YOU CAN LEARN TOKENIZATION END-TO-END WITH REINFORCEMENT LEARNING

Sam Dauncey* **& Roger Wattenhofer**
ETH Zürich
Zürich, Switzerland
{sdauncey,wattenhofer}@ethz.ch

ABSTRACT

Tokenization is a hardcoded compression step which remains in the training pipeline of Large Language Models (LLMs), despite a general trend towards architectures becoming increasingly end-to-end. Prior work has shown promising results at scale in bringing this compression step inside the LLMs' architecture with heuristics to draw token boundaries, and also attempts to learn these token boundaries with straight-through estimates, which treat the problem of drawing discrete token boundaries as a continuous one. We show that these token boundaries can instead be learned using score function estimates, which have tighter theoretical guarantees due to directly optimizing the problem of drawing discrete token boundaries to minimize loss. We observe that techniques from reinforcement learning, such as time discounting, are necessary to reduce the variance of this score function sufficiently to make it practicable. We demonstrate that the resultant method outperforms prior proposed straight-through estimates, both qualitatively and quantitatively at the 100 million parameter scale.

1 INTRODUCTION

Tokenization is a crucial pre-processing step in the training and inference pipelines of modern LLMs. Standard practice compresses text into symbols representing commonly occurring substrings. This is typically done using algorithms such as Byte-Pair Encoding (BPE), which recursively groups frequently co-occurring byte sequences into individual tokens. State-of-the-art open-source models further augment BPE with numerous hand-crafted decisions. For example, the Gemma-series tokenizers (Rivière et al., 2024) explicitly split digits and preserve whitespace, reflecting the extent to which tokenizer design remains largely artisanal.

A growing line of work seeks to eliminate this BPE step entirely by operating directly on UTF-8 bytes (Xue et al., 2022; Wang et al., 2024; Zheng et al., 2025). This can be viewed as using a tokenizer with a maximally small vocabulary and an effective downsampling rate of 1 token per byte. Recent scaling analyses suggest that downstream loss is optimized by increasing vocabulary size, and thus downsampling rate, with model scale (Tao et al., 2024).

We thus focus our investigation onto methods which admit a growing downsampling rate. Interestingly, BPE itself experiences harsh diminishing returns in downsampling rate as vocabulary size scales. Further, a large vocabulary size can have undesirable downstream effects, such as the emergence of very rare "glitch tokens" (Rumbelow & Watkins, 2023). Marginally increasing the achieved downsampling rate to vocabulary size tradeoff with curricula (Liu et al., 2025), has thus proved fruitful in improving LLM performance.

An alternative family of approaches processes text at the byte level for several transformer layers before downsampling into a shorter sequence of latent tokens (Nawrot et al., 2022; Yu et al., 2023). Some of these methods facilitate a simple modification of the downsampling rate as a hyperparameter. When token boundaries are chosen heuristically—e.g., using whitespace (Slagle, 2024) or spikes in next-byte entropy (Nawrot et al., 2023)—these models have been reported to achieve superior performance to pure BPE transformer models at large scales (Pagnoni et al., 2024).

*Corresponding author.

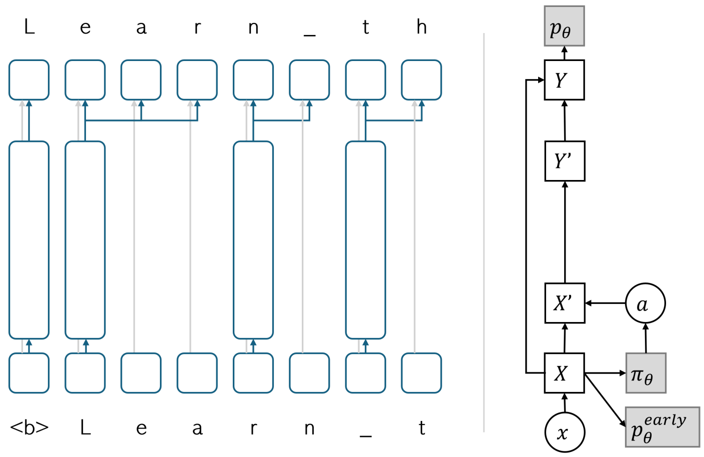

Figure 1: [Left] An example of the autoregressive U-net architecture (Nawrot et al., 2022), computed with the values $x_{0:7}$ = `<b>Learn t` and $a_{0:7} = 11001010$. Blocks with rounded edges represent transformer blocks, arrows represent flow of representations. [Right] the stochastic computation graph (Schulman et al., 2015) of a forward pass of our method, deterministic nodes in squares, stochastic nodes in circles, distributions in gray.

However, such heuristics raise a natural question: can we improve on these boundary rules by learning tokenization itself from end-to-end training? Prior approaches to this question have focused on straight-through estimators (Nawrot et al., 2023), which craft rules for backpropagating gradients from representations internal to the model. We instead investigate score function estimators, which directly approximate the gradient of the expected loss with respect to the token boundaries. Score function estimators have stronger theoretical guarantees, at the cost of a higher variance.

Our main contributions are as follows:

- We show that we can learn tokenization strategies that align closely with semantic boundaries without any explicit prior structure or inductive bias with a score-function estimator, equipped with variance-reduction techniques from reinforcement learning.
- We further find that our method qualitatively and quantitatively outperforms prior approaches using straight-through estimators.
- We demonstrate robust performance across a range of downsampling rates.

2 Theory & Method

In this section, we motivate our method from first principles. We first present some desiderata for an end-to-end tokenisation method §2.1, then show that these desiderata necessarily lead to an autoregressive U-net style architecture §2.2. We then present score functions as the canonical way to learn tokenization under this framework §2.3 and present our method for making them practicable §2.4 - 2.7

2.1 Desiderata for designing a end-to-end tokenization method

For the reasons outlined in §1, we are interested in bringing the tokenization process inside the architecture and training of an LLM. We propose the following desiderata for such a method to be practicable and general:

- **End-to-end tokenizer training**: the token boundary decisions should be learned to minimise loss, in favor of hand-crafted methods and heuristics.

- **End-to-end architecture**: learned representations at the byte level should be re-used at the token level.
- **Efficiency**: use less than 0.1% additional pretraining compute, on existing hardware, than byte-pair-encoding guided tokenization in the forward & backward passes.

2.2 AUTOREGRESSIVE U-NET ARCHITECTURE AND SETUP

In this section, we define notation for the autoregressive U-net architecture (Nawrot et al., 2022), depicted in Figure 1 as the canonical architecture which satisfies the **Efficiency** and **End-to-end architecture** desiderata.

For autoregressive models, tokenization is a fundamentally *discrete* question as it dictates how much compute is spent on a given (sub)sequence at inference time. Furthermore, modern GPUs cannot efficiently perform sparse memory accesses (Yuan et al., 2025), meaning that tokenization for feed-forward models needs to be *consistent across layers*.

