SpaceByte: Towards Deleting Tokenization from Large Language Modeling

Kevin Slagle
Rice University
kevin.slagle@rice.edu

Abstract

Tokenization is widely used in large language models because it significantly improves performance. However, tokenization imposes several disadvantages, such as performance biases, increased adversarial vulnerability, decreased character-level modeling performance, and increased modeling complexity. To address these disadvantages without sacrificing performance, we propose SpaceByte, a novel byte-level decoder architecture that closes the performance gap between byte-level and subword autoregressive language modeling. SpaceByte consists of a byte-level Transformer model, but with extra larger transformer blocks inserted in the middle of the layers. We find that performance is significantly improved by applying these larger blocks only after certain bytes, such as space characters, which typically denote word boundaries. Our experiments show that for a fixed training and inference compute budget, SpaceByte outperforms other byte-level architectures and roughly matches the performance of tokenized Transformer architectures.

1 Introduction

Most language models are trained using tokenization, which partitions text into tokens that typically consist of words or subwords. Tokenization is useful because it significantly decreases the inference and training computational costs of large language models. However, tokenization also imposes several disadvantages, including performance penalties for languages that are priortizes less by the tokenizer [1–3]; increased vulnerability to adversarial attacks [4]; and worse character-level modeling performance [5, 6], and additional model complexity.[1]

Recently, MegaByte [7], MambaByte [6], and more [8–12] have been proposed as new byte-level autoregressive language models that model bytes instead of tokens. (See [13–21] for encoder and encoder-decoder byte-level modeling.) To address the longer context size resulting from modeling bytes instead of tokens, MegaByte uses multiscale modeling [22–24], while MambaByte uses Mamba blocks [25] instead of Transformer blocks. But although MegaByte and MambaByte have been shown to perform better than a standard byte-level Transformer, to our knowledge, no byte-level autoregressive large language model architecture has been shown to match the performance of tokenized models when controlling for compute costs.

In this work, we study the performance of byte-level and subword-level autoregressive models when trained using a fixed compute budget. We measure the performance in terms of the cross entropy (measured in bits-per-byte), which has been shown to be a strong predictor of down-stream performance [26]. In addition to controlling for training compute, we also control for inference compute costs (measured in FLOPs). We find that byte-level Transformer and MegaByte models can require roughly 10 times more training FLOPs to achieve the same performance as a subword-

[1] See also Andrej Karpathy's tweet twitter.com/karpathy/status/1657949234535211009 and video youtube.com/watch?v=zduSFxRajkE&t=6725s on the disadvantages of tokenization.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

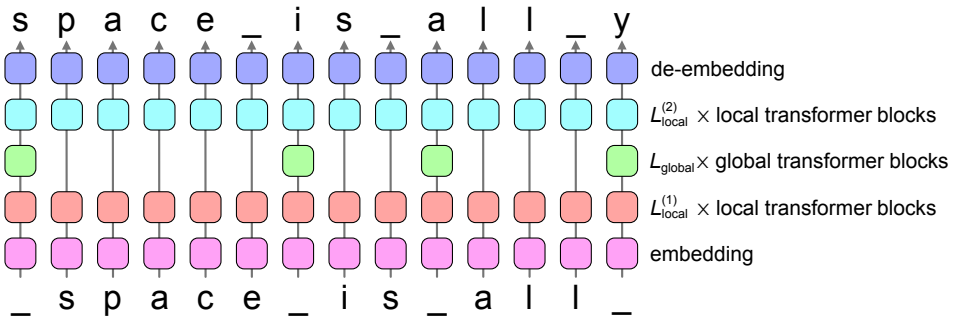


Figure 1: An overview of the SpaceByte architecture. The embedding, local transformer blocks, and de-embedding (i.e. a layer norm and linear) are the standard Transformer decoder layers. SpaceByte modifies the standard transformer by applying "global" transformer blocks only after certain bytes, such as space characters. The intuition is that the first character of a word is typically the hardest to predict; thus this positioning of the global blocks should make the best use of the global blocks (which use a larger model dimension).

level Transformer. To close this substantial performance gap, we propose a new byte-level decoder architecture: SpaceByte.

SpaceByte also utilizes multiscale modeling to improve efficiency by grouping bytes into patches. But unlike MegaByte, which uses a fixed patch size, SpaceByte uses a simple rule to dynamically partition the bytes into patches that are aligned with word and other language boundaries. Our compute-controlled experiments show that this simple modification is crucial for performance, allowing SpaceByte to outperform other byte-level architectures and roughly match the performance of subword Transformers across a variety of text modalities.

Our experiments are performed on datasets consisting of English books, LaTeX formatted arXiv papers, and open-source code. For other data modalities, SpaceByte with our simple patching rule might not be as effective.

2 SpaceByte

The SpaceByte architecture is summarized in Figure 1. In a nutshell, SpaceByte can be thought of as a byte-level Transformer model, but with extra "global" transformer blocks (with a larger model dimension) inserted in the middle, which are only applied a fraction of the time. While the MegaByte architecture applies the global transformer blocks every $P \sim 8$ bytes, we hypothesize that this fixed spacing hinders performance. Our intuition is that the first character of a word is typically significantly harder to predict than the following characters. We therefore expect that performance can be improved by applying the global blocks primarily at word boundaries.

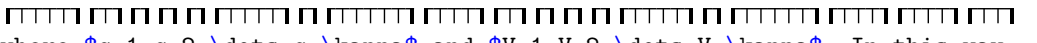
Global Block Insertion Rule In this work, we consider a very simple rule to dynamically decide when to apply the global blocks. We assume that the text bytes are encoded using the UTF-8 encoding. We define a byte to be *spacelike* if the byte does not encode a letter, number, or UTF-8 continuation byte[2]. We apply the global blocks after any spacelike byte that is not preceded by another spacelike byte (and after any BOS token). See Figure 2 for examples.

The most common spacelike byte is the space character. Thus, the global blocks are applied most frequently to predict the first character of a word, which we expect is the hardest character to predict [27] in a given word. With fixed patch size (e.g. as in MegaByte), the global blocks are typically inserted in the middle a word, which we expect is inefficient because predicting the rest of the word could likely be more efficiently accomplished using the local blocks. We define continuation bytes to be spacelike so that languages that do not use spaces between words can still benefit from the global blocks between multi-byte characters (e.g. Chinese characters consists of three bytes in UTF-8).

[2]UTF-8 uses a variable number of bytes to encode a character. English letters or numbers consist of a single byte. Characters that are encoded using multiple bytes are encoded using a leading byte (which is spacelike by our definition) followed by one or more continuation bytes (which are not spacelike).

PG-19:

arXiv:



Github:

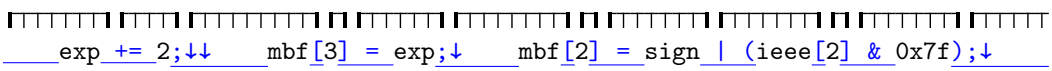


Figure 2: Examples of patch boundaries from datasets that we study. Spacelike bytes are underlined and colored blue. Patches boundaries are drawn above the text. Each patch ends after a spacelike byte that is not preceded by another spacelike byte. Consequently, each patch begins with zero or more spacelike bytes, followed by one or more non-spacelike bytes, and ends with a single spacelike byte. The global blocks predict the first character of each patch. The downward arrow (↓) denotes a newline byte. The left and right quotation characters, (“) and (”) in the PG-19 example, are encoded using three bytes in UTF-8. The first of the three bytes is spacelike, while the later two bytes are UTF-8 continuation bytes, which are not spacelike and are each denoted using a bullet point (•) above.

Although this very simple “spacelike” rule is likely not the optimal rule, we find that it works surprisingly well in practice for English text, LaTeX formatted papers, and code. Nevertheless, a critical future direction is to optimize [28, 14, 9] better rules using data rather than our simple heuristic.

Important Details Since the global blocks are not applied as often as the local transformer blocks, it is advantageous to use a larger model dimension for the global transformer blocks. To increase the dimensions of an activation vector before the global blocks, we simply pad the activation vector with zeros. To decrease the dimension, we truncate the activation vector.

