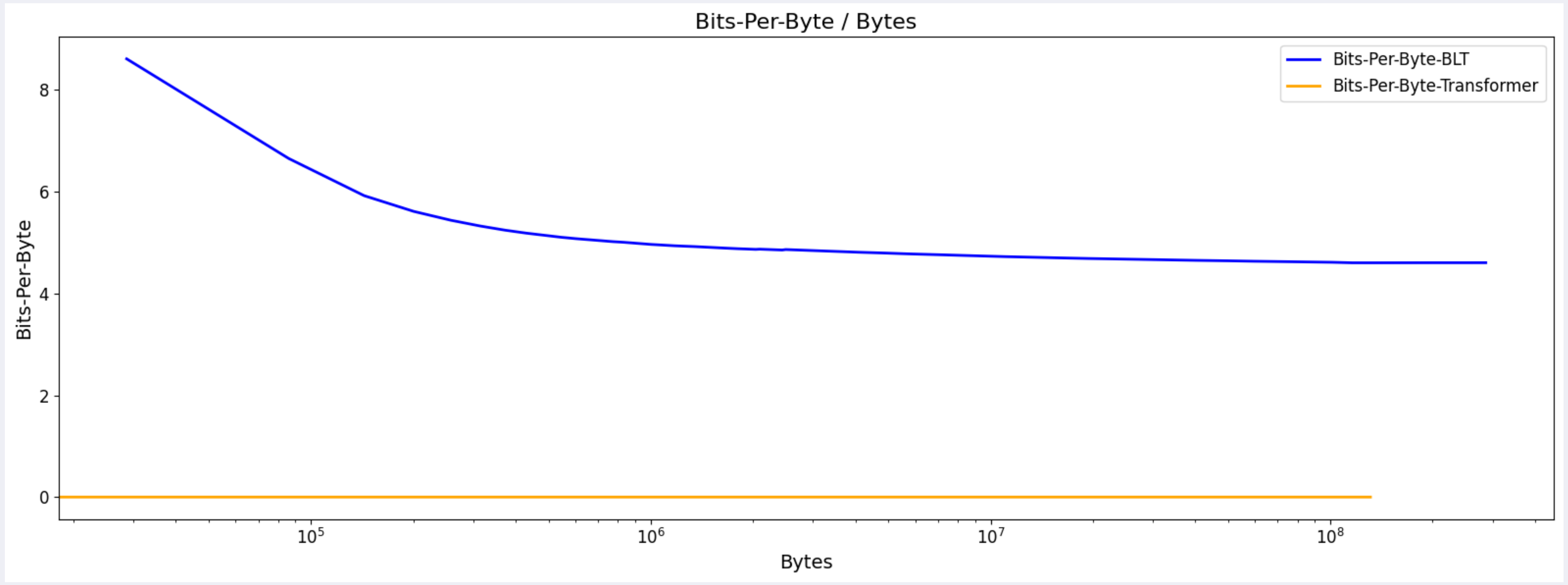
Improved Robustness

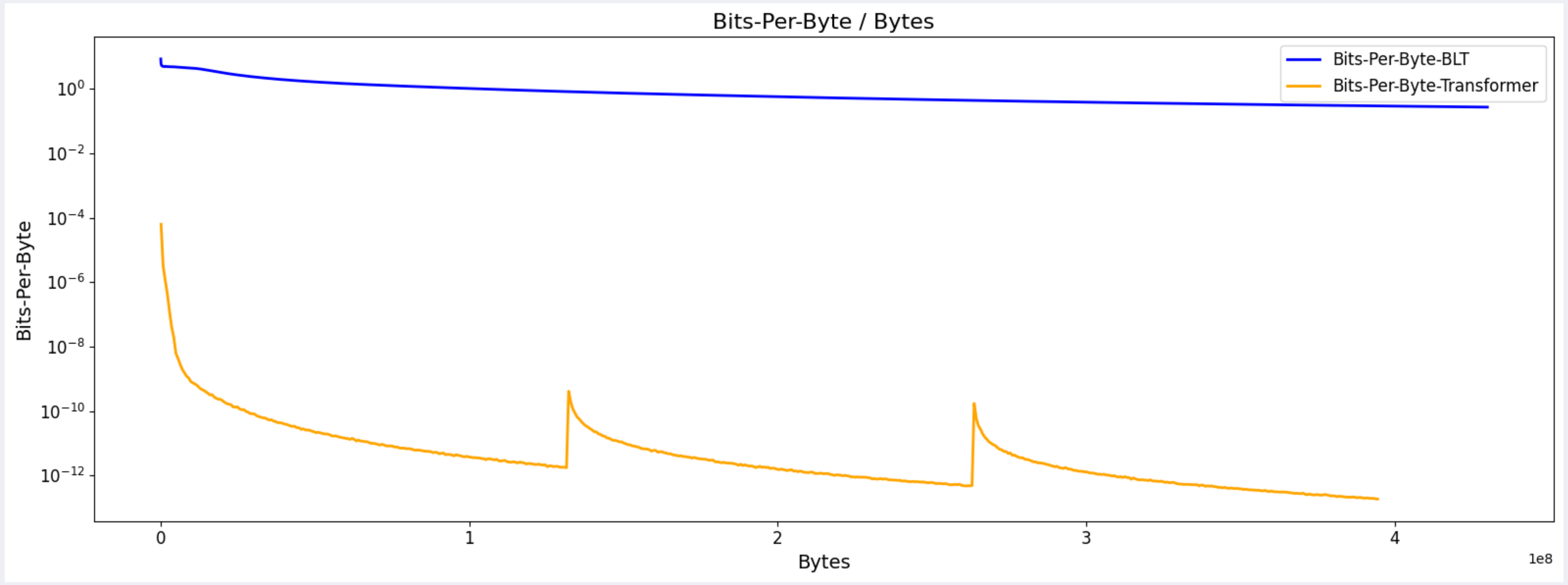
Direct engagement with raw byte data improves the model's handling of noisy inputs, character-level tasks, and multilingual content.

The Byte Latent Transformer represents a significant advancement in language model architecture, challenging the conventional dependency on fixed-vocabulary tokenization. By dynamically allocating compute based on data complexity, BLT offers a more efficient and adaptable framework for language modeling that maintains performance while improving robustness and multilingual capabilities.

Whether 2025 will fully "**say goodbye to tokenization**" remains to be seen, but BLT has certainly laid the conceptual groundwork for a future where NLP models **learn their own text representations on the fly**, leading to more scalable and adaptable language technology.

Code Notebook





UNIVERSITY OF TORONTO