Thus, bringing tokenization inside an autoregressive feedforward architecture necessarily have some amount of layers applied at the byte-level with some amount of layers applied at the token-level, in contrast to BPE tokenizer models which operate soley on the token-level. The result is a forward pass structured as follows. Let $x_1 \dots x_N$ be a sequence of input bytes and d_{enc}, d_{mid}, d_{dec} be hyperparameters for the model dimensions.

1. Autoregressively encode the input bytes into byte-level representations $X \in \mathbb{R}^{N \times d_{enc}}$:
$$X = \texttt{encode}(x) \tag{1}$$
2. (Potentially stochastically) predict token boundaries from the byte-level representations, where $a_i = 1$ if we wish to draw a token boundary at x_i and $a_i = 0$ if not.
$$a \sim \pi(X, x) \tag{2}$$
3. Downsample the byte-level representations into token-level representations $X' \in \mathbb{R}^{M \times d_{mid}}$ where $M = \sum_{i=0}^{N} a_i$:
$$X' = \texttt{downsample}(X, a) \tag{3}$$
4. Enrich token-level representations $Y' \in \mathbb{R}^{M \times d_{mid}}$ with an autoregressive feedforward network:
$$Y' = \texttt{mid}(Y) \tag{4}$$
5. Upsample into updated byte-level information $Y \in \mathbb{R}^{N \times d_{dec}}$, potentially carrying encoded byte-level information:
$$Y = \texttt{upsample}(Y', X, a) \tag{5}$$
6. Decode the resulting byte-level representations into predictions of the next bytes $y_i = x_{i+1}$
$$y \sim \texttt{decode}(Y) \tag{6}$$

All the above operations must be autoregressive, for example the `decode` function must be formulated such that X'_j, the downsampled representation at position j, depends only on the preceding bytes $X_{\leq i}$, where i is the minimum token index such that $j = \sum_{k=0}^{i} a_k$.

The score function estimate we detail in the following sections can be flexibly applied to any implementation of the above functions, in contrast to straight-through estimate based approaches which require specialized up/downsamplers. We make the following standard choices for our experiments in §4: `mid` is a decoder-only transformer with full attention, and `encode`, `decode` are decoder-only transformers with sliding window attention and linear embedding/unembedding matrices respectively. Our implementation of `downsample` simply selects the values of X' which correspond to $a_i = 1$ values:

$$\texttt{downsample}(X, a)_i = X'_j \qquad \text{for } j \text{ the minimum value such that } j = \sum_{k=0}^{i} a_k \tag{7}$$

For upsampling, we employ a simple distribute-then-add:

$$\texttt{upsample}(Y', X, a)_j = X_j + Y_i \quad \text{for } i = \sum_{k=0}^{j} a_k \tag{8}$$

2.3 Score function estimation for tokenization

As tokenization is discrete, to satisfy the **End-to-end tokenizer training** desideratum, our model must explore strategies for drawing token boundaries stochastically.

The model next-token cross-entropy loss outputted is thus conditional on a sampled tokenization strategy $a \sim \pi_\theta$, so we can formulate the problem of simultaneously learning the (potentially shared) parameters of the autoregressive model p_θ and the gating strategy π_θ as minimizing the next-token cross-entropy marginalized over all a:

$$\log p_\theta(y|x) = \mathbb{E}_{a\sim\pi_\theta} \log p_\theta(y|a, x). \tag{9}$$

This is a case of a stochastic computation graph as in Schulman et al. (2015), who show that the gradient of this likelihood can be computed as the expectation of the sum of two separate gradients, one being the standard next-token cross-entropy loss conditioned on the tokenization strategy and a correction term which can be interpreted as applying REINFORCE (Williams, 1992) to the tokenization strategy π_θ with the next-token cross-entropy as the reward:

$$\nabla_\theta \mathbb{E}_{a\sim\pi_\theta} \log p_\theta(y|a, x) = \mathbb{E}_{a\sim\pi_\theta} (\underbrace{\nabla_\theta \log p_\theta(y|a, x)}_{\text{conditional loss gradient}} + \underbrace{\log p_\theta(y|a, x) \nabla_\theta \log \pi_\theta(a|x)}_{\text{policy gradient}}). \tag{10}$$

See Appendix A.1 for a complete derivation. This type of estimator is known as a score function estimate. In particular this means that, in the large compute and data limit, performing gradient descent on the policy gradient term above will yield a locally optimal tokenization strategy. No such theoretical guarantee exists for the straight-through estimates we discuss in §3.

2.4 Reducing the variance of the score function estimate

In practice, we wish to use a Monte-Carlo estimate of the policy gradient term in equation 10 with a single sample of a per sequence: using more would break our **Efficiency** desideratum. We empirically find that the naïve REINFORCE policy gradient is too noisy to efficiently learn in this setting. In this section, we show how standard techniques from reinforcement learning can be used to denoise this estimate, solving the corresponding "reward attribution problem": associating which token boundary decisions are responsible for increasing or decreasing the loss in the succeeding bytes.

For this section, we will let d_{model} be the model dimension, `vocab_size` be the number of unique `utf-8` bytes and special characters forming our vocabulary. We will also use the token index of i and the batch index of b, which will be omitted where unused.

Early exit relative rewards As our model is autoregressive, we restrict our analysis to the case where the token boundary decision at byte a_i may only depend on the preceding bytes and token boundary decisions, $x_{<i}$ and $a_{<i}$. Thus, the policy gradient term in eq. (10) treats the token boundary policy $\pi_\theta(a_i|x_{\leq i}, a_{<i})$ as having corresponding rewards $\log p(x_j|a_{<j}, x_{<j})$ for $j > i$.

As $\mathbb{E}_{a\sim\pi_\theta} \nabla_\theta \log \pi_\theta(a_i|x_{\leq i}, a_{<i}) = \mathbf{0}$, we may add any term independent of a_i to these rewards and get a valid policy gradient estimate. The feed-forward nature of the autoregressive U-net architecture

presents a natural method to estimate the baseline, tokenization-independent difficulty of predicting the next token by using the early byte-level embeddings to estimate the next-token probability:

$$\log p_\theta^{early}(x_i = t_j | x_{<i}) = \log \texttt{softmax}(W_{early} X_{i-1})_j. \tag{11}$$

Where $W_{early} \in \mathbb{R}^{d_{model} \times \texttt{vocab\_size}}$ is an unembedding matrix. This gives a reward with a large part of the tokenization-independent noise subtracted out:

$$R_i = \log p_\theta(x_i | x_{<i}, a_{<i}) - \log p_\theta^{early}(x_i | x_{<i}). \tag{12}$$

We initialize the weights of W_{early} to match the final output head to facilitate easy transfer.

Time discounting Summing the above rewards to produce advantages still suffers from too high a variance and due to the reward attribution problem. A common solution to this problem for long-horizon RL is to apply time discounting to the rewards when computing advantages: introducing a small amount of bias into the policy gradient to massively reduce the variance. Intuitively, this decouples the advantages given to far away parts of the sequence, giving us many approximately independent training stimuli for the token boundaries per sequence.

$$G_i = \sum_{j=0}^{N-i-1} \gamma^j R_{i+j+1} \tag{13}$$

In our experiments, we use a discount factor of $\gamma = 0.99$.