In our experiments, we use a significantly larger context size than the model dimension D_{local} of the local transformer blocks. To prevent the attention mechanism from dominating the compute costs for the local model, we use an attention window [29–31] of length D_{local} for the local transformer blocks. The global blocks use a global attention that attends to all other global blocks.

See Appendix C for pseudocode. Additional details specific to our experiments are provided in Sections 4.1 and 4.2 and Appendix A.

3 Related Work

The most straight-forward consequence of modeling bytes instead of subword tokens is that the length of a sequence typically increases by about a factor of four. This increased sequence length increases the training and inference compute cost for modeling a given long sequence of text for a Transformer due to the quadratic scaling of attention.

MegaByte The MegaByte architecture strives to use multiscale Transformer modeling to lessen these performance issues. In particular, MegaByte groups bytes into patches of a fixed patch size P. Each patch of bytes is vectorized and then fed into a “global” Transformer model. The output of the global model is then fed into a “local” Transformer model that autoregressively outputs byte-level logits. [7]

For a context size of T bytes, MegaByte’s global Transformer model compresses the context into only T/P patches, which can significantly decrease the compute cost for modeling long sequences. Similar to Yu et al. [7], we also find that MegaByte outperforms a standard byte-level Transformer. However, we find that MegaByte’s performance is remarkably close to a stronger byte-level Transformer baseline that simply uses a sliding window attention mechanism [29–31] to increase the context size without increasing the compute costs.

Yu et al. [7] do not compare MegaByte to subword-level Transformer in compute controlled experiments. In our compute controlled experiments, we find that MegaByte's performance significantly lags behind a subword-level Transformer.

Compared to MegaByte, SpaceByte makes the crucial change that patches are dynamically sized to be commensurate with the text, e.g. with word boundaries. We also add an additional local model before the global model (while MegaByte only utilizes a single local model after the global model) to help the model deal with the dynamical patch sizes. We also use significantly longer attention windows for our local models. We find that these changes allow SpaceByte to significantly improve upon the performance of MegaByte and roughly match the performance of subword-level Transformers.

MambaByte The MambaByte architecture [6] takes an alternative approach to avoiding the quadratic compute scaling of the attention mechanism by replacing the Transformer block with a Mamba block [25]. Wang et al. [6] find that their byte-level MambaByte models outperform byte-level Transformer and byte-level MegaByte models. They perform one controlled experiment with a subword model, where they find that MambaByte slightly outperforms Mamba (using tokens) when controlling for the amount of model parameters and training data (which was 14 epochs of the PG-19 dataset). But this experiment was not controlled for compute as MambaByte was trained using roughly four times as much compute than Mamba. We view the Mamba and MambaByte architectures as complementary to our work, as the Mamba block could be integrated into SpaceByte (or MegaByte) in place of Transformer blocks.

Layer Skipping SpaceByte could be though of as a Transformer that employs a novel kind of text-dependent layer skipping [32–38] on the middle layers.

Word Boundary Prior works have shown utility in using word boundaries to partition patches for autoregressive multi-scale byte-level modeling [9, 8, 11] (and also [18] for encoder-decoder modeling). However, these works did not compare autoregressive byte-level models to subword-level models, nor did they identity a patch partitioning rule that could generically be applied to UTF-8 encoded text. Our primary contributions beyond these prior works is to show how to scale word-boundary byte-level modeling to more diverse text modalities while roughly matching the performance of subword-level models in compute-controlled experiments.

Nawrot et al. [9] and Fleshman and Van Durme [11] make use of the Hourglass Transformer architecture [23]. The SpaceByte architecture is similar to the Hourglass Transformer, except SpaceByte uses a simpler technique for shortening and upscaling the activations before and after the global blocks, and SpaceByte uses a sliding window attention [29–31] in the local blocks to improve performance for long context sizes.

4 Experiment Setup

Our experiments compare the performance of our byte-level SpaceByte architecture to subword-level Transformer and byte-level Transformer and MegaByte architectures. To fairly compare the performance between the byte and subword level models, we measure the cross-entropy of the test dataset in terms of bits-per-byte.[3] Given the substantial variation in inference compute costs across the models we study, we also compare their inference compute costs to provide a more comprehensive evaluation. We use FLOPs-per-byte as a simple software and hardware–independent proxy for inference compute costs, which is the number of FLOPs (see Appendix A.1) required to model a byte of text.[4]

Note that by controlling for both total training compute and FLOPs-per-byte, we are also controlling for the amount of training data since $(\text{bytes trained}) = (\text{training FLOPs})/(\text{training FLOPs-per-byte})$. The FLOPs-per-byte during training is equal to three times the FLOPs-per-byte during inference (due to the backwards pass during training).

[3] The bits-per-byte (BPB) is equal to $\text{BPB} = \text{XE}\, N_{\text{tokens}}/(N_{\text{bytes}} \ln 2)$, i.e. the cross entropy per token (XE) times the number of tokens per byte ($N_{\text{tokens}}/N_{\text{bytes}}$) divided by $\ln 2$ (to convert nats to bits). N_{tokens} and N_{bytes} are the number of tokens and bytes in the dataset, respectively. For byte-level models, $N_{\text{tokens}} = N_{\text{bytes}}$ since bytes are used in place of tokens.

[4] We note that in order for memory bandwidth to not be a bottleneck during inference, the batch size must be sufficiently large and e.g. grouped-query attention [39, 40] must be used.

We therefore study the Pareto frontier of lowest bits-per-byte and lowest FLOPs-per-byte. We train all models using a compute-controlled setup, using either 10^{18} or 10^{19} FLOPs. In order to effectively explore this Pareto frontier, we train models using a grid of different model dimensions and numbers of layers, as specified in Appendix B.3.

Datasets Following the MegaByte [7] and MambaByte [6] experiments, we benchmark our models on a diverse range of long-form datasets: PG-19 (English-language books written before 1919) [41]; arXiv (papers from ArXiv written in LaTeX, extracted from the arXiv component of The Pile [42]); and Github (open-source code repositories, extracted from the Github component of The Pile [42]).

4.1 Models

The models we study tend to perform best when the compute cost is roughly evenly split between the attention and feedforward layers. To ensure this, we fix the context size (or attention window) to be equal to the model dimension for every layer. We detail our model setup below.

Notation For all models, we use T to denote the context length, and D to be the model dimension (of the global model for SpaceByte and MegaByte).

For SpaceByte and MegaByte, D_{local} is the dimension of the local model, and T_{global} is the maximum context size for the global model. The patch size P is the number of bytes between global blocks. If the patch size is fixed (which is always the case for MegaByte), we naturally set the context size to be $T = PT_{\text{global}}$.

Below, we describe each of the model architectures that we compare in our experiments.

SpaceByte We fix the global context size and global model dimension to be equal, $T_{\text{global}} = D$, and we set the local attention window W_{local} equal to the local model dimension, $W_{\text{local}} = D_{\text{local}}$. For the PG-19 and arXiv datasets, the average patch size is roughly 6, so we take $T = 6T_{\text{global}}$ for these datasets; for the Github dataset, the average patch size is roughly 8, so we instead take $T = 8T_{\text{global}}$ for the Github dataset.

For simplicity, we fix the number of global transformer blocks L_{global} to be equal to the total number of local blocks, $L^{(1)}_{\text{local}} + L^{(2)}_{\text{local}}$, and we evenly split the number of local blocks before ($L^{(1)}_{\text{local}}$) and after ($L^{(2)}_{\text{local}}$) the global blocks, i.e. we fix $L^{(1)}_{\text{local}} = L^{(2)}_{\text{local}} = \frac{1}{2}L_{\text{global}}$.

SpaceByte (fixed patches) To clearly demonstrate the utility of dynamically aligning patch boundaries in SpaceByte, we also train a simplified version SpaceByte where the patches all have a fixed size. In order to roughly match SpaceByte's average patch size, we take the fixed patch size to be $P = 6$ for all datasets except for the Github dataset, for which we use $P = 8$. We again use $T_{\text{global}} = D$ and $T = PT_{\text{global}}$.

Byte-level Transformer For a simple baseline comparison (following Yu et al. [7]), we train byte-level Transformer models. We take the context size to be equal to the model dimension, $T = D$.

Note that in our setup, a Transformer with model dimension D only sees a context size of D, which is significantly smaller than the context size of PD for SpaceByte (and MegaByte) with patch size P.

