TURNIP: TOKENIZERS ARE SECRETLY CONTEXT COMPRESSORS FOR TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer language models process text as sequences of tokens, and their self-attention cost grows quadratically with the number of tokens. As a result, tokenization largely determines the computational cost as contexts grow. However, the de facto standard tokenizer byte-pair encoding (BPE) produces token sequences longer than the information content of text requires. In this paper, we formulate tokenization as a constrained predictive-compression problem. Under this formulation, we show that the compression of BPE is limited by two constraints: its tokens are lossless and context-independent. Guided by this theory, we introduce Turnip (**t**he ne**ur**al toke**ni**zer **p**ipeline), a neural tokenizer that removes both constraints and provides direct control over the compression ratio. Turnip is model-agnostic and can be used with existing pretrained language models. On different models and benchmarks, Turnip nearly doubles the compression ratio of BPE while matching its generation quality.

1 INTRODUCTION

Transformer language models have become a general-purpose foundation for understanding and generating text (Brown et al., 2020; Touvron et al., 2023). They access text only through a tokenizer, which converts the input into a sequence of tokens and the model's predictions back into text. Since each token occupies one position and self-attention compares every pair of positions (Vaswani et al., 2017), the cost of the language model grows quadratically with the number of tokens. How many bytes each token covers therefore determines how expensive long contexts are to process.

The de facto standard tokenizer is byte-pair encoding (BPE) (Sennrich et al., 2016; Radford et al., 2019; Brown et al., 2020; Touvron et al., 2023), which merges frequent adjacent byte sequences into the tokens of a fixed vocabulary. Despite its popularity, BPE compresses the context inefficiently. Shannon (1951) estimated that English carries about 1 bit of information per letter. In modern BPE tokenizers with more than 100K entries, each token can carry more than 16 bits. Yet a token covers only about 4 characters on average (Grattafiori et al., 2024) and thus carries only about 4 bits of information, leaving substantial room for compression.

We attribute this inefficiency to two constraints of BPE. First, its tokens are *lossless* and identify the input exactly even when the context contains information that does not help predict what comes next (Shalizi & Crutchfield, 2001). Second, they are *context-independent* and are always detokenized into the same bytes (Zouhar et al., 2023), even though the previous tokens could help detokenize them. Byte-level models relax both constraints by grouping bytes into patches whose representations are learned inside the model (Yu et al., 2023; Slagle, 2024; Pagnoni et al., 2025; Hwang et al., 2026). However, their patch representations are learned jointly with the model through end-to-end training, which ties their compression to a model trained with them rather than providing a tokenizer that existing models can use. In addition, their patching rules are based on heuristics, so the theoretical limit of how far a tokenizer can compress the context remains an open question.

In this work, we formulate tokenization as a constrained predictive-compression problem (Tishby et al., 1999; Marzen & Crutchfield, 2016) that maximizes the number of bytes per token while preserving the best achievable next-byte loss. Under this formulation, we prove that both constraints of BPE cost compression. Specifically, a lossless tokenizer can compress strictly less than a lossy one

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

that still attains the best achievable loss. Moreover, context-independent detokenization is suboptimal at the compression limit under next-token prediction.

Guided by this theory, we introduce Turnip (**t**he ne**ur**al toke**ni**zer **p**ipeline) as a neural tokenizer that removes both constraints. It splits the byte stream into segments whose average length is set by a budget and encodes each segment into a continuous token without requiring exact reconstruction. Each byte is then predicted from the token history rather than from a single token. Unlike byte-level models, Turnip is trained through next-byte prediction without a language model and can therefore be used with existing pretrained language models. In experiments on multiple such models, Turnip nearly doubles the compression ratio of BPE while matching its generation quality. Its speedup over BPE also grows with both the context length and the size of the language model.

2 METHOD

We first formulate tokenization as a constrained predictive-compression problem. This formulation shows that BPE loses compression to two constraints: context-independent detokenization and lossless tokenization. Based on this analysis, we propose Turnip and compare its computational cost with that of BPE.

2.1 TOKENIZATION AS PREDICTIVE COMPRESSION

A tokenizer converts a byte stream into the token sequence that a language model processes. Let p denote the distribution of a stationary byte source $\mathbf{X} = (X_0, X_1, \ldots)$ over the byte alphabet $\mathcal{X} = \{0, \ldots, 255\}$. A tokenization function ε maps each byte history $\mathbf{X}_{<t}$ to a token sequence $\varepsilon(\mathbf{X}_{<t}) \in \mathcal{C}^*$, where $\mathcal{C}$ is a fixed token vocabulary. Since t bytes are represented by $|\varepsilon(\mathbf{X}_{<t})|$ model positions, we define the compression ratio of ε as

$$\rho(\varepsilon) := \lim_{n\to\infty} \frac{n}{\mathbb{E}_p[|\varepsilon(\mathbf{X}_{<n})|]}. \tag{1}$$

A model with parameters θ then predicts the next byte from the tokenized history with distribution $q_\theta(\cdot \mid \varepsilon(\mathbf{X}_{<t}))$. Predictors may also depend on the position t, which we omit from the notation. We further assume that all limits in this section exist. To compare tokenization functions under the same prediction target, we measure the next-byte loss as the average negative log-likelihood (Delétang et al., 2024)

$$\mathcal{L}(\theta, \varepsilon) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} \mathbb{E}_p[-\log_2 q_\theta(X_t \mid \varepsilon(\mathbf{X}_{<t}))]. \tag{2}$$

Using this loss, we formulate predictive compression as the constrained problem

$$\begin{aligned} \max_{\varepsilon} \quad & \rho(\varepsilon) \\ \text{s.t.} \quad & \inf_{\theta'} \mathcal{L}(\theta', \varepsilon) = \inf_{\theta', \varepsilon'} \mathcal{L}(\theta', \varepsilon'). \end{aligned} \tag{3}$$

In other words, we seek the most compressive tokenizer that does not increase the best achievable next-byte loss. The following lemma shows how ε affects both sides of this problem.

Lemma 1. *For any deterministic tokenization function ε whose output length grows at most linearly with its input length and any model q_θ, we have*

$$\rho(\varepsilon) \le \frac{\log_2 |\mathcal{C}|}{h^{\mathrm{rep}}(\varepsilon)} \quad \textit{and} \quad \mathcal{L}(\theta, \varepsilon) = \underbrace{h^{\mathrm{src}}}_{\textit{source}} + \underbrace{\ell^{\mathrm{tok}}(\varepsilon)}_{\textit{tokenization}} + \underbrace{\ell^{\mathrm{lm}}(q_\theta; \varepsilon)}_{\textit{model}}, \tag{4}$$

where

$$h^{\mathrm{rep}}(\boldsymbol{\varepsilon}) := \lim_{n\to\infty} \frac{1}{n} H(\boldsymbol{\varepsilon}(\mathbf{X}_{<n}))\,, \tag{5}$$

$$h^{\mathrm{src}} := \lim_{n\to\infty} \frac{1}{n} H(\mathbf{X}_{<n}), \tag{6}$$

$$\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} I(X_t; \mathbf{X}_{<t} \mid \boldsymbol{\varepsilon}(\mathbf{X}_{<t}))\,, \tag{7}$$

$$\ell^{\mathrm{lm}}(q; \boldsymbol{\varepsilon}) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} \mathbb{E}_p[D_{\mathrm{KL}}(p(\cdot \mid \boldsymbol{\varepsilon}(\mathbf{X}_{<t})) \,\|\, q(\cdot \mid \boldsymbol{\varepsilon}(\mathbf{X}_{<t})))]\,. \tag{8}$$

Proof. See Appendix A.1. □

Lemma 1 reveals a trade-off between the two sides. For a sufficiently expressive model, the tokenizer affects the next-byte loss only through $\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon})$, which measures the predictive information that the tokens discard. Meanwhile, the compression ratio is bounded by the information $h^{\mathrm{rep}}(\boldsymbol{\varepsilon})$ that the tokens retain per byte because each token carries at most $\log_2 |\mathcal{C}|$ bits. An ideal tokenizer should therefore keep the information that helps predict the next byte and discard the rest.