Batch-relative advantages Our G_i values tend to be positive as the final-layer model p_θ tends to outperform the early-exit model p_θ^{early}. This bias depends on the token index, with the gap being larger for later token indices. To remedy this, we leverage that during batched training we in fact have a batch of B such values $G_{i,1} \dots G_{i,B}$ for a given forward/backward pass, for B separate sequences. These can be used to center the advantage estimates:

$$A_{i,b} = G_{i,b} - \bar{G}_i \qquad \text{where} \qquad \bar{G}_i = \frac{1}{B} \sum_{b=1}^{B} G_{i,b}. \tag{14}$$

This gives the final policy loss, whose gradient approximates the right hand term of equation (10):

$$\mathcal{L}^\pi = -\sum_{i=0}^{N} \log \pi_\theta(a_i | x_{<i}, a_{<i}) \cdot \texttt{detach}(A_i). \tag{15}$$

Where we use `detach` to emphasize that we do *not* allow gradients propagate through A_i directly.

2.5 Defining the token boundary function

We define a token boundary policy π_θ as a function which gives the probability that our model draws a token boundary at a given index, conditioned on the preceding bytes and token boundaries. We parameterize this probability as the sigmoid of the corresponding logit l_i.

$$a_i | x_{\leq i}, a_{<i} \sim \texttt{Bernoulli}(p_i), \qquad p_i = \pi_\theta(a_i = 1 | x_{\leq i}, a_{<i}) = \sigma(l_i). \tag{16}$$

We design the computation of l_i to be of negligible computational cost relative to the total forward pass and sufficiently expressive that our model could learn the token boundary heuristics that have been explored by prior work. Even without explicit training, models already encode rich information about the sequence in their internal representations: for example entropy (Nawrot et al., 2022; Pagnoni et al., 2024) has been shown to be mediated by directions in the internal representations of models soley trained on next-token prediction (Stolfo et al., 2024). Nonetheless, to express fixed striding strategies, as in MEGABYTE (Yu et al., 2023), we need to give the model access to some sliding window of preceding token boundaries.

Concretely, we compute the raw logit l_i^{raw} by applying a set of linear projections $W_j \in \mathbb{R}^{1\times d_{model}}$ to the byte-level representation X_i to get a base value for the logit and a series of terms conditional on each token boundary in the window. This operation is amenable to fast computation on modern hardware by pre-computing $W_k X_i$ for all i, k in a single matrix multiply and then performing a fast scan operation.

$$l_i^{raw} = W_0 X_i + \sum_{j=1}^{w} a_{i-j} W_k X_i. \tag{17}$$

In our experiments, we set a window size of $w = 8$.

With no constraint on the downsample rate, the model will elect to use the computationally expensive strategy of separating every byte with a token boundary. To avoid this, we need to push our model towards a target downsample rate $\bar{\pi}_{target}$, which we discuss further in §2.6. As in attention, we need to scale the raw logits to be approximately uniform at initialization, we further aid stability by adding a $\sigma^{-1}(\bar{\pi}_{target})$ term so that $p_i \approx \bar{\pi}_{target}$ at initialization.

$$l_i^{scaled} = \frac{l_i^{raw}}{D} + \sigma^{-1}(\bar{\pi}_{target}) \tag{18}$$

We choose a scaling factor of $D = 16$ and a target downsample rate of $\bar{\pi}_{target} = \frac{1}{5}$. To avoid numerical issues caused by exploding logits, we finally apply softcapping (Rivière et al., 2024).

$$l_i = \texttt{softcap}(l_i^{scaled}) \tag{19}$$

During evaluation, we skip this softcapping step, and simply set $l_i = l_i^{scaled}$

2.6 Downsample Rate Targeting

We a mechanism to keep the downsample rate close to $\bar{\pi}_{target}$ by applying an even pressure across all the logits $l_{i,b}$ in the batch. We prefer this to operating on the token boundary probabilities, which we find can have unstable results due to the uneven gradient magnitude of the sigmoid function. Specifically, we apply a negative or positive pressure to the whole batch mean logit $\bar{l} = \frac{1}{NB}\sum_{i,b} l_{i,b}$ if the mean token boundary probability $\bar{p} = \frac{1}{NB}\sum_{i,b} p_{i,b}$ exceeds or falls short of the target downsample rate respectively. We operationalize this with the loss:

$$\mathcal{L}^{target} = \bar{l} \cdot \texttt{detach}\left(\bar{p} - \bar{\pi}_{target}\right) \tag{20}$$

2.7 Full loss formula

We define the autoregressive losses, to learn the full model and the early exit model as:

$$\mathcal{L}^{auto} = -\sum_{i=0}^{N} \log p_\theta(x_i | x_{<i}, a_{<i}), \qquad \mathcal{L}^{early} = -\sum_{i=0}^{N} \log p_\theta^{early}(x_i | x_{<i}). \tag{21}$$

This gives our total loss calculation:

$$\mathcal{L} = \mathcal{L}^{auto} + \lambda_\pi \mathcal{L}^{\pi} + \lambda_{target}\mathcal{L}^{target} + \lambda_{early}\mathcal{L}^{early} \tag{22}$$

In our experiments, we set $\lambda_\pi = \lambda_{target} = 10^{-2}$ to encourage exploration and $\lambda_{early} = 10^{-1}$.

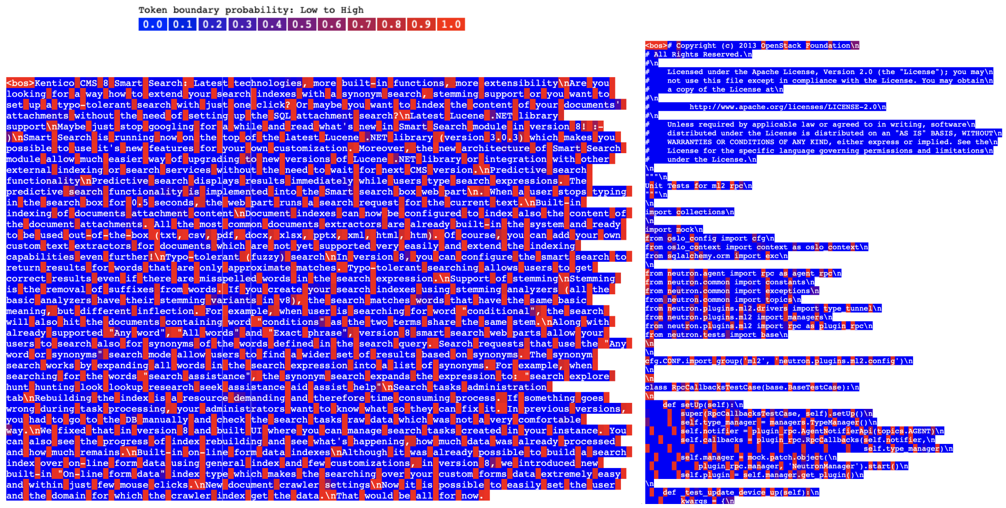

Figure 2: [Left] Token boundaries learned by our 147M-parameter model on a held-out sample of the `FineWeb` dataset. [Right] Token boundaries learned by our 90M-parameter model on a held-out sample of the `CodeParrot` dataset. Red and blue characters characters indicate high or low values of $\pi_\theta(a)$ respectively at the corresponding bytes.

