Corpus-Dependent Subcharacter Encoding via HMM-Guided Code Assignment

Tatsuya Hiraoka
Mohamed bin Zayed University of Artificial Intelligence,
Nara Institute of Science and Technology, RIKEN
Tatsuya.Hiraoka@mbzuai.ac.ae

Abstract

We propose a corpus-dependent alternative to byte encoding that learns fixed-length atomic codes for characters directly from text, which we refer to as **Latom** (Learned Atom-based Encoding). We instantiate this framework by training an HMM on N-repeated character sequences to estimate "atom" posteriors, followed by a Hungarian assignment yielding a globally optimal one-to-one character–code mapping. Across 14 languages, the encodings improve intrinsic metrics, including token counts after subword tokenization and bigram perplexity, with appropriate code lengths. On Amazon Reviews in six languages, Latom improves text classification accuracy and reduces decoding errors in language model generation. Overall, these results demonstrate that character encodings can be learned from corpus statistics while remaining reversible and compatible with standard tokenization pipelines.

1 Introduction

Text in modern NLP systems is represented as a sequence of discrete symbols, and in many pipelines, this base representation is fixed to UTF-8 bytes by convention. While byte-level encoding provides universality across languages, it is inherently corpus-agnostic and does not adapt to the statistics of the target data. On top of this base representation, many NLP systems tokenize text at the subword level (Sennrich et al., 2016; Kudo and Richardson, 2018), where a token may consist of multiple characters. At the other extreme, bit-, byte-, or character-level processing provides a language-agnostic alternative that helps alleviate the unknown-token problem (Xue et al., 2022; Clark et al., 2022; Tay et al., 2022; Moon et al., 2025). Beyond bytes and characters, prior work has explored injecting language-specific subcharacter information, such as radicals and strokes for Chinese and Japanese, Hangul Jamo for Korean, and pronunciations for various languages (Shi et al., 2015; Stratos, 2017; Jia et al., 2021). While these subcharacter features can improve performance, they are handcrafted and independent of the target corpus. This leaves room to explore subcharacter encodings that adapt to corpus statistics.

Original Sequence		
Character	自	然
Hand-crafted and corpus-independent		
Byte†	0xe8 0x87 0xaa	0xe7 0x84 0xb6
Pronunciation		
Chinese:	zì	rán
Japanese:	shi	zen
Radical	自	月犬灬
Stroke	ノ丨フ一一一	ノフ丶丶一ノ𠂉丿丶丶丶
Corpus-dependent (Latom)		
Latom $N = 2$ †	a31 b34	a12 b26
Latom $N = 3$ †	a12 b21 c12	a28 b41 c18

Table 1: Comparison of hand-crafted and learned subcharacter representations for the word "自然" (nature). The symbol † indicates representations that are reversibly decodable to the original characters.

In this paper, we introduce **Latom** (Learned Atom-based Encoding), a framework for corpus-dependent base text representations that assigns structured, reversible codes to characters based on distributional statistics[1].

We study how to learn subcharacter units directly from a given corpus by assigning each character a fixed-length code composed of multiple elements ("atoms"). We investigate whether such character–code assignments can be learned from corpus statistics without relying on external inventories such as dictionaries. As illustrated in Table 1, we compare conventional hand-crafted subcharacter representations with our automatically learned ones on the Chinese and Japanese word "自然" (nature). The middle part lists corpus-independent features such as byte sequences, pronunciations, radicals, and strokes, while the lower part shows examples

[1]Code: https://github.com/tatHi/latom

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 34569–34593

of our corpus-dependent atom codes learned with code lengths $N = \{2, 3\}$. The obtained codes (e.g., a31 b34) are fixed-length and corpus-specific, and are reversibly decodable to the original characters.

This paper instantiates Latom using an HMM to estimate corpus-dependent preferences over atoms, followed by a Hungarian algorithm to obtain a one-to-one character–code assignment.

We evaluate the proposed encoding both intrinsically and extrinsically. Across 14 languages, it improves tokenization efficiency and bigram perplexity when the code length is chosen appropriately relative to the character vocabulary size. On Amazon Reviews in six languages, it achieves competitive classification performance and reduces decoding errors in language model generation.

Conceptually, our approach can be viewed as a corpus-dependent alternative to byte-level encoding, where the base alphabet is learned from data while remaining reversible and compatible with standard subword tokenization pipelines. Our contributions are:

- We introduce Latom, a framework for corpus-dependent atomic encoding of text as an alternative to fixed byte representations.
- As one instantiation of Latom, we propose a method that learns fixed-length codebooks using HMM posteriors and a Hungarian assignment.
- We provide intrinsic and extrinsic evaluations across multiple languages, showing improvements in compression, perplexity, and downstream performance.

2 Related Work

Modern NLP systems typically employ subword tokenization to control vocabulary size and token granularity (Sennrich et al., 2016; Kudo and Richardson, 2018). While subwords are built from characters and often yield semantically meaningful fragments, they abstract away character-level information. Consequently, character-based inputs are sometimes adopted to retain finer-grained information (Zhang et al., 2015; Kuru et al., 2016; El Boukkouri et al., 2020; Clark et al., 2022; Tay et al., 2022).

Byte-level processing has become a practical alternative in both pretraining and downstream tasks (Gillick et al., 2016; Radford et al., 2019; Xue et al., 2022; Huang et al., 2025; Pagnoni et al., 2025). Recent work further explores byte-level architectures and training strategies, including bit-level variants below bytes (Moon et al., 2025). However, these approaches fix the base alphabet (bytes or bits) and do not adapt it to the target corpus. Recent work has explored learning latent byte-level representations using vector quantization (Hsiao et al., 2024), but these methods rely on learned decoders and do not guarantee a reversible or injective mapping. In contrast, our method constructs a deterministic, one-to-one encoding from characters to codes.