2.2 Suboptimality of BPE

Problem (3) assumes that the model predicts the next byte directly from the tokens. However, language models are trained to predict the next token by minimizing

$$\mathcal{L}^{\mathrm{next}}(r; \boldsymbol{\varepsilon}) := \lim_{n\to\infty} \frac{1}{n} \sum_{m=0}^{n-1} \mathbb{E}_p[-\log_2 r(C_m \mid \mathbf{C}_{<m})], \tag{9}$$

where $r(\cdot \mid \mathbf{C}_{<m})$ denotes the model's distribution over the next token C_m given the first m tokens $\mathbf{C}_{<m}$. This loss requires the tokenized histories to extend a single token stream $\mathbf{C}$, so that $\boldsymbol{\varepsilon}(\mathbf{X}_{<t}) = \mathbf{C}_{<m}$ when the first t bytes yield m tokens. To connect this token-level model to the next-byte loss, we introduce a *detokenization function* $\boldsymbol{\delta} : \mathcal{C}^* \to \Delta(\mathcal{X})$ that maps the token history to a distribution over the next byte. Here $\Delta(\mathcal{X})$ denotes the set of probability distributions over $\mathcal{X}$. A tokenizer is thus a pair $\tau = (\boldsymbol{\varepsilon}, \boldsymbol{\delta})$ of tokenization and detokenization functions, which defines the byte prediction model

$$q_{r,\tau}(x \mid \boldsymbol{\varepsilon}(\mathbf{X}_{<t})) := \sum_{c\in\mathcal{C}} r(c \mid \boldsymbol{\varepsilon}(\mathbf{X}_{<t}))\boldsymbol{\delta}(x \mid \boldsymbol{\varepsilon}(\mathbf{X}_{<t}) \,\|\, (c)), \tag{10}$$

where $\|$ denotes concatenation. BPE maps each token to the same byte string regardless of its history (Radford et al., 2019; Zouhar et al., 2023). Its detokenization function is therefore *context-independent*: $\boldsymbol{\delta}(\cdot \mid \mathbf{C}_{<m} \,\|\, (c)) = \boldsymbol{\delta}(\cdot \mid (c))$. We now show that this constraint leads to suboptimal byte prediction under next-token training once tokenization attains the compression bound.

Proposition 2 (Suboptimality of context-independent detokenization)**.** *If $\boldsymbol{\varepsilon}$ attains the compression bound in Lemma 1, $\rho(\boldsymbol{\varepsilon}) = (\log_2 |\mathcal{C}|)/h^{\mathrm{rep}}(\boldsymbol{\varepsilon})$, and its token count concentrates, $|\boldsymbol{\varepsilon}(\mathbf{X}_{<n})|/n \to 1/\rho(\boldsymbol{\varepsilon})$ in probability, then the uniform predictor $r^*(\cdot \mid \mathbf{C}_{<m}) = \mathrm{Uniform}(\mathcal{C})$ minimizes the next-token loss $\mathcal{L}^{\mathrm{next}}(\cdot; \boldsymbol{\varepsilon})$. In addition, there exists a detokenization function $\boldsymbol{\delta}^*$ such that every context-independent $\boldsymbol{\delta}^0$ satisfies*

$$\ell^{\mathrm{lm}}(q_{r^*,\boldsymbol{\delta}^0}; \boldsymbol{\varepsilon}) \geq H(X_0) - h^{\mathrm{src}} - \ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}) > \ell^{\mathrm{lm}}(q_{r^*,\boldsymbol{\delta}^*}; \boldsymbol{\varepsilon}) = 0 \tag{11}$$

whenever $\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}) < H(X_0) - h^{\mathrm{src}}$.

Proof. See Appendix A.2. □

Intuitively, tokens at the compression bound are as unpredictable as possible, so the uniform distribution already minimizes the next-token loss and a stronger token-level language model cannot help. The next-byte distribution must then be recovered from the context by the detokenization function, which a context-independent one cannot do.

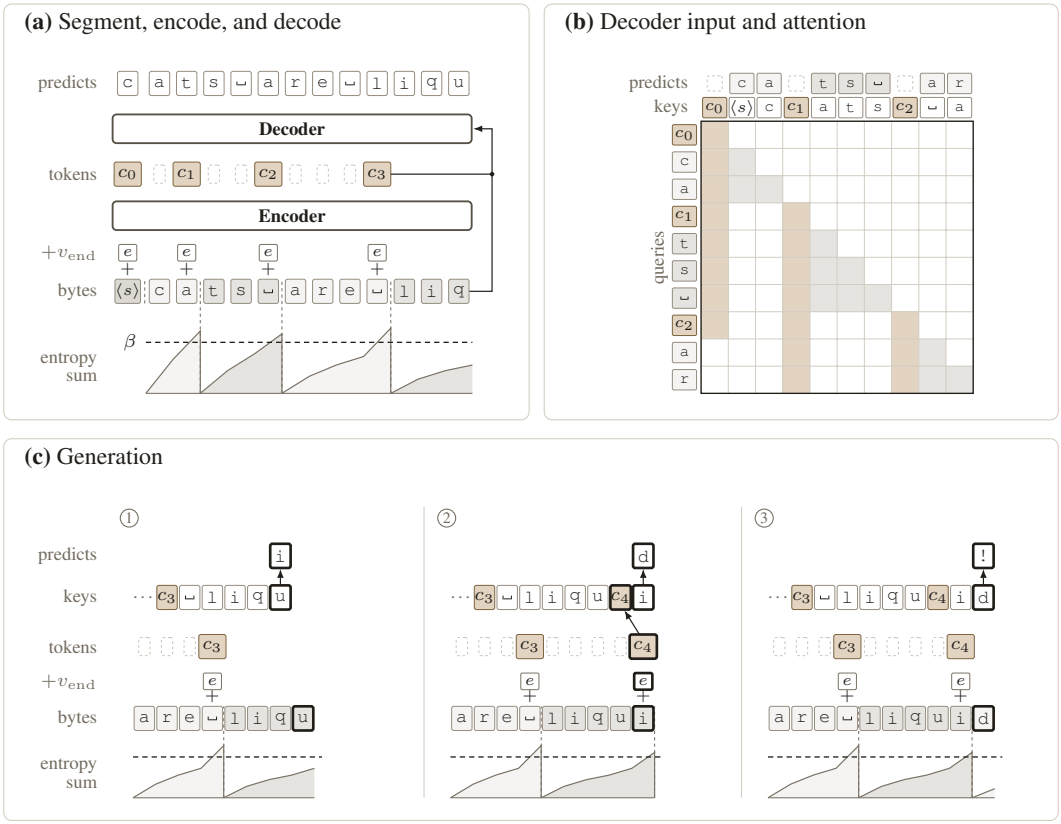

Figure 1: Overview of Turnip on the text "cats are liquid". (a) Segmentation, encoding, and decoding. (b) Decoder input and attention mask for $w_c = 2$, where byte rows are labeled by the byte they predict. (c) Generation with one byte per step, where bold cells are new.