3 Related Work

Prior work on end-to-end tokenization has largely relied on straight-through estimators (STEs) to enable gradient-based optimization of token boundaries, in contrast to our use of a score-function estimator. These approaches relax the discrete boundary variables a into continuous surrogates, differentiate through them, and then apply heuristic update rules to adjust $\pi_\theta(a)$ given the gradient $\frac{dL}{da}$. While such heuristics lack the theoretical guarantees enjoyed by score-function estimators §2.3, they have previously been proved effective in practice for learning mixture-of-experts (Shazeer et al., 2017) routing strategies, whereby there are too many decisions per token to provide a sufficiently low variance score function estimate.

The earliest demonstrations of STE-based boundary learning, such as the work of Godey et al. (2022), showed that token boundaries can be learned for bidirectional encoders by introducing a non-causal pooling mechanism that injects a small amount of every byte's representation into each token representation. Although elegant, this technique is fundamentally incompatible with the causal constraints of contemporary autoregressive LLMs, and adapting STEs to handle counterfactuals of the form "what if this boundary were placed one byte later?" has proven difficult.

Subsequent work has extended STE strategies to autoregressive settings. Nawrot et al. (2023), for instance, employ a straight-through estimator built on segmentation heuristics introduced by Bhati et al. (2021). We compare to this method in our experiments §4. Kallini et al. (2025) propose a more robust scheme that performs a full forward pass over all bytes using only soft downsampling during training, discarding low-scoring bytes only at inference time; however, this violates our **Efficiency** desideratum, since it preserves the full computational cost of byte-level processing during training. Concurrent with our work, Hwang et al. (2025) pursue a related STE-based approach using specialized up- and downsampling modules, which themselves introduce heuristic design choices—for example, initializing the downsampler to favor boundary placement at dissimilar byte transitions (Main Horse (pseudonym), 2025). Together, these methods highlight both the promise and the limitations of STEs for token-boundary learning, motivating alternative estimators with stronger theoretical footing.

We highlight prior work on scaling byte-level models and further innovations on the autoregressive U-net architecture in Appendix C.

Table 1: Performance of 147M parameter models on downstream natural language understanding tasks. Performance on `FineWeb Test` is in bits-per-byte, all others are in zero-shot accuracy.

	PIQA	HellaSwag	ARC-Easy	LAMBADA	FineWeb Test
Uniform	0.559	0.266	0.300	0.036	1.376
Dynamic (Nawrot et al.)	0.534	0.268	0.288	0.029	1.372
H-Net (Hwang et al.)	**0.576**	0.267	0.307	0.020	1.386
Ours	0.565	**0.271**	**0.308**	**0.086**	**1.297**

4 EXPERIMENTS

To evaluate our proposed score-function–based approach, we train autoregressive transformers on a filtered subset of the `FineWeb` dataset (Penedo et al., 2024), where we keep sequences of at least 4096 bytes. Subsequently, we truncate these sequences to exactly 4096 bytes. All models have approximately 147 million parameters and are trained with a target downsample rate of $\frac{1}{5}$. We select dataset size, batch size, and learning rate using the scaling laws derived by Porian et al. (2024). See Appendix D for further experimental details and hyperparameters.

We compare four dynamic tokenization strategies: a uniform baseline that selects exactly $4096\bar{\pi}_{target} = 819$, the straight-through estimators of Nawrot et al. (2023) Hwang et al. (2025) and our score function estimator.

Because our models are trained with several orders of magnitude less compute than contemporary frontier LLMs, the global downstream effects of learned tokenization state-of-the-art language modeling quality are difficult to assess directly. For example, the recently reported superiority autoregressive U-Nets over purely token-based architectures (Pagnoni et al., 2024; Hwang et al., 2025) was only observed to emerge at the $> 10^{21}$-`Flop` scale. For these reasons, our analysis focuses on qualitative comparison to other dynamic tokenization strategies.

4.1 LEARNED TOKENIZATION STRATEGIES FOR NATURAL LANGUAGE

We train 147-million parameter models on our filtered version of `FineWeb`. Figure 2 illustrates the token boundaries produced by our method on held-out `FineWeb` samples. Remarkably, despite having no inductive bias toward linguistic structure, the model reliably learns to place boundaries at whitespace-like characters, such as "\n" and " ". Additional samples and tokenization strategies learned by straight-through estimates are provided in Appendicies B.1 & B.2, B.3.

In appendix B.6, we further study the sensitivity of the learned tokenization strategy to the target downsample rate by varying the *tokenization aspect ratio:* we train models with $n = 2, 4, 6$ token-level transformer layers (sizes ranging from 20 to 40-million parameters) and a target downsample rate of $\bar{\pi}_{target} = \frac{1}{n}$, keeping the `FLOPs` -per-sequence fixed. The $n = 2$ model converges to a tokenization strategy of allocating almost exactly two bytes per token, with the exception of also drawing token boundaries at periods. By contrast, the $n = 4, 6$ models again discover whitespace-aligned boundaries and exhibit with varying degrees of chunking of longer words.

4.2 NATURAL LANGUAGE PERFORMANCE

In Appendix D we give further details on our experimental setup and `FLOPs` calculations. In Figure 10, we plot validation loss curves on our filtered `FineWeb` for the 147-million parameter training runs. We observe the qualitatively more semantically meaningful token boundaries that our method finds to translate to a consistent cross-entropy loss improvement as expended `FLOPs` varies over the training run. In table 1 we compare methods across a range of downstream natural language understanding benchmarks (Bisk et al., 2020; Zellers et al., 2019; Clark et al., 2018; Paperno et al., 2016), demonstrating improved language modelling performance.

4.3 PYTHON CODE

We train 90 million parameter models on the `CodeParrot` dataset (Hugging Face, 2022) of python code, using both our method and the straight-through estimate of Hwang et al. (2025). In Figure 2 we report that our model learns to draw token boundaries at the beginning of module names, space tokens to contain at least 2 bytes, and learns not to expend test-time compute on the Apache License, which is frequently repeated throughout the training dataset. Additional samples are provided in Appendix B.4 and tokenization strategies learned by H-Net are provided in appendix B.5. In Appendix D Figure 10 we plot the validation loss for both methods, finding that this intuitively appealing tokenization strategy leads to an improved downstream loss.