Beyond standard NLP modeling, compression-oriented schemes such as SCSU and BOCU-1 provide reversible Unicode byte encodings (Scherer and Davis, 2006). More recent work proposes alternative encodings to improve compression for specific languages (Xue, 2023), to remap byte-level representations for computational efficiency (Limisiewicz et al., 2024), or to structure pretokenization with script-aware codes (Land and Arnett, 2025). These approaches focus on designing efficient encodings, rather than learning corpus-dependent subcharacter representations.

For languages with compositional characters, many studies inject subcharacter information to improve downstream performance, including radicals and strokes for Chinese and Japanese (Sun et al., 2014; Shi et al., 2015; Meng et al., 2019; Wu et al., 2025), Hangul Jamo for Korean (Stratos, 2017; Cho et al., 2019; Lee et al., 2025), and pronunciation features (Jia et al., 2021; Zouhar et al., 2024; Jung et al., 2024). Glyph-based methods often apply CNNs over rendered characters (Meng et al., 2019). These features are typically corpus-independent and rely on external linguistic resources.

Closer to our goal, Morfessor learns corpus-dependent word segmentation without supervision (Virpioja et al., 2013), and Brown clustering assigns discrete class codes to words based on distributional context (Brown et al., 1992). Other approaches aim to improve tokenization depending on the dataset (He et al., 2020; Hiraoka et al., 2020, 2021). However, these methods operate above the character level.

In contrast, Latom learns a fixed-length, reversible mapping at the subcharacter level, assigning codes directly to characters based on corpus statistics. Our implementation aligns forward and backward contexts using an HMM and solves a global one-to-one assignment, drawing inspiration from speech recognition (Bahl et al., 1983).

3 Proposed Method

3.1 Core Idea of Latom: Learned Atom-based Encoding

Our goal is to learn a one-to-one mapping between each character and a length-N code formed by combining K atom types. Unlike hand-crafted subcharacter units (e.g., bytes or radicals), our codes are learned from the target corpus, and their length and combinatorial capacity are controlled by N and K. We assign a fixed-length code to every character, which simplifies decoding and allows characters appearing in similar contexts to share parts of their codes.

Let $c \in C$ be a character in the vocabulary and $x = (c_1, \ldots, c_L)$ a sequence of L characters. We seek an injective mapping (codebook) f from characters to length-N atom codes:

$$f : C \to A^N, \qquad f(c) = \alpha = (a_1, \ldots, a_N).$$

Injectivity requires $c \neq c' \Rightarrow f(c) \neq f(c')$, and feasibility requires $|C| \leq K^N$ when each digit has K candidate atom types. Applying f to x yields a sequence of $L \times N$ atoms.

The key idea of Latom is to learn the atom–character codebook from data, and this codebook can be constructed in multiple ways. In this paper, as one instantiation, we construct f from the corpus in two steps:

1. Train an HMM on N-repeated sequences and compute posteriors over hidden states (atoms) at every time step (§ 3.2).

2. Aggregate these posteriors into scores for character-code pairs and solve a one-to-one assignment via the Hungarian algorithm (§ 3.3).

Figure 1 provides an overview of this process.

3.2 Step 1: Estimating Atom Posteriors with an HMM

We realize atoms as the hidden states of an HMM with hidden-state set A and emission characters C.

N-repeated observation:

Given a character sequence $x = (c_1, \ldots, c_L)$ and code length N, we first form an N-repeated observation $y = \mathcal{Y}(x)$ by repeating each character N times, similarly to traditional speech recognition modeling (Rabiner, 2002; Ostendorf et al., 2002). The observation length is $T = NL$. For example,

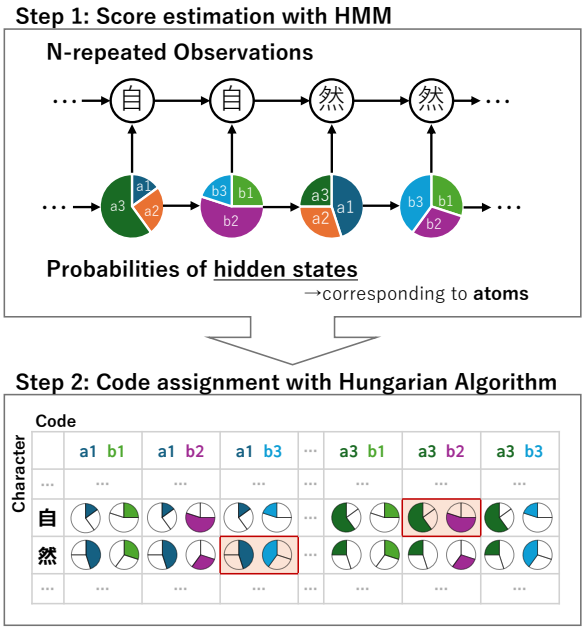

Figure 1: Overview of the proposed method, which combines HMM-based posterior estimation with Hungarian assignment. Step 1 computes posterior distributions over hidden states (atoms) from N-repeated observations. Step 2 assigns each character a fixed-length atom code via a global one-to-one assignment. The example shows code assignment with $N = 2$ and $K = 3$ under a periodic partition of states by digit. Red boxes indicate selected assignments.

with $N = 3$ and the text "自然言語 (natural language)," the repeated observation is "自自自然然然言言言語語語" with $T = 3 \times 4 = 12$. Repeating each character N times allows the model to associate different latent states with different contextual roles of the same character, rather than forcing a single latent representation.

Training and posteriors:

We fit the HMM to the repeated sequences y by maximum likelihood using the Baum-Welch algorithm (Baum et al., 1970). With trained parameters $\hat{\theta}$, the model provides posteriors over hidden states z at each time step t:

$$\gamma_{y,t}(a) = P(z_t = a \mid y; \hat{\theta}).$$

Let the n-th copy of the l-th character in x correspond to time index $t = N(l-1)+n$ in $y = \mathcal{Y}(x)$. For convenience, we define

$$\gamma_{x,n,l}(a) := \gamma_{y=\mathcal{Y}(x),t=N(l-1)+n}(a).$$

For example, with the text "自然" as x and $N = 3$, $\gamma_{x,n=1,l=2}(a)$ indicates the posterior when the first

hidden state of the second character (i.e., the first copy of the second character "然" in the repeated sequence "自自自然然然") is a.