BPE is also *lossless*: its tokens uniquely identify the bytes they encode (Radford et al., 2019; Zouhar et al., 2023), so $H(\mathbf{X}_{<t} \mid \varepsilon(\mathbf{X}_{<t})) = 0$ for every t. Proposition 3 shows that this constraint limits the compression ratio.

Proposition 3 (Suboptimality of lossless tokenization)**.** *Suppose $|\mathcal{C}| \geq 3$ and the model family contains every predictor $q : \mathcal{C}^* \to \Delta(\mathcal{X})$. Then there exist a source distribution p and a tokenization function ε such that ε attains the best achievable next-byte loss,*

$$\inf_{\theta'} \mathcal{L}(\theta', \varepsilon) = \inf_{\theta', \varepsilon'} \mathcal{L}(\theta', \varepsilon'), \tag{12}$$

and every lossless tokenization function ε_0 whose output length grows at most linearly with its input length satisfies $\rho(\varepsilon_0) < \rho(\varepsilon)$.

Proof. See Appendix A.3. □

A lossless tokenizer thus keeps even the information that an ideal tokenizer would discard. Since its tokens determine the history, $h^{\mathrm{rep}}(\varepsilon) \geq h^{\mathrm{src}}$ and Lemma 1 caps its compression ratio at $\log_2 |\mathcal{C}|/h^{\mathrm{src}}$. By contrast, a lossy tokenizer can exceed this cap without increasing the best achievable next-byte loss.

2.3 Turnip: the neural tokenizer pipeline

Guided by Problem (3), we design the tokenizer for next-byte prediction rather than for exact reconstruction. Specifically, Turnip learns a tokenization function ε_ϕ and a detokenization function $\boldsymbol{\delta}_\theta$ that together form the byte predictor:

$$q_\theta(X_t \mid \varepsilon_\phi(\mathbf{X}_{<t})) = \boldsymbol{\delta}_\theta(X_t \mid \varepsilon_\phi(\mathbf{X}_{<t})). \tag{13}$$

In light of Proposition 2, $\boldsymbol{\delta}_\theta$ conditions on the token history rather than on a single token. Following Proposition 3, we let $\boldsymbol{\varepsilon}_\phi$ be lossy so that it can exceed the compression cap of lossless tokenizers. Turnip implements $\boldsymbol{\varepsilon}_\phi$ with an entropy model and an encoder, and $\boldsymbol{\delta}_\theta$ with a decoder.

Segmentation. Turnip first splits the byte stream into segments that carry similar amounts of information (Figure 1a, bottom). Since a token of fixed size can retain only a limited amount of information, a segment that carries more than its token can hold increases $\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}_\phi)$ in Lemma 1. To balance the segments, a causal entropy model (Pagnoni et al., 2025) scores every byte x_t by the entropy s_t of its predicted distribution for x_{t+1}. This entropy model is a one-layer sliding-window Transformer trained to predict the next byte. The scores are accumulated within the current segment until the sum reaches a budget β or the segment reaches a maximum length w_b. Formally, a segment that starts at byte a ends at byte

$$b = \min\Big\{b' \geq a : \sum_{u=a}^{b'} s_u \geq \beta \text{ or } b' - a + 1 = w_b\Big\}. \tag{14}$$

We set $w_b = 256$ and prepend a start symbol $\langle s\rangle$ as segment 0. We write e_k for the position of the last byte of segment k, so that $x_{e_0} = \langle s\rangle$. For each target ratio, we set β such that the average segment length on a sample of the training data matches the target.

Encoder. The encoder then turns each segment into a token (Figure 1a, middle). It is a causal Transformer that runs over the whole byte stream rather than per segment, and its sliding-window attention lets a token draw on bytes before its segment. To mark segment boundaries, we add a learned embedding v_{end} to the input at the last byte of every segment. Only the hidden state at x_{e_k} is kept and serves as the token c_k.

Decoder. The decoder maps the tokens back to next-byte predictions (Figure 1b). It is a causal Transformer whose input interleaves the tokens with the bytes shifted right by $\langle s\rangle$. Each token c_k is inserted right before the position that reads x_{e_k}, which predicts the first byte of segment $k+1$. In every layer, a byte position attends to the w_c most recent token positions and to the byte positions after the last token. A token position instead attends to itself and to the w_c token positions before it. As a result, the decoder predicts each byte x_t of segment $k+1$ from the context

$$\boldsymbol{\varepsilon}_\phi(\mathbf{x}_{<t}) = \big(c_0, \ldots, c_k,\, x_{e_k}, \ldots, x_{t-1}\big). \tag{15}$$

Earlier bytes reach the decoder only through the tokens. Here the vocabulary $\mathcal{C}$ contains both the token vectors and the bytes.

Tokenizer training and language modeling. The entropy model is trained beforehand and kept fixed. The encoder and decoder are then trained with the next-byte loss $\mathcal{L}$ without a language model. The resulting tokenizer can be used with any pretrained language model after short fine-tuning. Specifically, we insert the language model between the encoder and the decoder, where it runs on the token sequence $c_0, c_1, \ldots$. A linear adapter maps each token into the language model, and a second linear adapter A_{out} maps the model's output h_k at token k back to the decoder. The decoder reads $\tilde{c}_k = c_k + A_{\mathrm{out}} h_k$ in place of c_k. We initialize A_{out} to zero so that fine-tuning starts from the tokens that the decoder was trained on, and we fine-tune the language model and adapters together with the encoder and decoder on $\mathcal{L}$ (Section 3.1).

Generation. Turnip generates one byte per step (Figure 1c). After a byte is sampled, the entropy model reads it and adds its score to the running sum. If the segment ends, the encoder emits the token c_k and the language model takes one step to produce $\tilde{c}_k$ for the decoder. The decoder input then receives $\tilde{c}_k$, if any, followed by the sampled byte, and the decoder predicts the next byte from it. The language model thus runs only once per segment.

2.4 Computational cost

We now compare the computational cost of Turnip with that of BPE. On m tokens, a Transformer language model costs about λm operations in its linear components (the attention projections and the feed-forward networks) and μm^2 operations in self-attention. Given n bytes, BPE and Turnip

feed n/ρ_{BPE} and n/ρ tokens to the language model, respectively. Although the tokenizer of Turnip processes every byte, each of its positions attends to a bounded number of positions. Its cost is therefore a constant κ per byte, which gives

$$F_{\mathrm{BPE}}(n) \approx \lambda \frac{n}{\rho_{\mathrm{BPE}}} + \mu \frac{n^2}{\rho_{\mathrm{BPE}}^2}, \tag{16}$$

$$F_{\text{Turnip}}(n) \approx \lambda \frac{n}{\rho} + \mu \frac{n^2}{\rho^2} + \kappa n. \tag{17}$$

Let $\gamma = \rho/\rho_{\mathrm{BPE}}$. Compared with BPE, the linear cost of the language model falls by a factor of γ and its attention cost by a factor of γ^2. As n grows, the quadratic terms dominate and

$$\lim_{n\to\infty} \frac{F_{\mathrm{BPE}}(n)}{F_{\text{Turnip}}(n)} = \gamma^2. \tag{18}$$

When the tokenizer is much cheaper than the language model ($\kappa \ll \lambda/\rho$), the reduction lies between γ and γ^2 at every length. More importantly, κ does not depend on the language model while λ and μ grow with its size. The benefit of compression therefore grows with the size of the language model. In addition, the key-value cache of the language model shrinks by a factor of γ, while that of the tokenizer stays bounded.