5 CONCLUSION

We propose a flexible parameterization of the problem of intra-architecture tokenization, which generalises prior proposed heuristics for this problem. We have demonstrated that applying our variance reductions yields a score function estimator that can optimize this parameterization to learn semantic boundaries §4.1 in text soley from optimizing the cross entropy and that doing so outperforms prior straight-through estimation based techniques §4.2.

REFERENCES

Dzmitry Bahdanau. The flops calculus of language model training, 2022. URL https://medium.com/@dzmitrybahdanau/the-flops-calculus-of-language-model-training-3b19c1f025e4.

Saurabhchand Bhati, Jesús Villalba, Piotr Żelasko, Laureano Moro-Velázquez, and Najim Dehak. Segmental contrastive predictive coding for unsupervised word segmentation. In *Interspeech 2021*, pp. 366–370, 2021. doi: 10.21437/Interspeech.2021-1874. URL https://www.isca-archive.org/interspeech_2021/bhati21_interspeech.html.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018. URL https://arxiv.org/abs/1803.05457.

Nathan Godey, Roman Castagné, Éric de la Clergerie, and Benoît Sagot. MANTa: Efficient gradient-based tokenization for end-to-end robust language modeling. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2859–2870, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.207. URL https://aclanthology.org/2022.findings-emnlp.207/.

Hugging Face. Codeparrot dataset. https://huggingface.co/datasets/transformersbook/codeparrot, 2022. URL https://huggingface.co/datasets/transformersbook/codeparrot/. Full Python code dataset used in Chapter 10 of *Natural Language Processing with Transformers*.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL https://arxiv.org/abs/2507.07955.

Julie Kallini, Shikhar Murty, Christopher D Manning, Christopher Potts, and Róbert Csordás. Mrt5: Dynamic token merging for efficient byte-level language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VYWBMq1L7H.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A Smith, and Yejin Choi. SuperBPE: Space travel for language models. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2503.13423.

Main Horse (pseudonym). H-nets. Blog post, 2025. URL https://main-horse.github.io/hnet/.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 1559–1571, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.117. URL https://aclanthology.org/2022.findings-naacl.117/.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353/.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, et al. Byte latent transformer: Patches scale better than tokens. *arXiv preprint arXiv:2412.09871*, 2024.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. 37:100535–100570, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/b6341525cd84f3be0ef203e4d7cd8556-Paper-Conference.pdf.

Morgane Rivière, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozinska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucinska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjösund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, and Lilly McNealus. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118, 2024. URL https://doi.org/10.48550/arXiv.2408.00118.

Jessica Rumbelow and Matthew Watkins. Solidgoldmagikarp (plus, prompt generation). LessWrong, 2023. URL https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems*, 28, 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/hash/de03beffeed9da5f3639a621bcab5dd4-Abstract.html.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Kevin Slagle. Spacebyte: Towards deleting tokenization from large language modeling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=KEe4IUp20I.

Alessandro Stolfo, Ben Wu, Wes Gurnee, Yonatan Belinkov, Xingyi Song, Mrinmaya Sachan, and Neel Nanda. Confidence regulation neurons in language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 125019–125049. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e21955c93dede886af1d0d362c756757-Paper-Conference.pdf.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 114147–114179. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/cf5a019ae9c11b4be88213ce3f85d85c-Paper-Conference.pdf.

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. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=JtBRnrlOEFN.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=X1xNsuKssb.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

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/.

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.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Lin Zheng, Xueliang Zhao, Guangtao Wang, Chen Wu, David Dong, Angela Wang, Mingran Wang, Yun Du, Haige Bo, Amol Sharma, Bo Li, Kejie Zhang, Changran Hu, Urmish Thakker, and Lingpeng Kong. Evabyte: Efficient byte-level language models at scale, 2025. URL https://hkunlp.github.io/blog/2025/evabyte.

A PROOFS

A.1 LOSS BREAKDOWN

$$
\begin{aligned}
\nabla_\theta \mathbb{E}_{a\sim\pi_\theta} \log p_\theta(y|a,x) =& \nabla_\theta \sum_a \log p_\theta(y|a,x)\pi_\theta(a|x) \\
=& \sum_a \left(\nabla_\theta \log p_\theta(y|a,x)\right)\pi_\theta(a|x) \\
&+ \sum_a \log p_\theta(y|a,x)\left(\nabla_\theta \pi_\theta(a|x)\right) \\
=& \nabla_\theta \sum_a \left(\nabla_\theta \log p_\theta(y|a,x) + \log p_\theta(y|a,x)\nabla_\theta \log \pi_\theta(a|x)\right)\pi_\theta(a|x) \\
=& \mathbb{E}_{a\sim\pi_\theta}(\underbrace{\nabla_\theta \log p_\theta(y|a,x)}_{\text{cross entropy loss}} + \underbrace{\log p_\theta(y|a,x)\nabla_\theta \log \pi_\theta(a|x)}_{\text{REINFORCE gradient}})
\end{aligned}
$$

B TOKENIZATION STRATEGIES

In the following plots, multi-byte characters, such as ü are rendered using the probability of the last byte representing the character.