We then average posteriors over all occurrences of a character in the corpus X to obtain the plausibility of atom a at digit n for character c:

$$q_{c,n}(a) = \frac{1}{M_c} \sum_{x \in X} \sum_{l=1}^{L} \gamma_{x,n,l}(a) \mathbb{1}\{c_l = c\},$$

$$M_c = \sum_{x \in X} \sum_{l=1}^{L} \mathbb{1}\{c_l = c\}.$$

Here $\mathbb{1}\{\cdot\}$ is the indicator function and M_c is the total number of occurrences of c in the corpus. Note that the length of each sample in the corpus is $L = |x|$ and c_l is the l-th character in x. This normalization averages over all occurrences of c and treats each digit position independently, summarizing how strongly each atom is preferred at a given code position.

Periodic state partition:

To ensure reversibility, we restrict which atoms can appear at each digit position n by partitioning the state set:

$$A = A_1 \dot{\cup} A_2 \dot{\cup} \cdots \dot{\cup} A_N, \qquad |A_n| = K,$$

where $\dot{\cup}$ indicates the disjoint union of atom sets. For instance, with $N = 2$ and $K = 3$, the model may use $A_1 = \{\texttt{a0}, \texttt{a1}, \texttt{a2}\}$ for the first digit and $A_2 = \{\texttt{b0}, \texttt{b1}, \texttt{b2}\}$ for the second (see the top of Figure 1)[2]. During training, we mask transitions so that $A_n \to A_{n+1}$ only (with $A_{N+1} \equiv A_1$), restrict the initial distribution to A_1 and the end distribution to A_N, thereby tying each atom position to a fixed phase that enables simple decoding and consistency checks (see §3.4).

3.3 Step 2: Code Assignment with the Hungarian Algorithm

Given $q_{c,n}$, we score a character-code pair (c, α), where $\alpha = (a_1, \ldots, a_N) \in A_1 \times \cdots \times A_N$, by

$$s(c, \alpha) = \sum_{n=1}^{N} \log(q_{c,n}(a_n)).$$

A greedy assignment based on local scores can easily lead to collisions between characters, whereas a global assignment explicitly enforces injectivity. We solve a linear assignment problem to obtain a one-to-one mapping $f : C \to A_1 \times \cdots \times A_N$, as shown at the bottom of Figure 1:

$$f^\star = \arg \max_{f \text{ injective}} \sum_{c \in C} s\big(c, f(c)\big). \qquad (1)$$

We apply the Hungarian algorithm to the cost matrix defined by $-s(c, \alpha)$. Feasibility requires $K^N \geq |C|$; any unused codes remain unassigned.

3.4 Encoding and Decoding

Given the learned codebook f, a character sequence is encoded by concatenating the codes $f(c_l)$. For example, with the codebook in Figure 1, the sequence "自然" is encoded as `a3 b2 a1 b3`. Note that f cannot encode characters that are not included in the codebook (i.e., characters not observed in the corpus used for codebook construction).

Because f is one-to-one and every character maps to a fixed-length code, decoding proceeds by reading the atom sequence in blocks of N from the beginning. The structure $A = A_1 \dot{\cup} \cdots \dot{\cup} A_N$ also allows simple error checks: (i) the sequence length must be a multiple of N; and (ii) the atom at absolute position t must belong to $A_{((t-1) \bmod N)+1}$. Violation of either condition flags a decoding error.

4 Intrinsic Evaluation

This section evaluates Latom from an intrinsic perspective, focusing on statistical properties independent of downstream models. Specifically, we examine the effects of corpus-dependent atomic encodings on compression efficiency under constrained vocabularies and on local statistics as reflected by bigram perplexity.

4.1 Experimental Setup

4.1.1 Dataset

We report experimental results on three languages (English, German, and Japanese). To control for cross-lingual content differences, we use a multilingual Bible corpus which provides parallel or near-parallel content[3]. Results on additional languages are reported in Appendix A.4.

[2] In this paper, we denote atoms with two-character symbols formed by concatenating a Latin letter with a digit, e.g., `a0`. The letter prefix indicates the digit position (e.g., `a*` for the first slot, `b*` for the second). One can use any notation if it ensures clear distinguishability of the digits.

[3] https://github.com/christos-c/bible-corpus

	$\lvert C\rvert$	K^2	K^3	K^4	K^5
English	60	8^2	4^3	3^4	3^5
German	78	9^2	5^3	3^4	3^5
Japanese	983	32^2	10^3	6^4	4^5

Table 2: Minimum values of K for each code length N, shown as K^N, such that $K^N \geq |C|$, where $|C|$ is the character vocabulary size for each language.

		2nd digit							
		b0	b1	b2	b3	b4	b5	b6	b7
1st digit	a0	U	N	T	F	A	B	Y	W
	a1	D	C	E	H	O	Z	G	
	a2	M	I	L	P	S	d	k	!
	a3	R	e	v	i	(	-	u	;
	a4	K	g	,	z			p	:
	a5	y	J	b	x		w	c	.
	a6	l	n	t	f	h	a	q	?
	a7	r	’	j	_	o	m	s	)

Table 3: Example learned codebook for English with $N = 2$. Rows correspond to first-digit atoms (a0, ...) and columns to second-digit atoms (b0, ...). For example, the character “T” maps to “a0 b2”. Empty cells indicate unused codes, and “_” denotes the whitespace character. Rows and columns are permuted for visualization so that similar groups appear adjacent.