3 Experiments

3.1 Settings

Datasets. We train all tokenizers and fine-tune all language models on Dolci-Think-SFT-32B (Team Olmo et al., 2025)[1]. This English supervised fine-tuning corpus pairs user prompts with long reasoning traces and responses, which makes it well suited to studying context compression. Two benchmarks then measure generation quality. (1) **IFEval** (Zhou et al., 2023) contains 541 prompts with instructions that can be checked automatically. For example, a prompt may ask for a poem in four marked sections that ends with a given phrase. These exact checks make IFEval a stringent test of whether the tokenizer preserves the details of the prompt. (2) **GPQA** Diamond (Rein et al., 2024) contains 198 four-choice questions in biology, physics, and chemistry written by domain experts. Following common practice (Biderman et al., 2024), models answer with a zero-shot chain of thought (Kojima et al., 2022). The tokenizer thus also compresses their long reasoning.

Evaluation metrics. For IFEval, we report strict prompt-level accuracy, which requires a response to follow all instructions of its prompt. We also report strict instruction-level accuracy, which measures the fraction of instructions followed. For GPQA, we follow lm-evaluation-harness and take the last parenthesized option letter in the output as the answer. Outputs without such a letter are counted as wrong. All scores are averaged over three sampling seeds.

In addition to accuracy, we measure the compression ratio of each tokenizer as the number of bytes per token averaged over the conversations of a held-out split.

Baselines. We compare Turnip with two baselines. (1) **BPE** is the standard tokenizer, which is lossless and context-independent. Its 160K-entry vocabulary is trained on the same corpus and paired with a randomly initialized embedding table, so that both tokenizers are new to the language model. (2) **BLT** (Pagnoni et al., 2025) resembles the workflow of Turnip in chunking bytes into segments. However, it is integrated with a predefined model rather than being a model-agnostic tokenizer, so the released BLT-1B is evaluated as a separate model. Its segmentation threshold is calibrated so that its patches are about 4 (the released default), 8, and 12 bytes long (CR4, CR8, and CR12).

Implementation details. We use Qwen2.5-0.5B and 1.5B (Qwen et al., 2024) as the language models. Since they are pretrained with a BPE tokenizer, replacing it tests whether a new tokenizer can be used with an existing model. Their two sizes further show how the effect of compression

[1] https://huggingface.co/datasets/allenai/Dolci-Think-SFT-32B

Table 1: Accuracy (%) on IFEval and GPQA Diamond and compression ratio of the released BLT-1B at three entropy thresholds and of BPE and Turnip on Qwen2.5-0.5B and 1.5B. Bytes/token is bytes per patch for BLT. Accuracies are mean ± standard deviation.

		IFEval		
Method	Bytes/token	Prompt	Instruction	GPQA Diamond
		BLT-1B		
BLT (CR4)	4.36	11.1 ± 0.5	22.0 ± 0.5	12.1 ± 4.8
BLT (CR8)	8.24	12.7 ± 1.4	23.7 ± 0.5	9.9 ± 1.8
BLT (CR12)	12.49	15.2 ± 1.4	27.5 ± 0.9	12.8 ± 3.4
		Qwen2.5-0.5B		
BPE	4.39	42.9 ± 1.3	55.0 ± 0.9	22.4 ± 2.0
Turnip (CR4)	4.11	44.5 ± 1.1	56.3 ± 0.2	24.7 ± 1.8
Turnip (CR8)	8.00	42.8 ± 1.5	55.7 ± 1.3	22.2 ± 3.6
Turnip (CR12)	12.27	40.5 ± 0.7	52.7 ± 0.4	20.7 ± 4.6
		Qwen2.5-1.5B		
BPE	4.39	46.4 ± 0.5	58.4 ± 0.9	19.5 ± 3.2
Turnip (CR4)	4.11	51.6 ± 0.8	62.3 ± 0.7	23.6 ± 1.3
Turnip (CR8)	8.00	47.6 ± 0.3	58.6 ± 0.7	23.2 ± 2.8
Turnip (CR12)	12.27	42.8 ± 1.9	54.9 ± 0.8	21.7 ± 2.7

depends on the language model. For Turnip, we train the encoder and decoder for the same three ratios as BLT. Among them, CR4 compresses about as much as BPE and thus serves as a control. We then fine-tune each language model with each tokenizer for 10K steps. Besides the tokenizer-specific parameters (the embedding table for BPE and the encoder, decoder, and adapters for Turnip), only the attention query and output projections of the language model are updated, so the setups differ only in the tokenizer. Checkpoints are selected by held-out bits per byte (Delétang et al., 2024). For generation, we use temperature 0.6, top-p 0.9, and an output limit of 8K bytes.

3.2 Main results

Table 1 shows that BPE covers about 4 bytes per token. Since BPE is the standard tokenizer, its accuracy serves as the reference that a more compressive tokenizer should at least preserve.

The pretrained BLT-1B scores far below the Qwen2.5 models at every threshold. This is expected, as mainstream language models are trained with more advanced methods and have better all-round capabilities. The integrated design of BLT, however, requires a language model to be trained with it.

By contrast, Turnip runs on the same pretrained language models as BPE and compresses the context well beyond it while preserving generation quality. At a similar compression ratio, Turnip is at least as accurate as BPE at both model sizes. At about 8 bytes per token, it nearly doubles the compression ratio at the same accuracy (e.g., 42.8 vs. 42.9 strict prompt accuracy at 0.5B and 47.6 vs. 46.4 at 1.5B). These results are consistent with our analysis in Section 2.2, which predicts that a lossy, context-dependent tokenizer can compress more without increasing the next-byte loss.

Moreover, the accuracy of Turnip scales with the language model. On IFEval, it rises from 0.5B to 1.5B at every compression ratio by margins comparable to those of BPE. Since both language models start from the same encoder and decoder, this gain comes from the larger language model. The benefit of a larger language model thus carries over to Turnip as it does to BPE.

3.3 In-depth analyses

Accuracy against compression ratios. By Lemma 1, longer segments load more information onto each token, which increases ℓ^{tok} once the tokens cannot retain it. To examine the resulting cost in generation quality, we plot the accuracy of Turnip against its compression ratio in Figure 2. Accuracy decreases gradually as the number of bytes per token grows, so the budget β offers direct control over this trade-off.

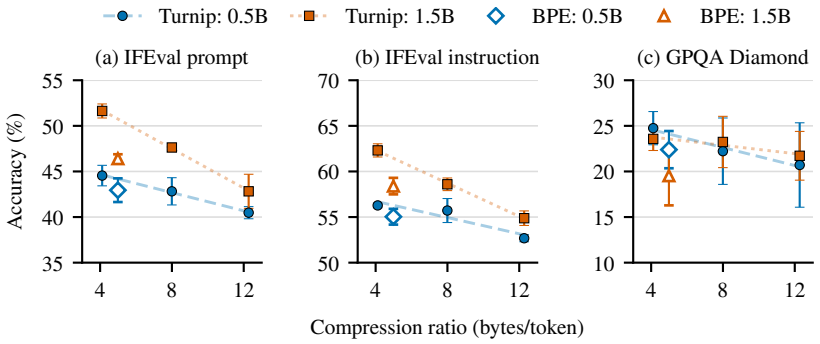

Figure 2: Accuracy of BPE and Turnip against compression ratio. Hollow markers show BPE. Error bars show standard deviations; transparent lines are linear fits to the Turnip results of each model size.

Table 2: Accuracy (%) of Turnip on Qwen2.5-1.5B with fixed-length and entropy-based segments.