B.1 OUR METHOD

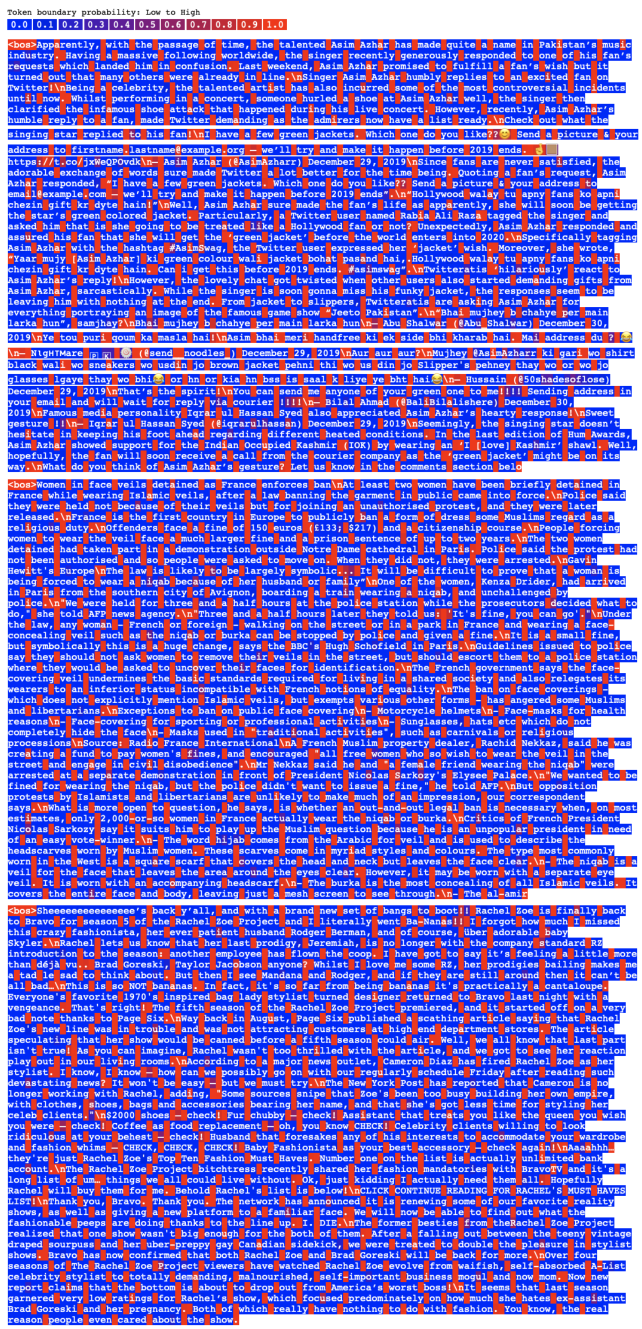

Figure 3: Token boundaries learned by our 147M-parameter model on a held-out sample of the FineWeb dataset. Red and blue characters characters indicate high or low values of $\pi_\theta(a)$ respectively at the corresponding bytes.

B.2 STRAIGHT-THROUGH ESTIMATOR FROM NAWROT ET AL. (2023)

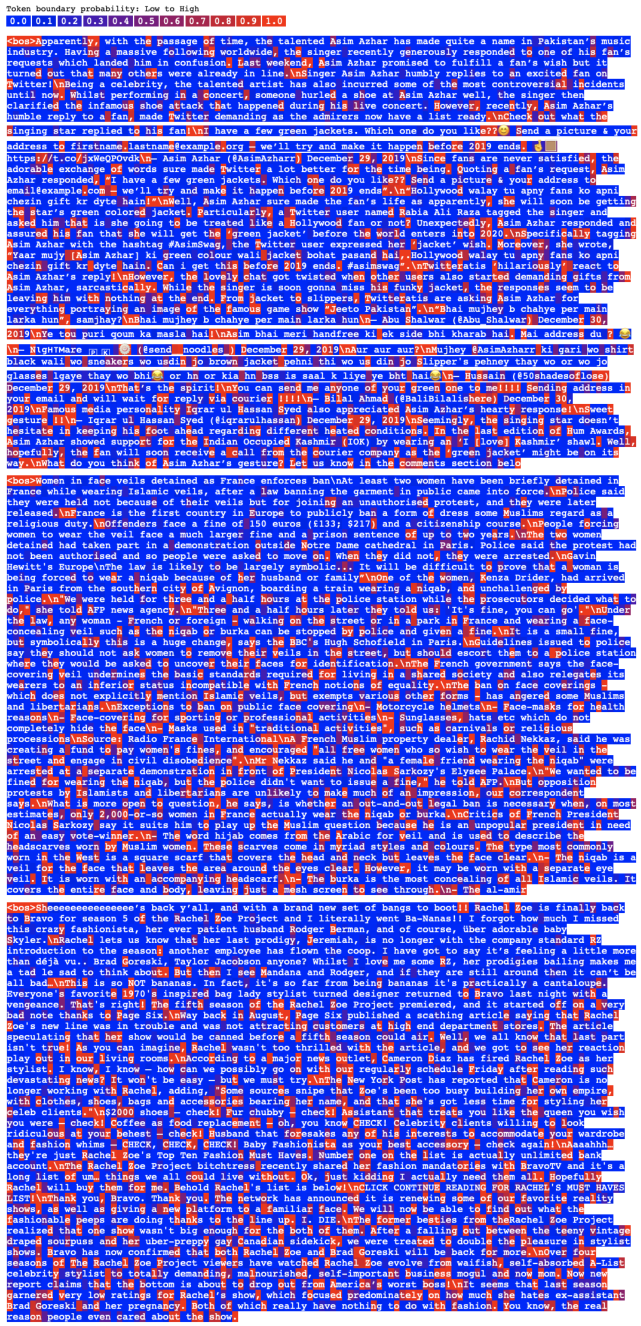

Figure 4: Token boundaries learned by a 147M-parameter model using the straight-through estimator of Nawrot et al. (2023), on held-out samples of the `FineWeb` dataset. Instead of the probabilities, here we plot the soft boundaries at the output of the Gumbel-Sigmoid. Red and blue characters characters indicate high or low values of $\hat{b}_t$ respectively at the corresponding bytes.

B.3 HNET HWANG ET AL. (2025)

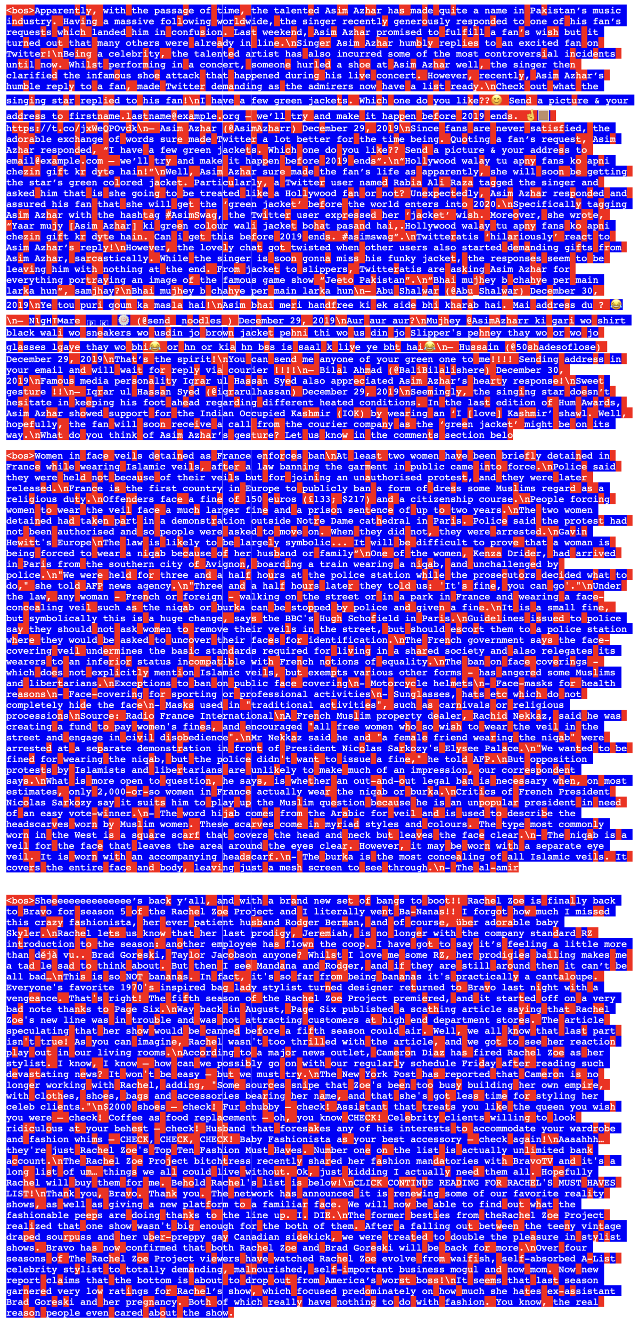

Figure 5: Token boundaries learned by a 147M-parameter model using the straight-through estimator of Hwang et al. Hwang et al. (2025), on held-out samples of the FineWeb dataset. Instead of the probabilities, here we plot the hard token boundaries. Red and blue characters characters indicate high or low values of $\hat{b}_t$ respectively at the corresponding bytes.