Byte-level Transformer (Window Attention) Since a shorter context is a significant disadvantage for long-form datasets, we also compare against a stronger Transformer baseline that uses a sliding window attention [29–31] to efficiently increase the context size without increasing compute costs. We train each window attention enhanced Transformer using a context size $T = PD$ and a sliding attention window size equal to D, with $P = 6$ for all datasets except for the Github dataset for which $P = 8$.[5]

MegaByte We also compare to MegaByte [7]. Although MegaByte was originally trained using a patch size of $P = 8$, we found that a patch size of $P = 4$ was often better in our setup. We thus include both of these patch sizes (4 and 8) in our hyperparameter grid for MegaByte. For simplicity, we fix the number of layers in the global and local blocks to be equal, $L_{\text{global}} = L_{\text{local}}$, which is close

[5] We also tried a simplified Sparse Transformer [30] where the first attention layer uses a sliding attention window of size D; the second attention layer uses a strided attention with stride P; and the remaining layers continue to alternate between sliding and strided attention. However in our setup, we found this to perform worse than just using a sliding window attention.

Table 1: **Best bits-per-byte.** Lowest bits-per-byte[3] for each model architecture when trained using 10^{19} FLOPs on different text modalities. The lowest bits-per-byte for each dataset are underlined; and the lowest within 2.5% are bolded. The largest statistical error (due to a finite number of evaluation samples) is 0.4%. SpaceByte significantly outperforms other byte-level architectures and performs on par with the SentencePiece subword Transformer.

	Model	PG-19	arXiv	Github
subword	Transformer (GPT2 tokenizer)	1.013	0.796	0.554
	Transformer (SentencePiece)	**0.989**	0.768	**0.508**
byte-level	Transformer	1.138	0.909	0.655
	Transformer (Window Attention)	1.089	0.818	0.560
	MegaByte	1.083	0.822	0.570
	SpaceByte (fixed P)	1.112	0.804	0.552
	SpaceByte	**1.009**	**0.748**	**0.500**

to what was used by Yu et al. [7]. Similar to SpaceByte, we set the context size to $T = PD$, where D is the global model dimension.

Subword Transformers Our most important baseline is the standard subword Transformer. We train subword Transformers using two different tokenizers (both with a vocabulary size of 50,257): (1) the GPT2 tokenizer [43], and (2) a SentencePiece [44] tokenizer using a byte-pair-encoding model [45] that was separately trained for each dataset. As usual, we set the context size to be equal to the model dimension, $T = D$.

4.2 More Details

We use fairly standard Pre-LN [46, 30, 47] Transformer [48] blocks with no bias terms. Since MegaByte uses Rotary Position Embedding (RoPE) [49], we also use RoPE for all models (which slightly improves the loss). To prevent loss divergences during training, we use qk-layernorm [50–52] (which we strongly recommend) for all models; i.e. we add an extra layer-normalization to the query and key vectors in the self-attention layers.

All hyperparameters have been carefully tuned using grid and random searches. See Appendices A and B for more details.[6]

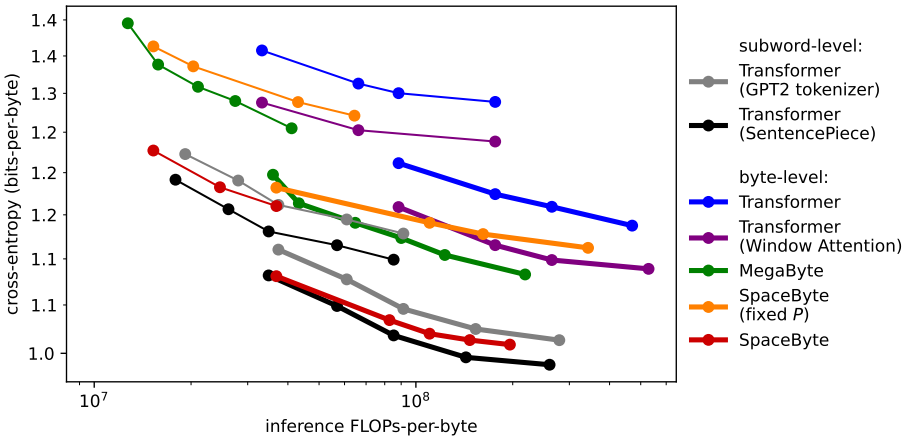
5 Results

We now present our experimental data comparing the different model architectures in compute-controlled settings. Figure 3 plots the Pareto frontier of lowest cross-entropy bits-per-byte and lowest FLOPs-per-byte (i.e. inference compute cost) for each architecture and training compute budget. We assume that the Pareto frontier is convex. Thus, for each architecture and compute budget, we perform a grid search over model dimension and number of layers; we then draw a piecewise-linear line connecting the best (i.e. minimal subset of) models such that all other models (not shown in figure) lie above and to the right of the line. Table 1 summarizes the results for the lowest overall bits-per-byte for each architecture.

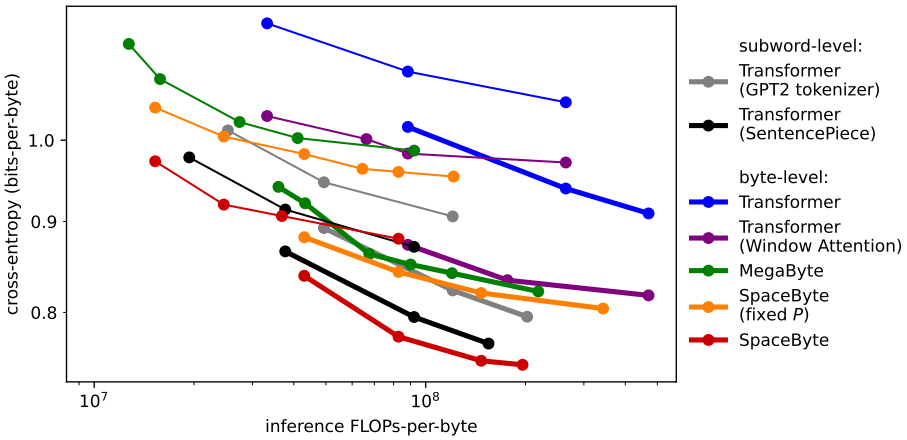
Across all datasets, training compute budgets, and inference compute budgets (i.e. FLOPs-per-byte), SpaceByte significantly outperforms all other byte-level architectures. SpaceByte also consistently outperforms the subword Transformer when using GPT2 tokens, and by a wide margin on the arXiv and Github datasets. SpaceByte roughly matches the performance of the most competitive baseline, the subword Transformer using the SentencePiece tokenizer, with SpaceByte performing slightly better on the arXiv and Github datasets. Figure 3 also suggests that SpaceByte's performance improves faster than the subword Transformer as the training compute budget increases.

Byte-level architectures other than SpaceByte perform significantly worse than SpaceByte or the SentencePiece Transformer. For example, for PG-19, the next best byte-level architecture is

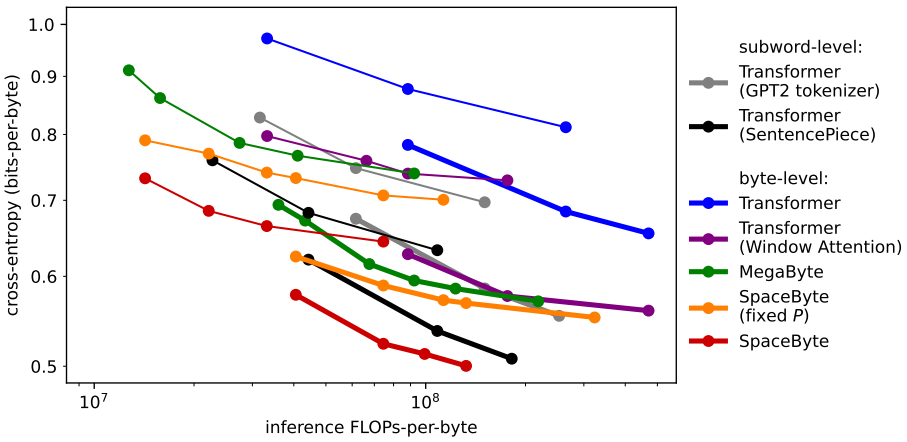
[6] Our training code and data reproduction steps can be found at github.com/kjslag/spacebyte.