4.1.2 Hyperparameters

We consider code lengths $N \in \{1, 2, 3, 4, 5\}$, where $N = 1$ corresponds to the original character sequence. For each N, we choose the smallest number of atom types per digit, K, such that $K^N \geq |C|$, where $|C|$ is the size of the character vocabulary in the corpus for that language[4]. For example, if $|C| = 60$ for English, we select $K = 8$ for $N = 2$ because $8^2 = 64 \geq 60$. Table 2 summarizes the (N, K) choices.

4.1.3 Baselines

We compare our method, Latom, with three reversibly decodable baselines[5].

Character: The original character sequence. This baseline evaluates whether atom-level encoding improves tokenization statistics relative to character-level processing.

Byte: A byte-level representation obtained by encoding each character in UTF-8 and treating each byte as an atomic symbol. For English, most characters are ASCII, so the byte and character sequences largely coincide in practice.

Random codebook: For each (N, K), we sample an injective mapping $f_{\text{rand}} : C \to A_1 \times \cdots \times A_N$ uniformly without replacement under the same periodic partition, matching our method's (N, K). This controls for the effect of code length and alphabet size without using corpus statistics.

4.2 Qualitative Analysis

Before the quantitative analysis, we present a learned codebook and a tokenization trace to illustrate the behavior of the method.

4.2.1 Obtained Codebook

Table 3 shows a codebook obtained by our method for English with $N = 2$. The assignment reflects contexts estimated from HMM transitions. The first digit often groups characters that share similar forward contexts, while the second digit groups characters with similar backward contexts.

For example, some first-digit symbols are shared by capital letters that frequently occur at sentence starts (a0 and a1), while the second-digit b7 groups punctuation that typically occur at sentence boundaries. Appendix A.1 shows similar tendencies for German and Japanese.

These observations indicate that the learned codes consistently reflect distributional character contexts, rather than visual or linguistic decomposition. Intuitively, this increases the frequency of certain atom bigrams in the encoded sequence, which later supports more efficient tokenization.

4.2.2 Tokenization Examples

Table 4 shows examples of English text encoded with the $N = 2$ codebook shown in Table 3 and then tokenized. To highlight the difference between character-level and atom-level encoding, we train a BPE tokenizer whose vocabulary size $|V|$ equals the number of character types $|C|$.

As shown in the table, for $|V| = 60$, some atoms remain unmerged (i.e., not concatenated with others) and thus cannot be decoded into complete characters. Frequent sequences (e.g., “_the”) tend to be merged first during BPE training, leaving infrequent atom sequences unmerged; thereby allocating vocabulary capacity to more frequent patterns. For instance, some capital letters such as “I: (a2, b1)” and “G: (a1, b6)” are not merged.

We also observe tokens whose boundaries do not align with character boundaries, for example,

[4]We selected the smallest K to cover the character vocabulary as a simple and practical choice. Appendix A.3 explores alternative choices of K with the hyperparameter tuning.

[5]Appendix A additionally reports the experimental results with other encoding methods, such as radical, stroke, and Myte, in Japanese and Chinese datasets.

$\|V\|$	$\|\tau\|$	Code
16	108	a2 b1 a6 b1 a7 b3 a6 b2 a6 b4 a3 b1 a7 b3 a5 b2 a3 b1 a4 b1 a3 b3 a6 b1 a6 b1 a3 b3 a6 b1 a4 b1 a7 b3 a1 b6 a7 b4 a2 b5 a7 b3 a5 b6 a7 b0 a3 b1 a6 b5 a6 b2 a3 b1 a2 b5 a7 b3 a6 b2 a6 b4 a3 b1 a7 b3 a6 b4 a3 b1 a6 b5 a3 b2 a3 b1 a6 b1 a7 b3 a6 b5 a6 b1 a2 b5 a7 b3 a6 b2 a6 b4 a3 b1 a7 b3 a3 b1 a6 b5 a7 b0 a6 b2 a6 b4 a5 b7
20	79	a2 b1 n _a6 b2 a6 b4 e _ a5 b2 e a4 b1 a3 b3 n n a3 b3 n a4 b1 _ a1 b6 a7 b4 a2 b5 _ a5 b6 a7 b0 e a6 b5 a6 b2 e a2 b5 _a6 b2 a6 b4 e _a6 b4 e a6 b5 a3 b2 e n _a6 b5 n a2 b5 _a6 b2 a6 b4 e _ e a6 b5 a7 b0 a6 b2 a6 b4 a5 b7
30	57	a2 b1 n _t h e _a5 b2 e a4 b1 i n n i n a4 b1 _ a1 b6 o d _a5 b6 r e a t e d _t h e _a6 b4 e a a3 b2 e n _a6 b5 n d _t h e _ e a r t h a5 b7
40	48	a2 b1 n _the _a5 b2 e a4 b1 i n n i n a4 b1 _ a1 b6 o d _a5 b6 r e a t e d _the _h e a a3 b2 e n _a nd _the _ e a r t h a5 b7
50	43	a2 b1 n _the _a5 b2 e g in n in g _ a1 b6 o d _a5 b6 r e a t e d _the _h e a a3 b2 e n _and _the _ e a r t h a5 b7
60	38	a2 b1 n _the _a5 b2e g in n in g _ a1 b6 o d _a5 b6 re a t e d _the _h e a a3 b2e n _and _the _ e a r th .
60	54	I n _ t h e _ b e g i n n i n g _ G o d _ c r e a t e d _ t h e _ h e a v e n _ a n d _ t h e _ e a r t h .

Table 4: Tokenization example for the opening verse of the English Bible encoded with the $N = 2$ codebook shown in Table 3. A BPE tokenizer is trained on the atom sequence with vocabulary size $|V| = |C| = 60$. Each row shows the resulting tokenization at that vocabulary size. $|\tau|$ denotes the number of tokens in the shown tokenization. Segments that can be decoded into complete characters are highlighted in red, and "_" marks whitespace. The bottom row shows character-level tokenization for comparison.