	IFEval prompt		IFEval instruction		GPQA Diamond	
Ratio	Fixed	Entropy	Fixed	Entropy	Fixed	Entropy
CR4	51.2 ± 1.3	51.6 ± 0.8	62.1 ± 1.9	62.3 ± 0.7	21.0 ± 0.8	23.6 ± 1.3
CR8	46.7 ± 1.8	47.6 ± 0.3	57.3 ± 2.2	58.6 ± 0.7	20.0 ± 0.8	23.2 ± 2.8
CR12	40.3 ± 1.0	42.8 ± 1.9	52.2 ± 0.4	54.9 ± 0.8	16.8 ± 0.8	21.7 ± 2.7

Entropy-based segmentation. To test whether segments should carry similar amounts of information (Section 2.3), we replace the entropy model with fixed-length segments at each of the three ratios. Entropy-based segmentation is more accurate at every compression ratio on both benchmarks (Table 2). This agrees with Lemma 1, as fixed-length segments overload their tokens where the text is dense and thus increase ℓ^{tok}.

Inference speed. We next verify that compression translates into faster inference even after accounting for the cost of the tokenizer itself. To this end, we measure the time to process a prompt and produce the first output (prefill time), where BPE runs in vLLM (Kwon et al., 2023). Since speed does not depend on the trained weights, we profile synthetic prompts at fixed compression ratios. The speedup of Turnip grows with both the prompt length and the compression ratio (Figure 3 a, b). That of the CR4 control approaches one, since $\gamma \approx 1$ and the linear cost κn of the tokenizer vanishes relative to attention. These measured speedups on long prompts also match the operation counts of Section 2.4 (Appendix B).

Scaling to larger language models. Section 2.4 further predicts larger speedups for larger language models, so we repeat the profile with untrained Qwen2.5-3B and 7B. With these models, the prefill speedup on long prompts exceeds the factor γ by which Turnip shortens the context (Figure 3 c, d). Even the control matches BPE, and Turnip pulls ahead from shorter prompts onward. Decoding is also faster for models of 1.5B and larger (Appendix B). These findings show that the benefit of Turnip grows with the size of the language model.

4 Related Work

Tokenization. Language models have long relied on subword tokenizers that map text losslessly onto a fixed vocabulary, such as BPE (Sennrich et al., 2016) and unigram tokenization (Kudo, 2018) implemented in SentencePiece (Kudo & Richardson, 2018). Much work has sought to understand and improve these tokenizers within this design. Their effectiveness has been linked to shorter sequences (Gallé, 2019), and the relation between compression and downstream performance has been examined (Goldman et al., 2024; Schmidt et al., 2024). Other work has analyzed the merge procedure of BPE formally (Zouhar et al., 2023) and refined how its vocabulary is built (Chizhov et al., 2024). Dynamic tokenization further shortens the sequences of pretrained models by merging

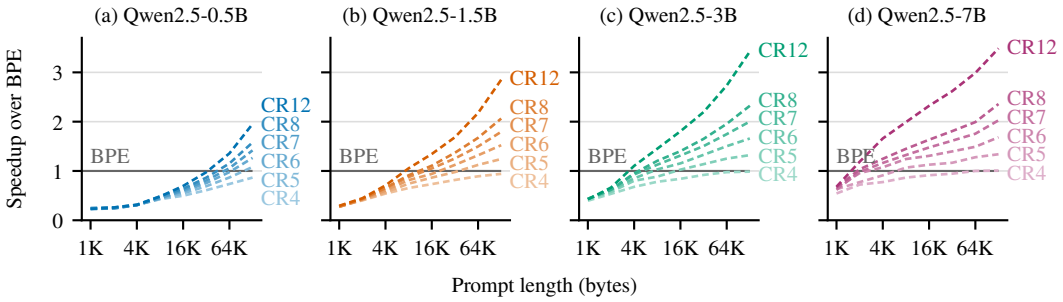

Figure 3: Prefill speedup of Turnip over BPE on the same language model against prompt length.

their subword tokens on the fly (Feher et al., 2025). Throughout this line of work, tokens remain lossless and context-independent. Our analysis shows that these two properties themselves cap compression, and Turnip lifts the cap by removing both.

Byte-level language modeling. Byte-level language models remove the fixed vocabulary and read raw bytes (Xue et al., 2022). Since a byte carries far less information than a subword token, they must group bytes to keep sequences short. Early models learn soft segmentations inside the model (Tay et al., 2022; Godey et al., 2022). Later ones pass patches of bytes to a larger model and form them by fixed sizes (Yu et al., 2023) or space-like bytes (Slagle, 2024). BLT and H-Net make the boundaries adaptive. BLT starts a new patch wherever the next-byte entropy exceeds a threshold (Pagnoni et al., 2025), and H-Net learns a router that places a boundary where adjacent representations differ (Hwang et al., 2026). These boundary criteria are designed heuristically and decide each boundary from a local signal, so a patch closes regardless of how much information it already holds. Turnip instead accumulates the entropy within each segment up to a budget so that each token carries a similar amount of information, which keeps ℓ^{tok} in Lemma 1 low. Moreover, the patching of these models is inseparable from their language model. By contrast, Turnip is trained without a language model and compresses the context for existing pretrained models beyond the lossless limit.

Efficient attention. Long contexts have also motivated methods that cut the quadratic cost of self-attention. Some attend only to a local window plus a few global tokens (Beltagy et al., 2020; Xiao et al., 2024) or to the keys and values judged most important (Zhang et al., 2023; Li et al., 2024). Others select the relevant context on the fly (Mohtashami & Jaggi, 2023; Hao et al., 2025) or retrieve it from an external memory (Wu et al., 2022; Bertsch et al., 2023). Kernel methods make attention linear in the context length (Katharopoulos et al., 2020; Peng et al., 2021), and IO-aware kernels speed up exact attention (Dao et al., 2022). All of them reduce only the attention cost and leave the number of positions unchanged. Turnip instead shortens the sequence by the factor γ, which also reduces the cost of the linear components of the language model (Section 2.4). Since the two approaches act on different parts of the cost, they are orthogonal and can be combined. With any attention whose cost grows superlinearly with the context length, Turnip then yields a speedup of more than γ.

5 Conclusion

In this paper, we present Turnip, a neural tokenizer that compresses the context for Transformer models. Its design is guided by our theoretical analysis of how far a tokenizer can compress the context. Our experiments confirm that Turnip preserves generation quality at a much higher compression ratio than BPE, and its benefit grows with the size of the language model and the context length. We hope that Turnip provides a new perspective on tokenization for efficient language modeling. We leave large-scale verification to future work if computational budgets permit.

AI USE STATEMENT

We used AI agents for three aspects of this work. (1) For the scientific idea and theoretical foundation, we proposed both without AI agents and used AI agents only to check the rigor of our formulation. (2) For the implementation, we used AI agents to migrate our codebase from JAX to PyTorch and to write CUDA kernels that improve GPU utilization. We verified the migrated code and the CUDA kernels by testing their outputs against our original JAX implementation. We designed the experiments and interpreted the results without AI agents. (3) For the paper writing, we decided the content, structure, and claims, and used AI agents to draft and revise the text under our direction down to individual words, write the scripts that render the figures and tables directly from our experiment logs, and format the bibliography. We read every AI edit and revised it ourselves until the text expressed our intent exactly. We take responsibility for the final content of this work, including all code, text, figures, and tables produced with the aid of AI agents.

ETHICS STATEMENT

Our research focuses on compressing the context of Transformer language models with a neural tokenizer to reduce the time and energy consumption of inference. All models and datasets we use are publicly available. As far as we are aware, it does not pose any new ethical or societal risks that require specific attention.