(a) PG-19 dataset



(b) arXiv dataset



(c) Github dataset

Figure 3: Pareto frontier of the cross-entropy bits-per-byte[3] vs FLOPs-per-byte during inference (details in Appendix A.1) for each model architecture trained using 10^{18} (connected by thin lines) or 10^{19} (thick lines) FLOPs on different datasets (on a log-log scale). Each dot describes a model with a different number of layers and/or model dimension. Lower and to the left is better. SpaceByte (red) outperforms all other byte-level architectures across the entire Pareto frontier for all datasets. SpaceByte roughly matches the performance of the subword Transformer using SentencePiece tokens, and outperforms the subword Transformer using GPT2 tokens.

Table 2: **Comparison with other works.** We compare SpaceByte to byte-level models trained in other works, along with a subword transformer that we train. All models are trained using roughly the same inference FLOPs-per-byte ($\approx$ 728M). The bits-per-byte for the Transformer, PerceiverAR, and MegaByte models are taken from Yu et al. [7], while MambaByte results are taken from Wang et al. [6]. The best bits-per-byte for each dataset are underlined; and the lowest within 3% are bolded. The largest 1-sigma statistical error (due to a finite number of evaluation samples) for the models we train is less than 0.001. SpaceByte is the overall best performing byte-level model and consistently performs within a few percent of the subword Transformer.
† These models used slightly different datasets for training and/or testing. For MambaByte-353M, we estimate that this difference very roughly amounts to an extra 3% statistical error.

	Model	Context size	Data trained	Test bits-per-byte ↓			
				PG-19	Stories	arXiv	Github
subword	Transformer-1B	2048 tokens $\sim$ 8192 bytes	$\approx$ 30B* bytes	**<u>0.908</u>**	**<u>0.809</u>**	**0.666**	**<u>0.400</u>**
byte-level	Transformer-320M [7]	1024	80B	1.057	1.064	0.816†	0.575†
	PerceiverAR-248M [7]	8192	80B	1.104	1.070	0.791†	0.546†
	MegaByte-758M+262M [7]	8192	80B	1.000	0.978	**0.678**†	**0.411**†
	MambaByte-353M [6]	8192	30B*	**0.930**	0.908†	**<u>0.663</u>**†	**0.396**†
	SpaceByte-793M+184M	8192 (bytes)	30B* (bytes)	**0.918**	**0.833**	**<u>0.663</u>**	**0.411**

MegaByte; however, MegaByte trained using 10^{19} FLOPs (thick green line in Figure 3a) performs worse across nearly the entire Pareto frontier than the SentencePiece Transformer trained using only 10% as many training FLOPs (thin black line). Although the standard byte-level transformer (which is the primary baseline used by Yu et al. [7], blue in Figure 3) performs significantly worse than the other byte-level models, we note that by simply using a sliding window attention mechanism to increase the context size to more closely match that of the other byte-level models, this stronger baseline (purple) performs almost as well as MegaByte. Nevertheless, SpaceByte still significantly outperforms this stronger baseline.

To verify the importance of dynamic patch sizes for SpaceByte's performance, we compare SpaceByte to a variant of SpaceByte with fixed patch sizes (orange in Figure 3). We observe that fixing the patch size significantly degrades the performance of SpaceByte.

Note that on the arXiv and Github datasets, the subword Transformer performs significantly worse when using GPT2 tokens (which were trained on WebText [43]) than SentencePiece tokens (which were trained using the specific dataset). This exemplifies the bias that tokenization can introduce on data distributions different from what the tokenizer was trained on.

6 Comparison with Other Works

We also compare SpaceByte performance to byte-level models trained in other works. Yu et al. [7] trained Transformer, PerceiverAR, and MegaByte models, each using the same amount of compute, FLOPs-per-byte, and data (80B bytes). Wang et al. [6] additionally trained a MambaByte model using the same FLOPs-per-byte but only 30B bytes of data. We train SpaceByte-793M+184M ($D = 1536$, $D_{\text{local}} = 768$, $L_{\text{local}} = 26$, $L_{\text{global}} = 28$) using roughly the same inference FLOPs-per-byte (728M) but also only 30B bytes of data (following Wang et al. [6]). Training these models thus requires roughly $3 \times 728\text{M}$ FLOPs-per-byte $\times$ 30B bytes $\approx 6.5 \times 10^{19}$ FLOPS, where the factor of three comes from converting inference FLOPs-per-byte to training FLOPs-per-byte (which additionally requires a backwards pass). For this experiment, we set the context size of SpaceByte to 8192 bytes to follow the prior works. See Appendix A for more details.

We also train subword Transformer-1B ($D = 1536$) models using the SentencePiece tokenizer (except for the Stories dataset, for which we use the GPT2 tokenizer). The average number of bytes per token

for the PG-19, Stories, arXiv, and Github datasets are 4.05, 4.39, 3.73, and 3.31, respectively. To match the FLOPs-per-byte of the subword Transformer-1B models to the byte-level models, we set the number of layers to 40, 44, 37, or 31, for Transformer-1B on these four respective datasets.

Results are shown in Table 2. We show experiments for the PG-19 [41], Stories [53], arXiv (extracted from The Pile [42]), and Github (extracted from The Pile [42]) datasets.[7] Yu et al. [7] used proprietary "arXiv" and "Code" datasets, which we do not have access to. Following Wang et al. [6], we compare Yu et al. [7]'s results to the similar (but likely slightly different) arXiv and Github components of The Pile [42]. However, Wang et al. [6] use their own test splits to evaluate MambaByte-353M on Stories, arXiv, and Github. Due to the rather small test splits (~ 100MB for the arXiv and Github datasets), this difference can be significant. For example, the validation (and test) bits-per-byte for SpaceByte-793M+184M on the Stories, arXiv, and Github datasets are 0.877 (0.833), 0.658 (0.663) and 0.397 (0.411), which differ by $+5\%$, -1%, and -3%, respectively. Given this variation, the bits-per-byte of MambaByte-353M and SpaceByte-793M+184M are not statistically different on the arXiv or Github datasets.

Overall, we find that SpaceByte outperforms the byte-level models trained in other works. SpaceByte outperforms MegaByte, even though MegaByte was trained using 2.7 times as much compute and data. Moreover, SpaceByte's performance is competitive with the subword Transformer-1B.

7 Conclusion

We have proposed a new byte-level Transformer decoder architecture, SpaceByte. Our compute-controlled experiments show that SpaceByte outperforms all other byte-level architectures and roughly matches the performance of sub-word level Transformers.

Limitations SpaceByte uses a simple byte partitioning rule that relies on "spacelike" bytes, such as spaces which typically denote word boundaries. As such, SpaceByte should not be expected to perform well on arbitrary sequences of bytes, such as images or audio. Some languages, such as Chinese, do not use spaces between words. SpaceByte is somewhat robust to these languages, since e.g. Chinese characters are encoded using three bytes in UTF-8, which SpaceByte will group together. However, our preliminary experiments suggest that SpaceByte performs worse than subword transformers on Chinese text. It would therefore be desirable to improve upon and generalize SpaceByte's global block insertion rule.

The variable spacing between global blocks makes it more challenging to design and implement an efficient batched inference sampling algorithm for SpaceByte.

Future Work SpaceByte uses multiscale modeling where the local model operates on bytes while the global model typically operates on words. Another natural extension of our work is to try recursively applying multiscale modeling at even longer scales, such as the sentence or paragraph level. It would also be fruitful to investigate if Mamba blocks [25] could further improve SpaceByte's performance.

Acknowledgments and Disclosure of Funding

We thank Tushaar Gangavarapu, Junxiong Wang, and Lili Yu for helpful conversations. This work was supported in part by the NSF Campus Cyberinfrastructure grant CC* Compute: Interactive Data Analysis Platform OAC-2019007 and by Rice University's Center for Research Computing (CRC).

References

[1] J. Pourmostafa Roshan Sharami, D. Shterionov, and P. Spronck. A Systematic Analysis of Vocabulary and BPE Settings for Optimal Fine-tuning of NMT: A Case Study of In-domain Translation. March 2023. doi: 10.48550/arXiv.2303.00722.