"b2e". This arises because characters that are often followed by "e" (such as b or v) share the same second-digit code (e.g., "b2", see Table 3), so mergers can straddle character boundaries. Overall, these properties reduce the number of tokens relative to character-level tokenization under the same vocabulary size.

4.3 Quantitative Analysis

We quantitatively evaluate whether Latom positively affects the compression efficiency and bigram perplexity. For each language and each N, we obtain a codebook on the Bible corpus, encode the corpus with that codebook, and train a separate BPE tokenizer[6] with a maximum vocabulary size of 3,000, increasing the budget in steps of 50 and allowing whitespace-crossing tokens (Kudo and Richardson, 2018; Kumar and Thawani, 2022; Liu et al., 2026). We then evaluate (i) token count after tokenization and (ii) bigram perplexity at each vocabulary budget. To characterize intrinsic behavior, the same corpus is used for codebook learning, tokenizer training, and evaluation[7].

4.3.1 Compression Performance

Figure 2 shows token-count differences after tokenizing the encoded text in the three languages. As the vocabulary size grows, differences approach zero, indicating that the effect of the codebook diminishes at large budgets. In most settings, the proposed method outperforms random codebooks. Atom-level encoding also often outperforms byte-level encoding, suggesting that corpus-dependent atom codes are a competitive alternative to bytes for compression.

At small $|V|$, both learned and random codebooks yield better compression than the character-level baseline. For languages with smaller alphabets, larger N can temporarily worsen compression, while $N{=}2$ yields more stable gains; Japanese shows consistent improvements across N. Overall, selecting N relative to $|C|$ (Table 2) is important: larger N can hurt compression for small $|C|$, whereas appropriate choices of $(N, |V|)$ for the character vocabulary size yield consistent gains.

4.3.2 Bigram Perplexity

Figure 3 plots the bigram-perplexity difference. Since the curves are computed on the training corpus, they reflect how the learned codebooks capture the distribution of frequent atom bigrams.

As the vocabulary size grows, the difference sometimes dips below zero, especially for smaller N in English and German. Compared to character-level representations, atom-level encoding starts from finer-grained units, which allows frequently co-occurring longer sequences to be merged more flexibly during subword learning. This leads to more stable high-frequency bigrams, contributing to lower bigram perplexity under constrained vocabulary budgets. With larger N, a wider range of budgets yields positive differences, indicating lower perplexity than the character-level baseline. These results suggest that corpus-dependent atomic encodings capture local co-occurrence patterns more effectively than fixed character or byte representations, particularly when vocabulary size is limited. In a few settings, random codebooks exceed our method, indicating headroom for improved code-assignment strategies.

[6] Appendix A.2 reports results with unigram tokenizers.

[7] Appendix A.7 reports the compression performance and bigram perplexity on unseen datasets.

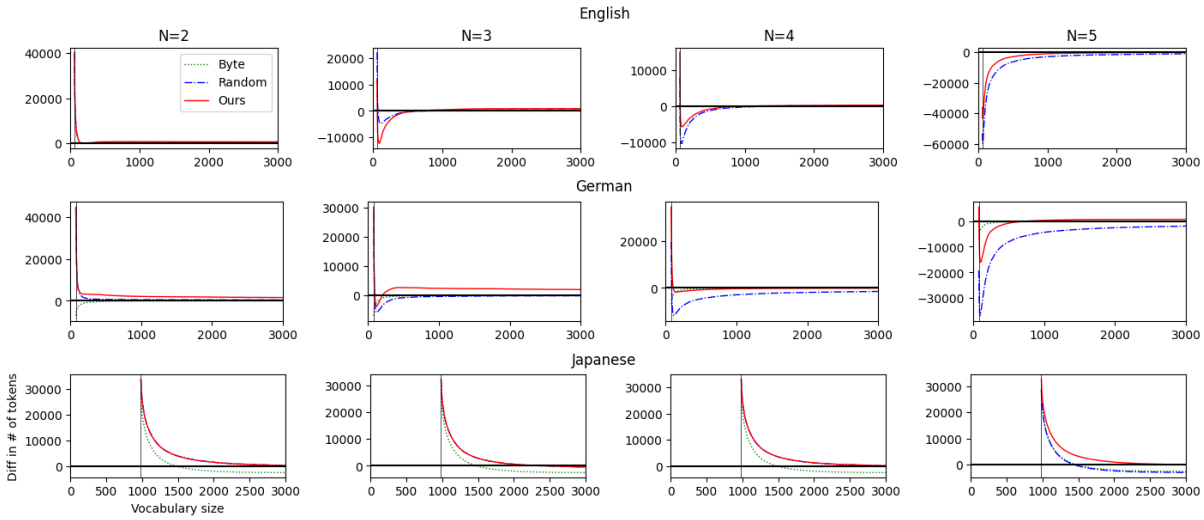

Figure 2: Token-count difference (character-level minus atom-level) across vocabulary sizes. **Higher is better**; values above zero (black horizontal line) indicate fewer tokens than the character-level baseline, showing the better compression performance. The black vertical line marks the character vocabulary size.