REPRODUCIBILITY STATEMENT

All models and datasets we use are publicly accessible. For the models, we download them from the HuggingFace hub using the following identifiers:

- `Qwen/Qwen2.5-0.5B`, `Qwen/Qwen2.5-1.5B`, `Qwen/Qwen2.5-3B`, and `Qwen/Qwen2.5-7B`
- `itazap/blt-1b-hf`

For the datasets, we similarly obtain them from HuggingFace:

- `allenai/Dolci-Think-SFT-32B`
- `google/IFEval`
- `Idavidrein/gpqa` (the `gpqa_diamond` subset)

Section 2.3 explains the procedure of Turnip, and Section 3.1 gives the recipe for training the tokenizers and fine-tuning the language models. We measure all speedups on synthetic prompts with exactly four bytes per BPE token, in BF16 on one NVIDIA RTX A6000 with batch size one and equal input and output byte counts. Prefill processes the input bytes and selects the first output. Decoding excludes prefill and times 96 output bytes after 96 warmup bytes. Each value uses three timed repetitions, and we discard one warmup repetition at every context size. BPE runs in vLLM, whereas Turnip uses its resident generation cache with CUDA graphs for its customized pipeline.

REFERENCES

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL `https://arxiv.org/abs/2004.05150`.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. Unlimiformer: Long-range transformers with unlimited length input. In *NeurIPS*, volume 36, pp. 35522–35543, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/6f9806a5adc72b5b834b27e4c7c0df9b-Abstract-Conference.html`.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa

Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024. URL https://arxiv.org/abs/2405.14782.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, volume 33, pp. 1877–1901, 2020. URL https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. BPE gets picky: Efficient vocabulary refinement during tokenizer training. In *EMNLP*, pp. 16587–16604. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.emnlp-main.925/.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, second edition, 2006.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, volume 35, pp. 16344–16359, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *ICLR*, 2024. URL https://openreview.net/forum?id=jznbgiynus.

Darius Feher, Ivan Vulić, and Benjamin Minixhofer. Retrofitting large language models with dynamic tokenization. In *ACL*, pp. 29866–29883. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1444/.

Matthias Gallé. Investigating the effectiveness of BPE: The power of shorter sequences. In *EMNLP-IJCNLP*, pp. 1375–1381. Association for Computational Linguistics, 2019. URL https://aclanthology.org/D19-1141/.

Nathan Godey, Roman Castagné, Éric de la Clergerie, and Benoît Sagot. MANTa: Efficient gradient-based tokenization for end-to-end robust language modeling. In *Findings of EMNLP*, pp. 2859–2870. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.findings-emnlp.207/.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In *Findings of ACL*, pp. 2274–2286. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.findings-acl.134/.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Yongchang Hao, Mengyao Zhai, Hossein Hajimirsadeghi, Sepidehsadat Hosseini, and Frederick Tung. Radar: Fast long-context decoding for any transformer. In *ICLR*, 2025. URL https://openreview.net/forum?id=ZTpWOwMrzQ.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. In *ICLR*, 2026. URL https://openreview.net/forum?id=ZbfLR9NbNF.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165, 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *NeurIPS*, volume 35, pp. 22199–22213, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *ACL*, pp. 66–75. Association for Computational Linguistics, 2018. URL https://aclanthology.org/P18-1007/.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP: System Demonstrations*, pp. 66–71, 2018. URL https://aclanthology.org/D18-2012/.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, pp. 611–626, 2023. URL https://dl.acm.org/doi/10.1145/3600006.3613165.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *NeurIPS*, volume 37, pp. 22947–22970, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Sarah E. Marzen and James P. Crutchfield. Predictive rate-distortion for infinite-order Markov processes. *J. Stat. Phys.*, 163(6):1312–1338, 2016. URL https://link.springer.com/article/10.1007/s10955-016-1520-1.

Amirkeivan Mohtashami and Martin Jaggi. Random-access infinite context length for transformers. In *NeurIPS*, volume 36, pp. 54567–54585, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/ab05dc8bf36a9f66edbff6992ec86f56-Abstract-Conference.html.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E. Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte Latent Transformer: Patches scale better than tokens. In *ACL*, pp. 9238–9258, 2025. URL https://aclanthology.org/2025.acl-long.453/.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. Random feature attention. In *ICLR*, 2021. URL https://openreview.net/forum?id=QtTKTdVrFBB.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language-models.pdf.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *COLM*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. Tokenization is more than compression. In *EMNLP*, pp. 678–702. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.emnlp-main.40/.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, pp. 1715–1725, 2016. URL https://aclanthology.org/P16-1162/.

Cosma Rohilla Shalizi and James P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *J. Stat. Phys.*, 104(3–4):817–879, 2001. URL https://link.springer.com/article/10.1023/A:1010388907793.

Claude E. Shannon. Prediction and entropy of printed English. *Bell Syst. Tech. J.*, 30(1):50–64, 1951. URL https://ieeexplore.ieee.org/document/6773263/.

Kevin Slagle. SpaceByte: Towards deleting tokenization from large language modeling. In *NeurIPS*, volume 37, pp. 124925–124950, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e1f418450107c4a0ddc16d008d131573-Abstract-Conference.html.

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 *ICLR*, 2022. URL https://openreview.net/forum?id=JtBRnrlOEFN.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025. URL https://arxiv.org/abs/2512.13961.

Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Allerton Conference on Communication, Control and Computing*, pp. 368–377, 1999. URL https://arxiv.org/abs/physics/0004057.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, volume 30, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *ICLR*, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

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. *TACL*, 10:291–306, 2022. URL https://aclanthology.org/2022.tacl-1.17/.

Lili Yu, Dániel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multi-scale transformers. In *NeurIPS*, volume 36, pp. 78808–78823, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/f8f78f8043f35890181a824e53a57134-Abstract-Conference.html.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, volume 36, pp. 34661–34710, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In *Findings of ACL*, pp. 598–614, 2023. URL https://aclanthology.org/2023.findings-acl.38/.

A Technical proofs

A.1 Proof of Lemma 1

Lemma 1. *For any deterministic tokenization function ε whose output length grows at most linearly with its input length and any model q_θ, we have*

$$\rho(\varepsilon) \le \frac{\log_2 |\mathcal{C}|}{h^{\mathrm{rep}}(\varepsilon)} \quad \textit{and} \quad \mathcal{L}(\theta, \varepsilon) = \underbrace{h^{\mathrm{src}}}_{\textit{source}} + \underbrace{\ell^{\mathrm{tok}}(\varepsilon)}_{\textit{tokenization}} + \underbrace{\ell^{\mathrm{lm}}(q_\theta; \varepsilon)}_{\textit{model}}, \tag{4}$$

where

$$h^{\mathrm{rep}}(\varepsilon) := \lim_{n\to\infty} \frac{1}{n} H(\varepsilon(\mathbf{X}_{<n}))\,, \tag{5}$$

$$h^{\mathrm{src}} := \lim_{n\to\infty} \frac{1}{n} H(\mathbf{X}_{<n}), \tag{6}$$

$$\ell^{\mathrm{tok}}(\varepsilon) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} I(X_t; \mathbf{X}_{<t} \mid \varepsilon(\mathbf{X}_{<t}))\,, \tag{7}$$

$$\ell^{\mathrm{lm}}(q; \varepsilon) := \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} \mathbb{E}_p[D_{\mathrm{KL}}(p(\cdot \mid \varepsilon(\mathbf{X}_{<t})) \,\|\, q(\cdot \mid \varepsilon(\mathbf{X}_{<t})))]\,. \tag{8}$$