[2] Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. Language model tokenizers introduce unfairness between languages. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=78yDLKi95p.

[7] Yu et al. [7] and Wang et al. [6] also show results for a "Books" dataset [42] derived from Books3 (which is similar to PG-19 but also includes modern books). However, the legality of obtaining Books3 is questionable due to copyrights. We consequently exclude comparisons to this dataset.

[3] Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614.

[4] Jessica Rumbelow and mwatkins. Solidgoldmagikarp (plus, prompt generation), 2023. URL https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation.

[5] Peter Welinder, May 2022. URL https://twitter.com/npew/status/1525900849888866307.

[6] Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. MambaByte: Token-free Selective State Space Model. January 2024. doi: 10.48550/arXiv.2401.13660.

[7] Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MegaByte: Predicting Million-byte Sequences with Multiscale Transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JTmO2V9Xpz.

[8] Avijit Thawani, Saurabh Ghanekar, Xiaoyuan Zhu, and Jay Pujara. Learn your tokens: Word-pooled tokenization for language modeling. In *Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=O9zrG7NB3X.

[9] Piotr Nawrot, Jan Chorowski, Adrian Łańcucki, and Edoardo M. Ponti. Efficient Transformers with Dynamic Token Pooling. art. arXiv:2211.09761, November 2022. doi: 10.48550/arXiv.2211.09761.

[10] Brian Lester, Jaehoon Lee, Alex Alemi, Jeffrey Pennington, Adam Roberts, Jascha Sohl-Dickstein, and Noah Constant. Training LLMs over Neurally Compressed Text. art. arXiv:2404.03626, April 2024.

[11] William Fleshman and Benjamin Van Durme. Toucan: Token-Aware Character Level Language Modeling. *arXiv e-prints*, art. arXiv:2311.08620, November 2023. doi: 10.48550/arXiv.2311.08620.

[12] Tomasz Limisiewicz, Terra Blevins, Hila Gonen, Orevaoghene Ahia, and Luke Zettlemoyer. MYTE: Morphology-Driven Byte Encoding for Better and Fairer Multilingual Language Modeling. *arXiv e-prints*, art. arXiv:2403.10691, March 2024. doi: 10.48550/arXiv.2403.10691.

[13] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

[14] Nathan Godey, Roman Castagné, Éric de la Clergerie, and Benoît Sagot. MANTa: Efficient Gradient-Based Tokenization for Robust End-to-End Language Modeling. art. arXiv:2212.07284, December 2022. doi: 10.48550/arXiv.2212.07284.

[15] Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. CANINE: Pre-training an Efficient Tokenization-Free Encoder for Language Representation. art. arXiv:2103.06874, March 2021. doi: 10.48550/arXiv.2103.06874.

[16] Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun'ichi Tsujii. CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In *International Committee on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online), December 2020. doi: 10.18653/v1/2020.coling-main.609. URL https://aclanthology.org/2020.coling-main.609.

[17] Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast Character Transformers via Gradient-based Subword Tokenization. art. arXiv:2106.12672, June 2021. doi: 10.48550/arXiv.2106.12672.

[18] Lukas Edman, Antonio Toral, and Gertjan van Noord. Subword-Delimited Downsampling for Better Character-Level Translation. art. arXiv:2212.01304, December 2022. doi: 10.48550/arXiv.2212.01304.

[19] Makesh Narsimhan Sreedhar, Xiangpeng Wan, Yu Cheng, and Junjie Hu. Local byte fusion for neural machine translation. In *Association for Computational Linguistics*, pages 7199–7214, Toronto, Canada, July 2023. doi: 10.18653/v1/2023.acl-long.397. URL https://aclanthology.org/2023.acl-long.397.

[20] Lukas Edman, Gabriele Sarti, Antonio Toral, Gertjan van Noord, and Arianna Bisazza. Are Character-level Translations Worth the Wait? Comparing ByT5 and mT5 for Machine Translation. art. arXiv:2302.14220, February 2023. doi: 10.48550/arXiv.2302.14220.

[21] Kris Cao. What is the best recipe for character-level encoder-only modelling? art. arXiv:2305.05461, May 2023. doi: 10.48550/arXiv.2305.05461.

[22] Sandeep Subramanian, Ronan Collobert, Marc'Aurelio Ranzato, and Y-Lan Boureau. Multi-scale Transformer Language Models. art. arXiv:2005.00581, May 2020. doi: 10.48550/arXiv.2005.00581.

[23] Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In *Association for Computational Linguistics*, pages 1559–1571, Seattle, United States, July 2022. doi: 10.18653/v1/2022.findings-naacl.117. URL https://aclanthology.org/2022.findings-naacl.117.

[24] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. art. arXiv:2006.03236, June 2020. doi: 10.48550/arXiv.2006.03236.

[25] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://openreview.net/forum?id=AL1fq05o7H.

[26] Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. Compression Represents Intelligence Linearly. art. arXiv:2404.09937, April 2024. doi: 10.48550/arXiv.2404.09937.

[27] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. AAAI, 2019. doi: 10.1609/aaai.v33i01.33013159. URL https://doi.org/10.1609/aaai.v33i01.33013159.

[28] Md Mofijul Islam, Gustavo Aguilar, Pragaash Ponnusamy, Clint Solomon Mathialagan, Chengyuan Ma, and Chenlei Guo. A vocabulary-free multilingual neural tokenizer for end-to-end task learning. Association for Computational Linguistics, May 2022. doi: 10.18653/v1/2022.repl4nlp-1.10. URL https://aclanthology.org/2022.repl4nlp-1.10.

[29] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. April 2020. doi: 10.48550/arXiv.2004.05150.

[30] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. April 2019. doi: 10.48550/arXiv.1904.10509.

[31] Ofir Press, Noah A. Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs. In *Association for Computational Linguistics*, pages 5493–5505, August 2021. doi: 10.18653/v1/2021.acl-long.427. URL https://aclanthology.org/2021.acl-long.427.

[32] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. art. arXiv:1603.08983, March 2016. doi: 10.48550/arXiv.1603.08983.

[33] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. SkipNet: Learning Dynamic Routing in Convolutional Networks. art. arXiv:1711.09485, November 2017. doi: 10.48550/arXiv.1711.09485.

[34] Andreas Veit and Serge Belongie. Convolutional Networks with Adaptive Inference Graphs. art. arXiv:1711.11503, November 2017. doi: 10.48550/arXiv.1711.11503.

[35] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. BlockDrop: Dynamic Inference Paths in Residual Networks. art. arXiv:1711.08393, November 2017. doi: 10.48550/arXiv.1711.08393.

[36] Tal Schuster, Adam Fisch, Tommi Jaakkola, and Regina Barzilay. Consistent accelerated inference via confident adaptive transformers. In *Association for Computational Linguistics*, pages 4962–4979, Online and Punta Cana, Dominican Republic, November 2021. doi: 10.18653/v1/2021.emnlp-main.406. URL https://aclanthology.org/2021.emnlp-main.406.

[37] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic Neural Networks: A Survey. art. arXiv:2102.04906, February 2021. doi: 10.48550/arXiv.2102.04906.

[38] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-Depths: Dynamically allocating compute in transformer-based language models. art. arXiv:2404.02258, April 2024.

[39] Noam Shazeer. Fast Transformer Decoding: One Write-Head is All You Need. art. arXiv:1911.02150, November 2019. doi: 10.48550/arXiv.1911.02150.

[40] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Association for Computational Linguistics*, pages 4895–4901, Singapore, December 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298.

[41] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

[42] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. December 2020. doi: 10.48550/arXiv.2101.00027.

[43] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[44] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Association for Computational Linguistics*, pages 66–71, nov 2018. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012.

[45] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Association for Computational Linguistics*, pages 1715–1725, August 2016. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

[46] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxZX20qFQ.

[47] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning deep transformer models for machine translation. In *Association for Computational Linguistics*, pages 1810–1822, July 2019. doi: 10.18653/v1/P19-1176. URL https://aclanthology.org/P19-1176.

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[49] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. April 2021. doi: 10.48550/arXiv.2104.09864.