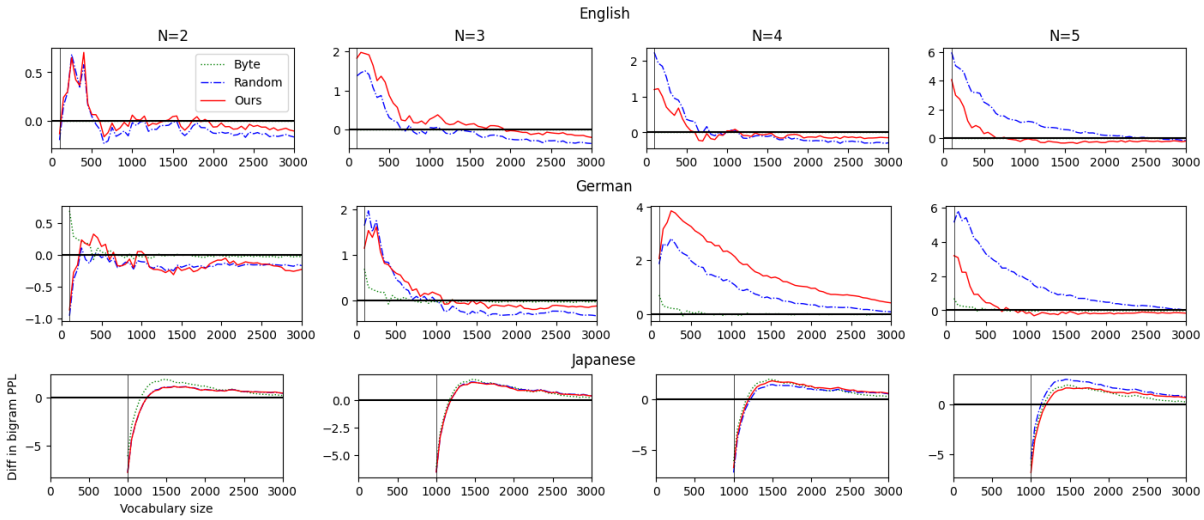

Figure 3: Bigram-perplexity difference (character-level minus atom-level) across vocabulary sizes. **Higher is better**; values above zero indicate lower perplexity than the character-level baseline. The black vertical line marks the character vocabulary size.

5 Extrinsic Evaluation

This section investigates the practical impact of adapting the base encoding when combined with standard neural models and tokenization pipelines. In particular, we assess the effects of corpus-dependent atomic encodings on downstream task performance (§5.1) and on generation robustness, as indicated by decoding errors (§5.2).

5.1 Text Classification

5.1.1 Experimental Setup

We use five-class sentiment classification on the Amazon Reviews corpus (McAuley and Leskovec, 2013; Muennighoff et al., 2023). We chose this task because it offers comparably sized datasets in the same domain across six languages: English, Spanish, German, French, Japanese, and Chinese. For each language, the training set contains 200,000 reviews, and the validation and test sets contain 5,000 reviews each, enabling cross-language comparisons under similar conditions. We report weighted-F1 (%) on the test set.

We train two standard classification architectures from scratch: a bidirectional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997) and a Transformer (Vaswani et al., 2017). Training hy-

↑		BiLSTM						Transformer					
		De	En	Es	Fr	Ja	Zh	De	En	Es	Fr	Ja	Zh
$\|V\| = \|C\|$	Character	61.41	58.99	56.06	57.35	57.23	55.24	48.89	52.22	50.14	50.01	52.27	52.31
	Byte	59.52	**60.71**	**56.14**	56.88	56.14	54.35	**57.49**	**55.54**	**53.30**	**53.30**	**54.33**	**53.41**
	$N = 2$												
	Random	**62.04**	**61.36**	**57.95**	**57.75**	56.59	54.69	**57.28**	**56.59**	**53.92**	**54.00**	**55.08**	**54.12**
	Ours	61.26	**61.67**	**57.18**	**58.00**	56.71	54.34	**57.59**	**56.82**	**54.10**	**54.56**	**55.74**	**54.09**
	$N = 3$												
	Random	60.52	**61.53**	**57.68**	**57.58**	57.05	55.10	**57.48**	**56.63**	**53.83**	**54.25**	**55.06**	**55.17**
	Ours	61.11	**61.69**	**57.33**	**57.87**	**57.39**	55.04	**57.40**	**56.75**	**53.94**	**54.51**	**55.19**	**54.33**
$\|V\| = 8000$	Character	60.74	61.43	56.51	57.35	56.18	54.74	59.71	59.59	56.04	55.97	54.80	53.49
	Byte	59.74	59.43	54.49	56.37	54.65	52.81	**59.74**	58.59	54.94	54.44	54.30	**53.54**
	$N = 2$												
	Random	60.27	60.52	**57.05**	56.39	55.78	54.45	**59.73**	59.46	**56.25**	**56.17**	**55.38**	**54.32**
	Ours	**60.87**	60.46	56.26	57.05	56.04	53.97	**60.36**	**59.79**	**56.45**	**56.23**	**55.31**	**54.65**
	$N = 3$												
	Random	60.06	60.05	**57.02**	56.47	**56.19**	54.24	**60.24**	59.56	**56.55**	55.66	**55.82**	**53.71**
	Ours	60.65	60.08	**56.99**	56.85	56.07	54.07	**60.22**	**59.71**	**56.27**	**56.12**	**55.47**	**54.25**

Table 5: Text classification results on the Amazon Reviews dataset measured by weighted F1 score (%) using BPE tokenization. Results for the BiLSTM model are shown on the left and those for the Transformer model on the right. Scores higher than *Character* are shown in bold, and scores higher than *Byte* are underlined.

perparameters are summarized in Appendix B.1.1. For each run, we select the checkpoint with the best validation score.

Inputs are first encoded by a codebook and then tokenized with a BPE tokenizer. We evaluate codebooks with $N \in \{2, 3\}$ under two token vocabulary size setups: (i) vocabulary size parity with $|V| = |C|$ (as in Table 4), and (ii) a fixed vocabulary size with $|V| = 8000$ for all languages. Both setups enable comparisons under matched token vocabulary sizes across baselines. For each setting, we report the mean over five random seeds. For atom-based methods (*Ours* and *Random*), a new codebook is constructed at each run using the task training data[8].

5.1.2 Results

Table 5 summarizes the text classification results. Under the $|V| = |C|$ setting, *Byte* often outperforms *Character*, especially with the Transformer, confirming the benefit of byte-level information when the token budget is small. Across languages, models, and vocabulary sizes, *Ours* is competitive with *Byte* and *Random*, and sometimes outperforms them depending on the setting. As the vocabulary size increases, differences between encodings generally narrow, particularly for European languages, while subcharacter encodings remain effective for languages with larger character inventories (see the results in Japanese and Chinese in Table 14). Overall, these results indicate that Latom provides a viable alternative to byte-level representations without relying on language-specific inventories.