Proof. For every t, the cross-entropy decomposition (Cover & Thomas, 2006) gives

$$\begin{aligned} \mathbb{E}_p[-\log_2 q_\theta(X_t \mid \varepsilon(\mathbf{X}_{<t}))] = {} & H(X_t \mid \varepsilon(\mathbf{X}_{<t})) \\ & + \mathbb{E}_p[D_{\mathrm{KL}}(p(\cdot \mid \varepsilon(\mathbf{X}_{<t})) \| q_\theta(\cdot \mid \varepsilon(\mathbf{X}_{<t})))]\,. \end{aligned} \tag{19}$$

Since $\varepsilon(\mathbf{X}_{<t})$ is a deterministic function of $\mathbf{X}_{<t}$,

$$H(X_t \mid \mathbf{X}_{<t}, \varepsilon(\mathbf{X}_{<t})) = H(X_t \mid \mathbf{X}_{<t}). \tag{20}$$

The definition of conditional mutual information therefore gives

$$H(X_t \mid \varepsilon(\mathbf{X}_{<t})) = H(X_t \mid \mathbf{X}_{<t}) + I(X_t; \mathbf{X}_{<t} \mid \varepsilon(\mathbf{X}_{<t})). \tag{21}$$

Substituting this identity into Equation (19) and summing over t with $\sum_{t=0}^{n-1} H(X_t \mid \mathbf{X}_{<t}) = H(\mathbf{X}_{<n})$, we obtain the three loss terms after dividing by n and taking the limit.

For the compression bound, the linear-growth condition provides constants a and b such that $|\boldsymbol{\varepsilon}(\mathbf{X}_{<n})| \leq an + b$. Conditioning on the output length gives

$$H(\boldsymbol{\varepsilon}(\mathbf{X}_{<n})) \leq \log_2(an + b + 1) + \mathbb{E}_p[|\boldsymbol{\varepsilon}(\mathbf{X}_{<n})|] \log_2 |\mathcal{C}|. \tag{22}$$

Dividing by n and taking the limit yields $h^{\mathrm{rep}}(\boldsymbol{\varepsilon}) \leq (\log_2 |\mathcal{C}|)/\rho(\boldsymbol{\varepsilon})$, which is equivalent to the stated bound. □

A.2 Proof of Proposition 2

Proposition 2. *If $\boldsymbol{\varepsilon}$ attains the compression bound in Lemma 1, $\rho(\boldsymbol{\varepsilon}) = (\log_2 |\mathcal{C}|)/h^{\mathrm{rep}}(\boldsymbol{\varepsilon})$, and its token count concentrates, $|\boldsymbol{\varepsilon}(\mathbf{X}_{<n})|/n \to 1/\rho(\boldsymbol{\varepsilon})$ in probability, then the uniform predictor $r^*(\cdot \mid \mathbf{C}_{<m}) = \mathrm{Uniform}(\mathcal{C})$ minimizes the next-token loss $\mathcal{L}^{\mathrm{next}}(\cdot; \boldsymbol{\varepsilon})$. In addition, there exists a detokenization function $\boldsymbol{\delta}^*$ such that every context-independent $\boldsymbol{\delta}^0$ satisfies*

$$\ell^{\mathrm{lm}}(q_{r^*,\boldsymbol{\delta}^0}; \boldsymbol{\varepsilon}) \geq H(X_0) - h^{\mathrm{src}} - \ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}) > \ell^{\mathrm{lm}}(q_{r^*,\boldsymbol{\delta}^*}; \boldsymbol{\varepsilon}) = 0 \tag{11}$$

whenever $\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}) < H(X_0) - h^{\mathrm{src}}$.

Proof. Since $\boldsymbol{\varepsilon}$ attains the compression bound, the definitions of $h^{\mathrm{rep}}(\boldsymbol{\varepsilon})$ and $\rho(\boldsymbol{\varepsilon})$ give

$$\lim_{n\to\infty} \frac{H(\boldsymbol{\varepsilon}(\mathbf{X}_{<n}))}{\mathbb{E}_p[|\boldsymbol{\varepsilon}(\mathbf{X}_{<n})|]} = h^{\mathrm{rep}}(\boldsymbol{\varepsilon})\rho(\boldsymbol{\varepsilon}) = \log_2 |\mathcal{C}|. \tag{23}$$

We show that the tokens are asymptotically uniform. Let $L_n = |\boldsymbol{\varepsilon}(\mathbf{X}_{<n})| \leq an + b$ with the constants of the linear-growth condition and let $H_k = H(C_k \mid \mathbf{C}_{<k}) \leq \log_2 |\mathcal{C}|$. Since $\boldsymbol{\varepsilon}(\mathbf{X}_{<n}) = \mathbf{C}_{<L_n}$, the chain rule gives

$$H(\boldsymbol{\varepsilon}(\mathbf{X}_{<n})) \leq H(L_n) + \sum_{k<an+b} g_k, \qquad g_k = \sum_{l>k} \Pr(L_n = l)\, H(C_k \mid \mathbf{C}_{<k}, L_n = l). \tag{24}$$

Each term satisfies $g_k \leq H(C_k \mid \mathbf{C}_{<k}, L_n) \leq H_k$ and $g_k \leq \Pr(L_n > k) \log_2 |\mathcal{C}|$. Since $\sum_k \Pr(L_n > k) = \mathbb{E}_p[L_n]$, $H(L_n) \leq \log_2(an + b + 1)$, and Equation (23) holds, the gaps $\Pr(L_n > k) \log_2 |\mathcal{C}| - g_k \geq 0$ sum to $o(n)$. Fix $\eta \in (0, 1)$ and let $M = \lfloor (1 - \eta)n/\rho(\boldsymbol{\varepsilon}) \rfloor$. Then

$$\sum_{k<M} (\log_2 |\mathcal{C}| - H_k) \leq \sum_{k<M} (\Pr(L_n > k) \log_2 |\mathcal{C}| - g_k) + M \Pr(L_n \leq M) \log_2 |\mathcal{C}| = o(n), \tag{25}$$

since the token count concentrates. Hence $\frac{1}{M} \sum_{k<M} H_k \to \log_2 |\mathcal{C}|$. As every predictor r satisfies $\mathbb{E}_p[-\log_2 r(C_k \mid \mathbf{C}_{<k})] \geq H_k$, its next-token loss is at least $\log_2 |\mathcal{C}|$, which the uniform predictor

$$r^*(\cdot \mid \mathbf{C}_{<m}) = \mathrm{Uniform}(\mathcal{C}) \tag{26}$$

attains. The uniform predictor therefore minimizes the next-token loss.

Let $\boldsymbol{\delta}^0$ be any context-independent detokenization function. Since r^* is uniform,

$$q_{r^*,\boldsymbol{\delta}^0}(x \mid \mathbf{C}_{<m}) = \frac{1}{|\mathcal{C}|} \sum_{c\in\mathcal{C}} \boldsymbol{\delta}^0(x \mid c), \tag{27}$$

which is independent of the token history. Since the best history-independent byte distribution is the marginal $p(\cdot)$, every $\boldsymbol{\delta}^0$ satisfies

$$\begin{aligned} \ell^{\mathrm{lm}}(q_{r^*,\boldsymbol{\delta}^0}; \boldsymbol{\varepsilon}) &\geq \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} I(X_t; \boldsymbol{\varepsilon}(\mathbf{X}_{<t})) \\ &= H(X_0) - h^{\mathrm{src}} - \ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}). \end{aligned} \tag{28}$$

Here the equality follows from the mutual-information chain rule and the definitions of h^{src} and $\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon})$, and the bound is positive whenever $\ell^{\mathrm{tok}}(\boldsymbol{\varepsilon}) < H(X_0) - h^{\mathrm{src}}$. Finally, we choose at each position t

$$\boldsymbol{\delta}^*(\cdot \mid \mathbf{C}_{<m} \parallel (c)) = p(\cdot \mid \mathbf{C}_{<m}). \tag{29}$$