[50] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4246–4253, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.379. URL https://aclanthology.org/2020.findings-emnlp.379.

[51] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin F. Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Patrick Collier, Alexey Gritsenko, Vighnesh Birodkar, Cristina Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetić, Dustin Tran, Thomas Kipf, Mario Lučić, Xiaohua Zhai, Daniel Keysers, Jeremiah Harmsen, and Neil Houlsby. Scaling Vision Transformers to 22 Billion Parameters. February 2023. doi: 10.48550/arXiv.2302.05442.

[52] Mitchell Wortsman, Peter J Liu, Lechao Xiao, Katie E Everett, Alexander A Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=d8w0pmvXbZ.

[53] Trieu H. Trinh and Quoc V. Le. A Simple Method for Commonsense Reasoning. art. arXiv:1806.02847, June 2018. doi: 10.48550/arXiv.1806.02847.

[54] Ofir Press and Lior Wolf. Using the Output Embedding to Improve Language Models. August 2016. doi: 10.48550/arXiv.1608.05859.

[55] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[56] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. art. arXiv:1211.5063, November 2012. doi: 10.48550/arXiv.1211.5063.

[57] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=H4DqfPSibmx.

[58] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

[59] Yoav Levine, Noam Wies, Or Sharir, Hofit Bata, and Amnon Shashua. The Depth-to-Width Interplay in Self-Attention. art. arXiv:2006.12467, June 2020. doi: 10.48550/arXiv.2006.12467.

Table 3: **Model hyperparameters.** Hyperparameters for models shown in Table 1.

Model	Dataset	Parameters	FLOPs-per-byte	Hyperparameters		
				L ($L_{\text{global}}/L_{\text{local}}$)	D (D/D_{local})	T
Transformer (GPT2 tokenizer)	PG-19	454M	279M	32	1024	1024
	arXiv	253M	202M	16	1024	1024
	Github	253M	253M	16	1024	1024
Transformer (SentencePiece)	PG-19	454M	260M	32	1024	1024
	arXiv	253M	155M	16	1024	1024
	Github	253M	182M	16	1024	1024
Transformer	PG-19	202M	470M	16	1024	1024
	arXiv	202M	470M	16	1024	1024
	Github	202M	470M	16	1024	1024
Transformer (Window Attention)	PG-19	227M	529M	32	768	4608
	arXiv	202M	470M	16	1024	6144
	Github	202M	470M	16	1024	8192
MegaByte	PG-19	201M+51M	219M	16/16	1024/512	4096
	arXiv	201M+51M	219M	16/16	1024/512	4096
	Github	201M+51M	219M	16/16	1024/512	4096
SpaceByte (fixed P)	PG-19	201M+113M	343M	16/16	1024/768	6144
	arXiv	201M+113M	343M	16/16	1024/768	6144
	Github	201M+113M	323M	16/16	1024/768	8192
SpaceByte	PG-19	201M+50M	196M	16/16	1024/512	6144
	arXiv	201M+50M	196M	16/16	1024/512	6144
	Github	151M+38M	132M	12/12	1024/512	8192

Table 4: **Model hyperparameters.** Hyperparameters for models shown in Table 2. In order to roughly match the FLOPs-per-byte of the other models, for the subword-level Transformer-1B, we used 40, 44, 37, and 31 layers for the PG-19, Stories, arXiv, and Github datasets, respectively.

Model	Parameters	FLOPs-per-byte	Hyperparameters		
			L ($L_{\text{global}}/L_{\text{local}}$)	D (D/D_{local})	Others
Transformer	1B	$\approx$ 730M	40, 44, 37, or 31	1536	subword-level
Transformer	320M [7]	732M	22	1024	byte-level
PerceiverAR	248M [7]		17	1024	latents $= 1024$
MegaByte	758M+262M [7]	729M	14/18	2048/1024	$P = 8$
MambaByte	353M [6]	713M	53	1024	$n_{\text{state}} = 16$
SpaceByte	793M+184M	728M	28/26	1536/768	$T_{\text{global}} = 1344$

A Model Details

Hyperparameters for models shown in Table 1 and Table 2 are summarized in Table 3 and Table 4, respectively. In Figure 4, we show another perspective of Figure 3 where we plot the bits-per-byte vs the bytes trained divided by the number of model parameters.

For the subword models (but not the byte-level models), we tie the input embedding weights with the output linear matrix weights [54]. In self-attention layers, we use a key dimension equal to 64. Although we apply RoPE embeddings, we also included trained position embeddings.

SpaceByte For SpaceByte, we include trained position embeddings just before the first local transformer block, and just before the first global transformer block.

Table 5: **Model non-embedding parameter counts.** For MegaByte and SpaceByte, we separate the number of parameters (m) into the global (m_{global}) and local (m_{local}) model contributions. We ignore embeddings and subleading parameters, such as layer norms, but include de-embedding parameters. See Section A.1 for symbol definitions.

Architecture	Parameters (non-embedding)	Component
Transformer	$m = L \times 4D^2$	Multi-head attention
	$+ L \times 2e_{\text{ff}}D^2$	Feed-forward
	$+ DV$	De-embedding
MegaByte	$m_{\text{global}} = L_{\text{global}} \times 4D^2$	Global multi-head attention
	$+ L_{\text{global}} \times 2e_{\text{ff}}D^2$	Global feed-forward
	$m_{\text{local}} = D_{\text{local}}\frac{D}{P}$	Global-to-local projection
	$+ L_{\text{local}} \times 4D_{\text{local}}^2$	Local multi-head attention
	$+ L_{\text{local}} \times 2e_{\text{ff}}D_{\text{local}}^2$	Local feed-forward
	$+ D_{\text{local}}V$	De-embedding
SpaceByte	$m_{\text{global}} = L_{\text{global}} \times 4D^2$	Global multi-head attention
	$+ L_{\text{global}} \times 2e_{\text{ff}}D^2$	Global feed-forward
	$m_{\text{local}} = L_{\text{local}} \times 4D_{\text{local}}^2$	Local multi-head attention
	$+ L_{\text{local}} \times 2e_{\text{ff}}D_{\text{local}}^2$	Local feed-forward
	$+ D_{\text{local}}V$	De-embedding

Table 6: **Inference FLOPs-per-token.** We calculate the inference FLOPs-per-token in terms of the numbers of parameters (m), shown in Table 5. See Section A.1 for symbol definitions.

Architecture	inference FLOPs-per-token
Transformer	$2m + 2L\,(2WD)$
MegaByte	$2m_{\text{global}}\frac{1}{P} + 2L_{\text{global}}\,(2\frac{T}{P}D)\,\frac{1}{P} +$ $2m_{\text{local}} + 2L_{\text{local}}\,(2PD_{\text{local}})$
SpaceByte	$2m_{\text{global}}\frac{T_{\text{global}}}{T_{\text{local}}} + 2L_{\text{global}}\,(2T_{\text{global}}D)\,\frac{T_{\text{global}}}{T_{\text{local}}} +$ $2m_{\text{local}} + 2L_{\text{local}}\,(2W_{\text{local}}D_{\text{local}})$

Just like the other models, SpaceByte is trained using a fixed context size of T bytes. At the same time, we also fix the maximum global context size of T_{global} patches. However, the number of patches in a given context of T bytes is usually not exactly equal to T_{global}. To handle this mismatch during training, if the number of patches from applying the patching rule (see e.g. Figure 2) to a context of T bytes is greater T_{global}, then we simply ignore the bytes within these extra patches when calculating the cross entropy. Alternatively, if the number of patches is less than T_{global}, then we pad the activations for the global transformer blocks with zeroes and ignore the output of these global blocks. Thus, the input activations to the global blocks is always a tensor of same shape for each iteration. This discrepancy between the maximal global context size T_{global} and the actual number of patches results in a small fracton of wasted compute during training, which we roughly minimize by roughly tuning T/T_{global}. See Appendix C for pseudocode.

During inference, the model must stop predicting tokens before either the max number of bytes (T) or the max number of patches (T_{global}) is reached.

A.1 FLOPs

The inference FLOPs-per-byte is the number of FLOPs required to output a byte of text during inference. We calculate the FLOPs-per-byte as the FLOPs-per-token divided by the average number of bytes per token (which is equal to 1 for byte-level models).