Finally, *Ours* typically slightly outperforms *Random*. These consistent yet small gaps suggest room for improvement in code-assignment strategies.

[8]Appendix B.3 reports the full experimental results, including other encoding methods (i.e., radical, stroke, and Myte), more vocabulary sizes, and the unigram tokenization.

↓	De	En	Es	Fr	Ja	Zh
Character	0.00	0.00	0.00	0.00	0.00	0.00
Byte	1.54	0.55	1.23	1.12	11.80	11.50
$N = 2$						
Random	**0.14**	**0.18**	**0.25**	**0.17**	**0.45**	**0.77**
Ours	**0.13**	**0.15**	**0.23**	**0.15**	**0.44**	**0.66**
$N = 3$						
Random	2.31	2.78	3.40	3.07	**2.40**	**2.16**
Ours	2.18	2.67	3.00	3.02	**2.07**	**2.11**

Table 6: Ratio (%; **lower is better**) of generated sequences that contain at least one decoding error, measured over 10,000 generations from Transformer language models and averaged over five random seeds. Scores better than *Byte* are shown in bold, and scores better than *Random* are underlined.

5.2 Decoding Errors in Generation

5.2.1 Experimental Setup

While introducing subcharacter units can increase modeling flexibility, it also raises the risk of producing invalid or undecodable outputs during generation (Firestone et al., 2025). We therefore train Transformer language models on the Amazon Re-

views training split and measure the fraction of generated sequences that contain at least one undecodable fragment. Concretely, we define a decoding error as the failure to map a generated atom sequence back to a valid character under the N-digit codebook (i.e., no complete N-digit code is formed or a forbidden interface occurs).

We use the $|V| = |C|$ setup from §5.1 to keep the token budget tight and encourage generation with subcharacter tokens. After training for up to 20 epochs, we select the best checkpoint by validation perplexity, then generate 10,000 sequences with a maximum length of 64 tokens. Starting tokens are drawn uniformly at random from the vocabulary. We then compute the proportion of sequences that include any decoding error.

5.2.2 Results

Table 6 reports decoding-error rates for each encoding. As expected, *Character* yields zero errors, while *Byte* shows non-zero errors due to incomplete or misaligned byte fragments. With atom-level encodings, both *Random* and *Ours* at N=2 consistently reduce decoding errors relative to *Byte*. For Japanese and Chinese, which typically use three-byte UTF-8 representations, N=3 also outperforms *Byte*. Overall, *Ours* is consistently better than *Random*, indicating that the HMM+Hungarian code assignment helps reduce decoding errors compared to uninformed random codebooks.

6 Conclusion

This paper introduces Latom, a framework for learning corpus-dependent, reversible subcharacter encodings by assigning each character a fixed-length sequence of atoms. Using an HMM-guided scoring procedure and a Hungarian assignment, we construct an injective codebook with explicit control over code complexity. Experiments demonstrate that these encodings capture contextual regularities, improve intrinsic metrics across multiple languages, and yield gains in downstream tasks such as classification and generation robustness. These results suggest that base text representations, such as UTF-8 bytes, can be treated as learnable, corpus-sensitive design choices while remaining compatible with standard tokenization pipelines.

Latom is not tied to the specific HMM-based instantiation used in this work. More generally, it defines a framework for learning character–code mappings from corpus statistics, and alternative scoring and assignment strategies may yield improved codebooks. For example, task-aware objectives could replace HMM-based scoring to better align the encoding with downstream tasks. Our results also show that random assignments can occasionally outperform the current implementation, indicating room for improvement in codebook construction. Future work will explore more effective instantiations of Latom along this direction.

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number JP24K20852.

Limitations

We present a corpus-dependent view of subcharacter encodings but acknowledge several limitations and avenues for follow-up work.

Methodological limitations

- **Stochastic code assignment:** Our method assigns codes using HMM posteriors, so different random seeds can yield different codebooks and, in rare cases, noticeably worse performance.
- **Unseen characters:** The model cannot assign codes to characters absent from the training corpus. This could be mitigated by covering likely characters during codebook training.
- **Interpretability at larger N:** While the method allows larger N, codes with $N \geq 3$ may be harder to interpret. Designing human-interpretable interfaces or analysis tools for atom sequences is an interesting direction.
- **Corpus-dependent, but not task-dependent:** The proposed codebooks are learned from raw text statistics of a given corpus and are agnostic to downstream tasks. While this design keeps the encoding simple and reusable across tasks within the same domain, task-specific objectives could potentially yield more specialized codebooks. Exploring task-aware or jointly optimized encodings is left for future work.

Experimental limitations

- **Explored code space:** Although the framework supports large (N, K), our experiments focus on N=2...5 and the smallest K that

covers each character set. Extremely redundant settings (e.g., N=10, K=100) were excluded because they degraded compression and n-gram perplexity. It remains open whether such redundancy could help certain downstream models, especially in multilingual transfer.

- **Variable-length codes:** The method can, in principle, assign variable-length codes by character type (e.g., N=2 for Latin letters and N=3 for Chinese characters). We did not evaluate this setting because heterogeneous lengths increased decoding errors in preliminary tests, similar to byte-level behavior. Constrained decoding or interface-aware language models may alleviate this.

- **Scope of datasets and tasks:** To isolate cross-lingual effects, we chose balanced datasets with similar domains. Broader evaluations are left for future work, such as machine translation, speech-to-text pipelines, or pretraining large language models. Regarding scalability to large language modeling, we note that the intrinsic analysis using bigram perplexity and the decoding-error analysis in the extrinsic evaluation provide model-agnostic indicators that are directly relevant to language modeling behavior.

- **Random vs. learned assignments:** In some settings, random atom assignments outperform our HMM+Hungarian approach. We view this as evidence that there is headroom for alternative objectives and search strategies. Identifying codebooks targeted to specific goals (compression, robustness, or task accuracy) is a promising next step.