Then $q_{r^*,\boldsymbol{\delta}^*}(\cdot \mid \mathbf{C}_{<m}) = p(\cdot \mid \mathbf{C}_{<m})$ and thus $\ell^{\mathrm{lm}}(q_{r^*,\boldsymbol{\delta}^*}; \boldsymbol{\varepsilon}) = 0$, which proves the claim. □

A.3 Proof of Proposition 3

Proposition 3. *Suppose $|\mathcal{C}| \geq 3$ and the model family contains every predictor $q : \mathcal{C}^* \to \Delta(\mathcal{X})$. Then there exist a source distribution p and a tokenization function ε such that ε attains the best achievable next-byte loss,*

$$\inf_{\theta'} \mathcal{L}(\theta', \varepsilon) = \inf_{\theta', \varepsilon'} \mathcal{L}(\theta', \varepsilon'), \tag{12}$$

and every lossless tokenization function ε_0 whose output length grows at most linearly with its input length satisfies $\rho(\varepsilon_0) < \rho(\varepsilon)$.

Proof. We first bound the compression ratio of lossless tokenization functions. If ε_0 is lossless, then $H(\mathbf{X}_{<n} \mid \varepsilon_0(\mathbf{X}_{<n})) = 0$ and therefore

$$H(\varepsilon_0(\mathbf{X}_{<n})) \geq I(\mathbf{X}_{<n}; \varepsilon_0(\mathbf{X}_{<n})) = H(\mathbf{X}_{<n}). \tag{30}$$

Dividing by n and taking the limit gives $h^{\mathrm{rep}}(\varepsilon_0) \geq h^{\mathrm{src}}$. If the output length of ε_0 grows at most linearly, Lemma 1 then gives

$$\rho(\varepsilon_0) \leq \frac{\log_2 |\mathcal{C}|}{h^{\mathrm{src}}}. \tag{31}$$

We now construct a source on which a lossy tokenization function exceeds this bound. Let $\alpha = 1/|\mathcal{C}|$ and consider a symmetric binary Markov source on two byte values $a \neq b$ with X_0 uniform and $\Pr(X_t \neq X_{t-1} \mid \mathbf{X}_{<t}) = \alpha$ for $t \geq 1$. This source is stationary with $H(\mathbf{X}_{<n}) = 1 + (n-1)h_2(\alpha)$, where h_2 is the binary entropy function. Hence $h^{\mathrm{src}} = h_2(\alpha)$. We pick three distinct tokens $c_a, c_b, c_{\mathrm{s}} \in \mathcal{C}$ and let ε emit c_{x_0} after the first byte and c_{s} after every later byte that differs from its predecessor. That is,

$$\varepsilon(\mathbf{x}_{<t}) = (c_{x_0}, \underbrace{c_{\mathrm{s}}, \ldots, c_{\mathrm{s}}}_{K_t \text{ times}}), \qquad K_t = \big|\{1 \leq u \leq t-1 : x_u \neq x_{u-1}\}\big|, \tag{32}$$

for $t \geq 1$, and $\varepsilon(\mathbf{x}_{<0})$ is empty.

For $t \geq 1$, the tokens determine the previous byte because $x_{t-1} = x_0$ if K_t is even and $x_{t-1} \neq x_0$ otherwise. Moreover, $\varepsilon(\mathbf{X}_{<t})$ is itself a function of $\mathbf{X}_{<t}$. Since the source is Markov, we therefore have

$$H(X_t \mid \mathbf{X}_{<t}) \leq H(X_t \mid \varepsilon(\mathbf{X}_{<t})) \leq H(X_t \mid X_{t-1}) = H(X_t \mid \mathbf{X}_{<t}). \tag{33}$$

Hence $I(X_t; \mathbf{X}_{<t} \mid \varepsilon(\mathbf{X}_{<t})) = H(X_t \mid \varepsilon(\mathbf{X}_{<t})) - H(X_t \mid \mathbf{X}_{<t}) = 0$ for every $t \geq 1$. The same holds trivially for $t = 0$, so $\ell^{\mathrm{tok}}(\varepsilon) = 0$. By Lemma 1, we then have $\mathcal{L}(\theta, \varepsilon) = h^{\mathrm{src}} + \ell^{\mathrm{lm}}(q_\theta; \varepsilon)$. The model family contains the predictor $q_\theta(\cdot \mid \varepsilon(\mathbf{X}_{<t})) = p(\cdot \mid \varepsilon(\mathbf{X}_{<t}))$, which makes the model term zero. Since ℓ^{tok} and ℓ^{lm} are nonnegative, no tokenization function and model attain a loss below h^{src}. It follows that

$$\inf_{\theta'} \mathcal{L}(\theta', \varepsilon) = h^{\mathrm{src}} = \inf_{\theta', \varepsilon'} \mathcal{L}(\theta', \varepsilon'). \tag{34}$$

Since each of the $n-1$ transitions in $\mathbf{X}_{<n}$ is a change with probability α, we have $\mathbb{E}_p[|\varepsilon(\mathbf{X}_{<n})|] = 1 + (n-1)\alpha$ and thus $\rho(\varepsilon) = 1/\alpha$. Finally,

$$h_2(\alpha) = \alpha \log_2 \frac{1}{\alpha} + (1-\alpha) \log_2 \frac{1}{1-\alpha} > \alpha \log_2 \frac{1}{\alpha} = \alpha \log_2 |\mathcal{C}|, \tag{35}$$

so every lossless ε_0 whose output length grows at most linearly satisfies

$$\rho(\varepsilon_0) \leq \frac{\log_2 |\mathcal{C}|}{h_2(\alpha)} < \frac{1}{\alpha} = \rho(\varepsilon). \tag{36}$$

□

B Prefill and decoding speedups

Section 3.3 reports the prefill speedups of Turnip. Here we add its decoding speedups and relate both to the cost analysis of Section 2.4. Prefill processes the entire prompt at once, whereas decoding then generates the output step by step. Table 3 lists both speedups for two prompt lengths.

Table 3: Speedup of Turnip over BPE on the same language model in prefill and decoding for prompts of 32K and 128K bytes.

		Prefill		Decode	
Language model	Ratio	32K	128K	32K	128K
Qwen2.5-0.5B	CR8	0.87	1.58	0.65	0.73
	CR12	0.97	1.95	0.77	0.89
Qwen2.5-1.5B	CR8	1.37	2.06	1.10	1.21
	CR12	1.69	2.85	1.36	1.55
Qwen2.5-3B	CR8	1.65	2.32	1.37	1.46
	CR12	2.19	3.42	1.81	1.99
Qwen2.5-7B	CR8	1.81	2.36	1.81	1.89
	CR12	2.62	3.49	2.50	2.65

In prefill, Turnip shortens the sequence that the language model processes by the factor γ. Its speedup therefore grows with the prompt length, as self-attention increasingly dominates the cost (Equation (18)). In decoding, the language model takes one step per token, that is, once every ρ bytes for Turnip rather than every ρ_{BPE} bytes for BPE, while the tokenizer adds a fixed cost per byte. The decoding speedup thus changes little with the prompt length but grows with the size of the language model, since a larger model makes each of its steps more costly relative to the tokenizer.

To test the cost analysis quantitatively, we count two operations per multiply-accumulate in the matrix multiplications and causal attention of each component. The encoder and entropy model run on every byte, and the language model and decoder on every token. These counts predict the 128K-byte prefill speedups in Table 3 to within 15% at every size.