The FLOPs-per-token is the number of FLOPs required to output a token of text during inference (or byte of text for byte-level models). The FLOPs-per-token for the various architectures is shown in Table 6.

Notation For all architectures, T is the context length; D is the model dimension (of the global model for SpaceByte and MegaByte); $e_{\text{ff}} = 4$ is the model dimension expansion factor for feed-forward layers; and V is the vocabulary size (which is 256 for byte-level models and 50257 for our subword models). For the transformer architecture, L is the number of transformer blocks, and W is the attention window size (which is equal to T if window attention is not used). For SpaceByte and MegaByte, D_{local} is the dimension of the local model; L_{local} is the number of local transformer blocks; and L_{global} is the number of global transformer blocks. For SpaceByte, T_{global} is the maximum context size for the global model, and W_{local} (which we set to D_{local}) is the attention window size for the local blocks. For MegaByte, P is the patch size.

B Training Details

B.1 Data

Each dataset prepared by downloaded it from Hugging Face[8], concatenating sequences together, and separating sequences with a special BOS token. When preparing a training sample with context size T, we uniformly and randomly sample a sub-sequence of length T from the concatenated dataset. If a BOS token is found in this subset, we align the context with the first BOS token found; i.e. we take the context to be the first BOS token followed by the next $T-1$ tokens in the concatenated dataset. If a BOS token is not found in the subset, we prepend a BOS token to the context. The context window is always full and always begins with a BOS token.

For the SpaceByte models, we always insert global blocks after a BOS token. A valid UTF-8 encoding never makes use of the byte values 254 or 255. We use 255 to encode the BOS token.

We train the SentencePiece tokenizers using the following command:

```
spm_train --input=train.txt --model_prefix=sp --model_type=bpe
    --vocab_size=50257 --num_threads=32 --byte_fallback=True
    --allow_whitespace_only_pieces=True --remove_extra_whitespaces=False
    --normalization_rule_name=identity --input_sentence_size=10000000
```

B.2 Training

All models are trained using AdamW [55] with $\beta_1 = 0.9$, $\beta_2 = 0.98$, batch size 64, weight decay of 0.01, and gradient clipping [56] with a maximum norm of 1.0. Trainable parameters are randomly initialized using a normal distribution with standard deviation $\sigma_{\text{init}} = 1$ for all parameters except for linear weight matrices, which are initialized with standard deviation of $\sigma_{\text{init}} = 1/\sqrt{d_{\text{in}}}$, where d_{in} is the input dimension for the linear layer. We scale the learning rate for each parameter by its initialization standard deviation σ_{init}.

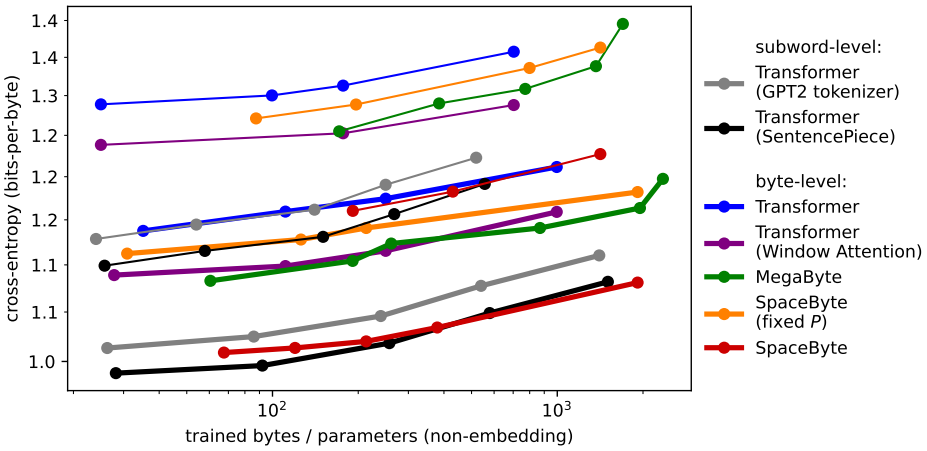
With this setup, we found in our early hyperparameter search experiments that the optimal max learning rate for all models is approximately $\gamma = 0.005B^{-0.5} = 0.000625$, where $B = 64$ is the batch size. We therefore used $\gamma = 0.000625$ as the max learning rate for all models trained in this work. We applied a linear learning rate warmup over the first 1% of training iterations. We also multiply the learning rate by a "half-cosine" learning rate decay function $\cos(\pi x/2)$, where $0 \leq x \leq 1$ is the fraction of training iterations completed.[9]

Each model was trained using PyTorch on a single 40GB Nvidia A40 and A100 GPUs with mixed-precision (bfloat16 and float32) training and FlashAttention [57, 58]. SpaceByte-793M+184M took the longest to train, requiring about 10 days on an A100 GPU.[10]

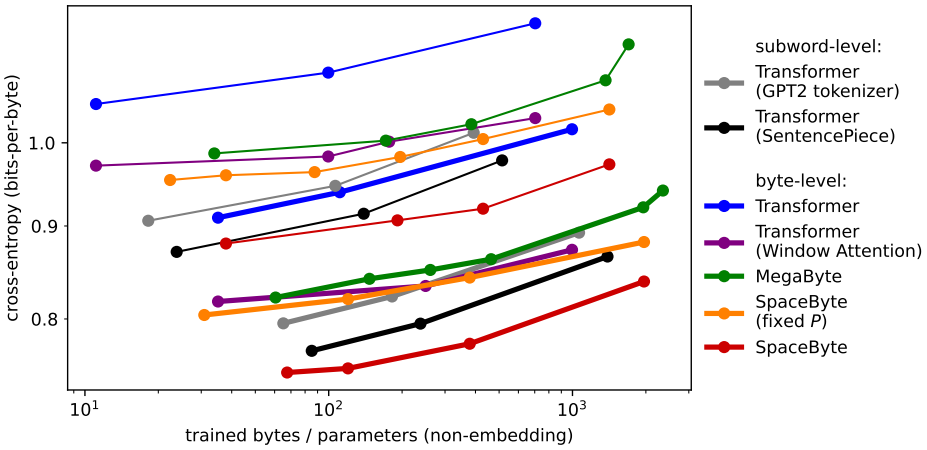
[8] https://huggingface.co/datasets/pg19
https://huggingface.co/datasets/lucadiliello/STORIES
https://huggingface.co/datasets/monology/pile-uncopyrighted

[9] In our setup, we found $\cos(\pi x/2)$ to slightly outperform the more standard cosine decay from 1 to 0.1.

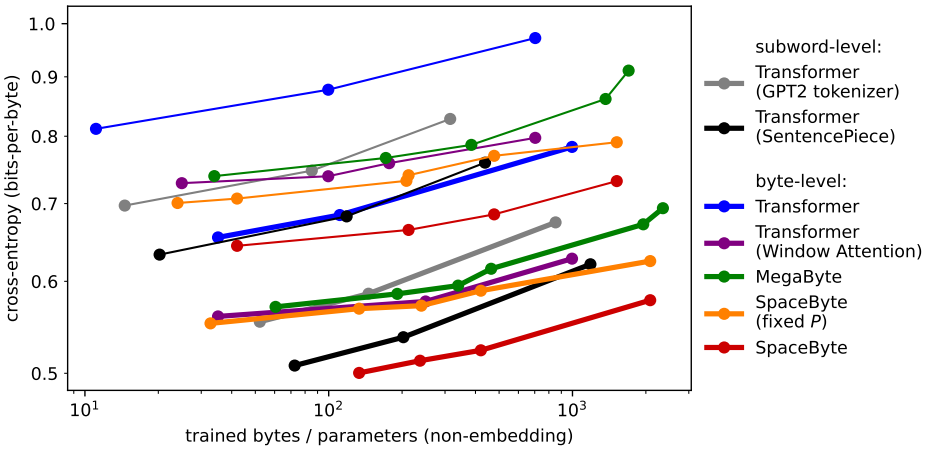
[10] We very roughly estimate that additional preliminary and failed experiments not shown in this work required roughly as many FLOPs as the experiments shown in this work.