B.4 OUR METHOD ON CODEPARROT

Figure 6: Token boundaries learned by our 90M-parameter model on a held-out sample of the CodeParrot dataset. Red and blue characters characters indicate high or low values of $\pi_\theta(a)$ respectively at the corresponding bytes.

B.5 H-NET HWANG ET AL. (2025) ON CODEPARROT

Figure 7: Token boundaries learned by a 90M-parameter model using the straight-through estimator of Hwang et al. Hwang et al. (2025), on held-out samples of the CodeParrot dataset. Instead of the probabilities, here we plot the hard token boundaries. Red and blue characters characters indicate high or low values of $\hat{b}_t$ respectively at the corresponding bytes.

B.6 VARYING THE DOWNSAMPLE RATE

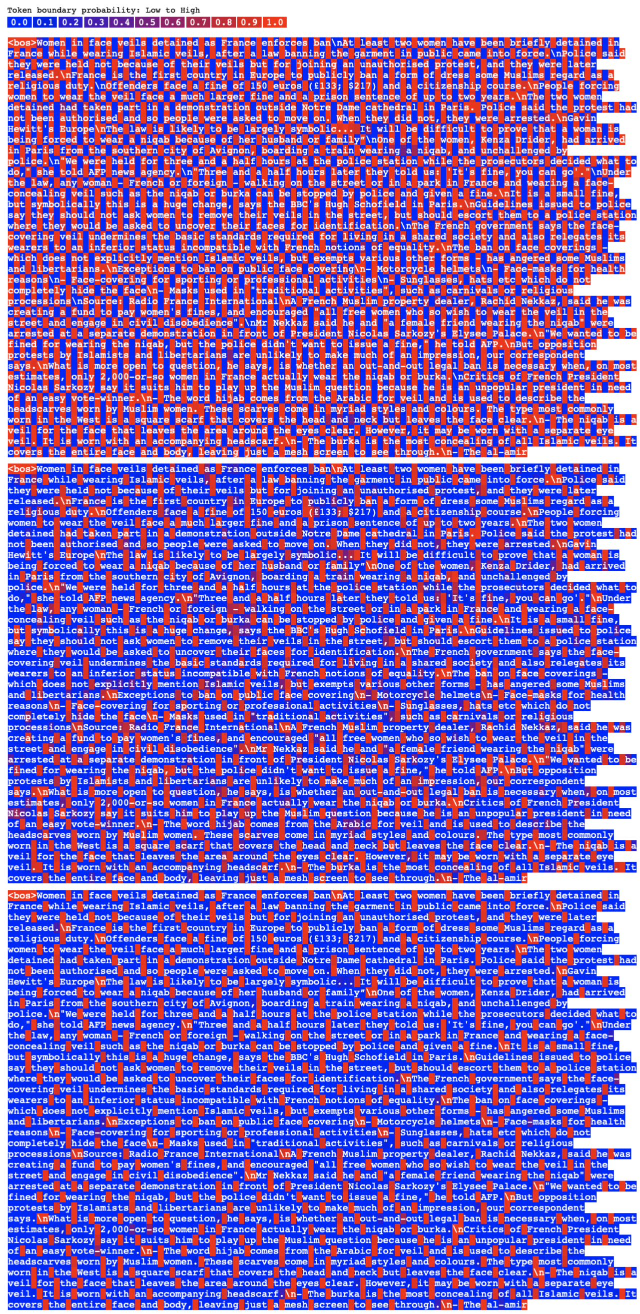

Figure 8: Token boundaries learned by an array of models using our method with varying target downsample rates $\bar{\pi}_{target} = \frac{1}{2}, \frac{1}{4}, \frac{1}{6}$ (top, middle, bottom, respectively) on a held-out sample of the FineWeb dataset. Red and blue characters characters indicate high or low values of $\pi_\theta(a)$ respectively at the corresponding bytes.

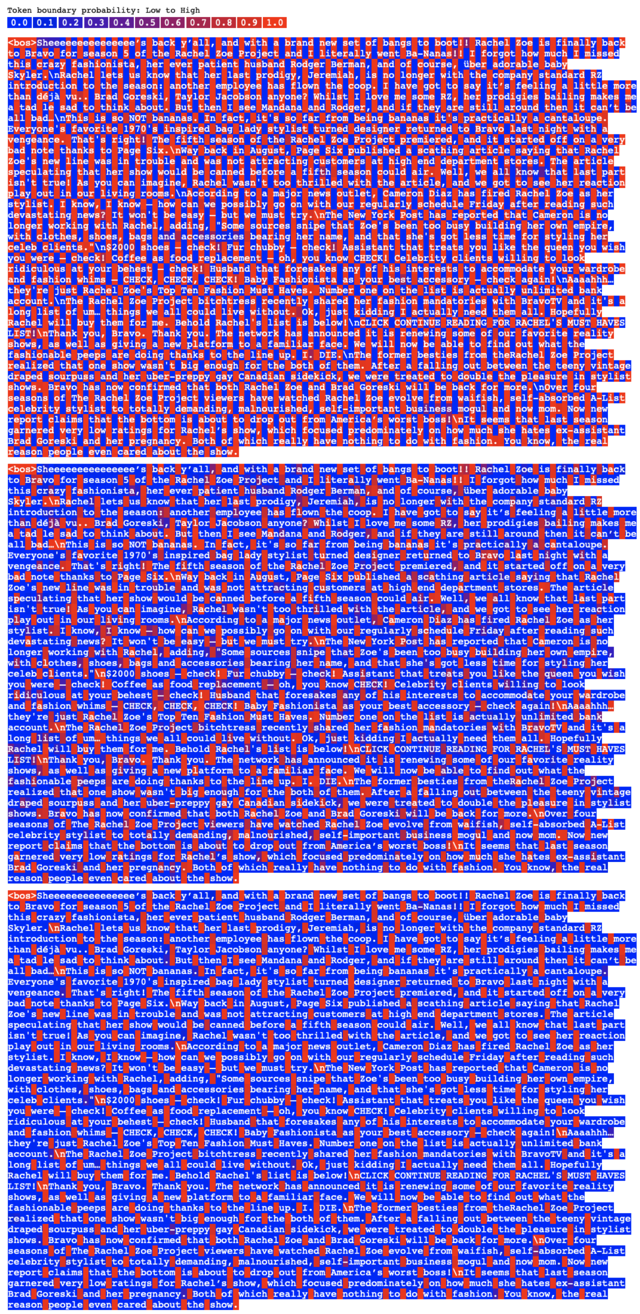

Figure 9: Token boundaries learned by an array of models using our method with varying target downsample rates $\bar{\pi}_{target} = \frac{1}{2}, \frac{1}{4}, \frac{1}{6}$ (top, middle, bottom,) of on a held-out sample of the `FineWeb` dataset. Red and blue characters characters indicate high or low values of $\pi_\theta(a)$ respectively at the corresponding bytes.