(a) PG-19 dataset



(b) arXiv dataset



(c) Github dataset

Figure 4: The Pareto frontier models from Figure 3, where we plot the bits-per-byte vs the number of bytes used for training divided by the number of non-embedding parameters (defined in Table 5).

B.3 Hyperparameter Grid

We train models using a grid of different model dimensions and numbers of layers. In our early small-scale experiments, we found that the hyperparameter grid described below effectively explores the bits-per-byte and FLOPs-per-byte Pareto frontier for all models. To simplify the hyperparameter grid, we restrict ourselves to model dimensions and layer numbers to half-powers of two, i.e. a power of two times 1 or $\frac{3}{2}$.

For models trained using 10^{18} FLOPs, we train model dimensions $D \in \{384, 512, 768\}$. For models trained using 10^{19} FLOPs, we train model dimensions $D \in \{512, 768, 1024\}$.

For SpaceByte and MegaByte, D is the global model dimension. The local model dimension is chosen from $D_{\text{local}} \in \{\frac{1}{2}D, \frac{3}{4}D\}$ if D is a power of two, or $D_{\text{local}} \in \{\frac{1}{2}D, \frac{2}{3}D\}$ if D is a power of two times $\frac{3}{2}$. However, in order to avoid excessively low FLOP utilization, we restrict $D_{\text{local}} \geq 256$ (or $D_{\text{local}} \geq 384$) for models trained using 10^{18} FLOPs (or 10^{19} FLOPs).

To set the number of layers, we roughly follow Levine et al. [59], who found that the compute-optimal number of layers for a Transformer roughly follows $L \sim 12.5 \log_2(D/154)$. We round this number to the nearest half-power of two to obtain L_D, for which $L_{384} = 16$, $L_{512} = 24$, $L_{768} = 32$, and $L_{1024} = 32$. For Transformer models, we choose the number of layers from $L \in \{\frac{1}{2}L_D, L_D\}$.

For SpaceByte and MegaByte models, we choose the number of local and global layers from $L_{\text{local}} = L_{\text{global}} \in \{\frac{3}{8}L_D, \frac{1}{2}L_D\}$ if L_D is a power of two, or $L_{\text{local}} = L_{\text{global}} \in \{\frac{1}{3}L_D, \frac{1}{2}L_D\}$ if L_D is a power of two times $\frac{3}{2}$.

C Pseudocode

See Listing 1 for Pytorch pseudocode for the SpaceByte forward method. The implementation of SpaceByte that we used in our experiments can be found at github.com/kjslag/spacebyte.

Listing 1: Pytorch pseudocode for SpaceByte

```
def forward(self, tokens, targets=None):
    B, T = tokens.shape # batch size, context size
    T_global = self.global_context_size
    D_local = self.local_model_dimension
    D = self.global_model_dimension

    # embedding:
    x = self.token_embedding(tokens)
    x = x + self.local_position_encoding
    # initial local transformer blocks:
    for block in self.initial_blocks:
        x = block(x)

    # global block insertion rule:
    use_global = ( # not a letter, number, or UTF-8 continuation byte
        (tokens < ord('0')) |
        ((ord('9') < tokens) & (tokens < ord('A'))) |
        ((ord('Z') < tokens) & (tokens < ord('a'))) |
        ((ord('z') < tokens) & (tokens < 0b1000_0000)) |
        (0b1100_0000 <= tokens) )
    use_global[:, 1:] &= use_global[:, :-1].bitwise_not() # not
       preceded by another spacelike byte
    use_global |= tokens == self.BOS_token # always use global blocks
       after BOS tokens

    # calculate global block indices:
    num_global = torch.full((B,), -1) # number of global blocks used
    global_idx = torch.full((B, T_global), T-1) # global block indices
    for b in range(B):
        idx, = use_global[b].nonzero(as_tuple=True)
        if targets is not None and len(idx) > T_global:
            # ignore targets with insufficient global blocks:
            targets[b, idx[T_global]:] = -1
        num_global[b] = len(idx[:T_global])
        global_idx[b, :num_global[b]] = idx[:T_global]

    # select activations for global blocks:
    y = x.gather(1, global_idx[:,:,None].expand(B, T_global, D_local))
    # expand model dimension by padding with zeros:
    y = torch.cat([torch.zeros(B, T_global, D - D_local), y], -1)

    # global transformer blocks:
    y = y + self.global_position_encoding
    for block in self.global_blocks:
        y = block(y)

    # add global block activations to local blocks:
    x = torch.stack([
        x[b].index_add(0, global_idx[b, :n], y[b, :n, -D_local:])
        for b, n in enumerate(num_global) ])

    # final local transformer blocks:
    for block in self.final_blocks:
        x = block(x)
    # de-embedding:
    logits = self.logits_linear(self.layer_norm(x))

    cross_entropy_loss = None
    if targets is not None:
        cross_entropy_loss = torch.nn.functional.cross_entropy(
            logits.view(B*T, 256), targets.view(B*T),
            ignore_index=-1).view(B, T)
    return logits, cross_entropy_loss
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our claims are summarized in the abstract, and restated in the introduction. Our abstract makes three claims, which we enumerate in the following quote:

   > "SpaceByte consists of a byte-level Transformer model, but with extra larger transformer blocks inserted in the middle of the layers. **(1)** We find that performance is significantly improved by applying these larger blocks only after certain bytes, such as space characters, which typically denote word boundaries. **(2)** Our experiments show that for a fixed training and inference compute budget, SpaceByte outperforms other byte-level architectures and **(3)** roughly matches the performance of tokenized Transformer architectures."

   These claims are substantiated in Sections 5 and 6. The first claim is supported by Figure 3 and Table 1, which compare SpaceByte to "SpaceByte (fixed P)," which is a variant of SpaceByte that uses fixed patch sizes instead of applying the larger blocks only after certain bytes. The final two claims are also supported by Figure 3 and Table 1, along with Table 2. At the end of our introduction, we note that our results could be limited to the text modalities (English text, LaTex formatted arXiv papers, and source code) that we study.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: At the end of our introduction, we note that our results could be limited to the text modalities (English text, LaTex formatted arXiv papers, and source code) that we study. We emphasize in Section 2 and our conclusion that generalizing SpaceByte's performance to other text modalities (by improving the global block insertion rule) is an important direction for future work.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be

used reliably to provide closed captions for online lectures because it fails to handle technical jargon.

- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: No theoretical results were presented.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Open source code, job execution scripts, and a jupyter notebook for fully reproducing our results are available at github.com/kjslag/spacebyte.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Open access to code and thorough instructions for reproducing our experiments are available at github.com/kjslag/spacebyte. This includes code for the models, training, and bits-per-byte evaluation, and a requirements.txt for pip.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setting is described in Section 4. The models studied are summarized in Sections 4.1, 4.2, and 6. Additional model details are provided in Appendix A. Training details are provided in Appendix B. Any additional details are available in our released source code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Tables 1 and 2 give upper bounds for the 1-sigma statistical error and clarify that these statistical errors account for the finite number of evaluation samples. We also estimate the statistical uncertainty due to the different test splits used by other works in Section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resource requirements are detailed at the end of Appendix B.2, along with an additional compute estimate for preliminary and failed experiments (in a footnote). Models trained in our main experiments used 10^{18} or 10^{19} FLOPs for training, as specified in Section 4. When comparing to other works, training each model required 6.5×10^{19} FLOPs, as specified in Section 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

Justification: Regarding the data-related concerns, we omit comparisons to the Books3 dataset since this dataset is no longer publically available and because it is unclear to us if Books3 is legal under fair use, as stated in Footnote 7.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We briefly mention in the abstract and introduction that tokenization can increase the vulnerability to adversarial attacks. Lessening this adversarial vulnerability was one of our motivations. We are not aware of any potential negative societal impacts of our work.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not release any data or models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We include citations to all datasets that we use. We believe that we have properly respected the licenses and terms of use for all code and datasets that we used. Notably, we avoided the Books3 dataset for this reason, since Books3 may not be respecting the licenses of the works from which it derives. Code attributions and licenses are provided in the 'license' directory of our released source code.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The only asset introduced is source code. We provide detailed instructions for using this code to reproduce our results in the 'reproduce' directory of the source code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA] .

    Justification: Human subjects were not involved in this research.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA] .

    Justification: Human subjects were not involved in this research.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.