C Continued Related Work

C.1 Scaling byte-level language models with Tokenization Heuristics

Byte-level language models have recently been shown to scale competitively with tokenized transformers, with ByT5 (Xue et al., 2022) demonstrating that a pure byte-level architecture can match token-level performance, though at the cost of quadratic attention that makes long-sequence scaling impractical. Subsequent work has addressed this limitation along two main directions: leveraging subquadratic attention mechanisms (Wang et al., 2024; Zheng et al., 2025), and reducing sequence length through hierarchical or pooled representations. The latter includes fixed-stride downsampling approaches (Tay et al., 2022; Nawrot et al., 2022; Yu et al., 2023), as well as methods that segment byte streams at more semantically meaningful boundaries. Nawrot et al. (2023) first explored pooling at whitespace characters, later scaled by Slagle (2024) to outperform fixed-stride schemes. In parallel, Nawrot et al. also introduced entropy-based boundary detection (Nawrot et al., 2023), leveraging spikes in next-byte entropy that correlate with semantic breaks; this idea was further scaled by Pagnoni et al. (2024), who showed that entropy-guided downsampling can enable byte-level models to surpass token-level baselines at the 10^{22}-FLOP scale.

While these methods demonstrate the promise of the autoregressive U-net architecture at scale and highlight some desirable properties that a tokenization mechanism for this architecture would have, all rely on heuristics, breaking the **End-to-end tokenizer training** desideratum in our setup.

C.2 Architectural Innovations

We focus on the method of deciding token boundaries, and thus use a simple architecture for the purpose of not confounding our experiments. Nonetheless, we would like to highlight the following architectural innovations that have been proposed to improve autoregressive U-nets:

- Using a vocabulary of byte-level n-grams which are added to the input embeddings to improve information flow Pagnoni et al. (2024).
- Using cross-attention between the byte-level and token-level transformers Pagnoni et al. (2024).
- Using subquadratic sequence-to-sequence blocks at the byte level Hwang et al. (2025).
- Using multiple levels of tokenization Hwang et al. (2025).
- Smoothing between tokenization levels in upsampling Hwang et al. (2025).

We would like to note that all of these methods could be used with our method to learn token boundaries.

Model size	147M	90M	20-40M
`embedding_dim`	768	768	512
`num_heads`	12	12	8
`n_down_layers`	4	4	4
`n_mid_layers`	12	6	n
`n_up_layers`	4	4	4
`learning_rate`	1.5×10^{-3}	2×10^{-3}	3×10^{-3}
`effective_batch_size`	128	128	64
`warmup_bytes`	4.5×10^{8}	2.5×10^{8}	1.1×10^{8}
`training_bytes`	6×10^{9}	3.4×10^{9}	1.5×10^{9}
`downsample_rate_target`	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{n}$

Table 2: The hyperparameters used for our runs varying the token boundary selection method used on natural language (left) python code (middle) and the varying tokenization aspect ratio (right)

D FURTHER EXPERIMENTAL DETAILS

We train models on a total of `training_bytes` bytes with a cosine learning rate schedule with a warmup of `warmup_bytes`and a maximum learning rate of `learning_rate`. We use the AdamW optimizer with default parameters $(\beta_1, \beta_2) = (0.9, 0.999)$. Per gradient update, we performa a forward & backward pass on `effective_batch_size` sequences of length 4096 .

We use a decoder-only transformer architecture, with `n_down_layers` and n_up_layers byte-level transformer decoder layers with sliding window attention with window size 64 before and after the token-level backbone which consists of `n_mid_layers` token-level transformer decoder layers with full causal attention. All attention layers have num_heads heads. We make the model dimension the same for byte and token-level layers, such that $d_{enc} = d_{mid} = d_{dec} =$ `embedding_dim`. Byte-level and token-level layers have an MLP hidden dimension of `embedding_dim` and $4 \times$ `embedding_dim` respectively, making the flops-per-layer roughly consistent. We closely follow the transformer architectural choices of Gemma 2 Rivière et al. (2024), with GeGLU non-linearity, Rotary Position Embeddings, post and pre- RMSNorm and Logit soft-capping.

See table 2 for the hyperparamters used in our 147-M parameter experiment and our experiments varying the token aspect ratio.

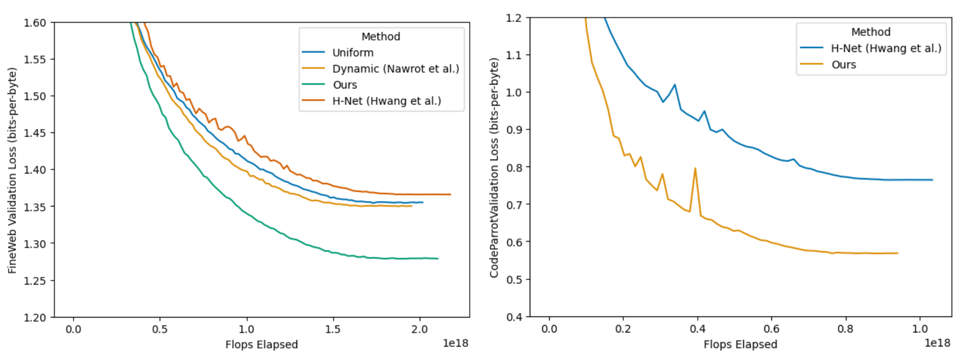

Figure 10: [left] Flops vs validation loss curves for 147M parameter models, trained on `FineWeb`, measured in bits-per-byte. Uniform random in blue, Straight through estimates in orange, ours in green. Validation loss values at 1.95×10^{18} `FLOPs` are 1.355, 1.350, 1.36 and 1.279 bits-per-byte respectively. [right] Flops vs validation loss curves for 90M parameter models, trained on `CodeParrot`, measured in bits-per-byte. H-Net in blue ours in orange. Validation loss values at 0.95×10^{18} `FLOPs` are 0.769 and 0.568 bits-per-byte respectively

Following Kaplan et al. (2020), we estimate the number of `FLOPs` in the model training run as approximately the number of `FLOPs` used for the matrix parameters in the forward/backward pass (which is the dominant source for LLMs), which takes a value of 6 `FLOPs` per parameter per byte or token, the latter depending on which level the layer operates on (Bahdanau, 2022). We assume that the embedding operation is performed using an efficient lookup. Small deviations in `FLOPs` per batch for each method occur as a result of variations in the max token sequence length in the batch.