References

Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, (2):179–190.

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1):164–171.

Peter F Brown, Vincent J Della Pietra, Peter V Desouza, Jennifer C Lai, and Robert L Mercer. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–480.

Won Ik Cho, Seok Min Kim, and Nam Soo Kim. 2019. Investigating an effective character-level embedding in Korean sentence classification. In *33rd Pacific Asia Conference on Language, Information and Computation, PACLIC 2019*.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun'ichi Tsujii. 2020. CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Preston Firestone, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2025. UTF-8 plumbing: Byte-level tokenizers unavoidably enable LLMs to generate ill-formed UTF-8. In *Second Conference on Language Modeling*.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306, San Diego, California. Association for Computational Linguistics.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2020. Dynamic programming encoding for subword segmentation in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3042–3051, Online. Association for Computational Linguistics.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2020. Optimizing word segmentation for downstream task. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1341–1351, Online. Association for Computational Linguistics.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2021. Joint optimization of tokenization and downstream model. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 244–255, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Roger Hsiao, Liuhui Deng, Erik McDermott, Ruchir Travadi, and Xiaodan Zhuang. 2024. Optimizing byte-level representation for end-to-end ASR. In *2024 IEEE Spoken Language Technology Workshop (SLT)*, pages 462–467. IEEE.

Langlin Huang, Mengyu Bu, and Yang Feng. 2025. MoCE: Adaptive mixture of contextualization experts for byte-based neural machine translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1011–1028, Albuquerque, New Mexico. Association for Computational Linguistics.

Ye Jia, Heiga Zen, Jonathan Shen, Yu Zhang, and Yonghui Wu. 2021. PnG BERT: Augmented BERT on phonemes and graphemes for neural TTS. *arXiv preprint arXiv:2103.15060*.

Haeji Jung, Changdae Oh, Jooeon Kang, Jimin Sohn, Kyungwoo Song, Jinkyu Kim, and David R Mortensen. 2024. Mitigating the linguistic gap with phonemic representations for robust cross-lingual transfer. In *Proceedings of the Fourth Workshop on Multilingual Representation Learning (MRL 2024)*, pages 200–211, Miami, Florida, USA. Association for Computational Linguistics.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Dipesh Kumar and Avijit Thawani. 2022. BPE beyond word boundary: How NOT to use multi word expressions in neural machine translation. In *Proceedings of the Third Workshop on Insights from Negative Results in NLP*, pages 172–179, Dublin, Ireland. Association for Computational Linguistics.

Onur Kuru, Ozan Arkan Can, and Deniz Yuret. 2016. Charner: Character-level named entity recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 911–921.

Sander Land and Catherine Arnett. 2025. Bpe stays on script: Structured encoding for robust multilingual pretokenization. *arXiv preprint arXiv:2505.24689*.

Junyoung Lee, Marco Cognetta, Sangwhan Moon, and Naoaki Okazaki. 2025. Jamo-level subword tokenization in low-resource Korean machine translation. In *Proceedings of the Eighth Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2025)*, pages 66–80, Albuquerque, New Mexico, U.S.A. Association for Computational Linguistics.

Tomasz Limisiewicz, Terra Blevins, Hila Gonen, Orevaoghene Ahia, and Luke Zettlemoyer. 2024. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15059–15076, Bangkok, Thailand. Association for Computational Linguistics.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A. Smith, and Yejin Choi. 2026. SuperBPE: Space travel for language models. In *Tokenization Workshop*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172.

Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. 2019. Glyce: Glyph-vectors for chinese character representations. *Advances in Neural Information Processing Systems*, 32.

Sangwhan Moon, Tatsuya Hiraoka, and Naoaki Okazaki. 2025. Bit-level BPE: Below the byte boundary. *arXiv preprint arXiv:2506.07541*.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia. Association for Computational Linguistics.

Mari Ostendorf, Vassilios V Digalakis, and Owen A Kimball. 2002. From HMM's to segment models: A unified view of stochastic modeling for speech recognition. *IEEE Transactions on speech and audio processing*, 4(5):360–378.

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. 2025. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258, Vienna, Austria. Association for Computational Linguistics.

Lawrence R Rabiner. 2002. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Markusk W. Scherer and Mark Davis. 2006. BOCU-1: MIME-compatible Unicode compression.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Xinlei Shi, Junjie Zhai, Xudong Yang, Zehua Xie, and Chao Liu. 2015. Radical embedding: Delving deeper to Chinese radicals. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 594–598, Beijing, China. Association for Computational Linguistics.

Karl Stratos. 2017. A sub-character architecture for Korean language processing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 721–726, Copenhagen, Denmark. Association for Computational Linguistics.

Yaming Sun, Lei Lin, Nan Yang, Zhenzhou Ji, and Xiaolong Wang. 2014. Radical-enhanced Chinese character embedding. In *International Conference on Neural Information Processing*, pages 279–286. Springer.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2022. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. 2013. Morfessor 2.0: Python implementation and extensions for Morfessor baseline.

Xiaofeng Wu, Karl Stratos, and Wei Xu. 2025. The impact of visual information in Chinese characters: Evaluating large models' ability to recognize and utilize radicals. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 331–350, Albuquerque, New Mexico. Association for Computational Linguistics.

Changshang Xue. 2023. Duncode characters shorter. *arXiv preprint arXiv:2307.05414*.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

Vilém Zouhar, Kalvin Chang, Chenxuan Cui, Nate B. Carlson, Nathaniel Romney Robinson, Mrinmaya Sachan, and David R. Mortensen. 2024. PWESuite: Phonetic word embeddings and tasks they facilitate. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 13344–13355, Torino, Italia. ELRA and ICCL.

A Supplemental Information of Intrinsic Evaluation

A.1 Codebook Examples in German and Japanese

In §4.2.1, we focused on the English codebook. Here, Table 8 and Table 7 present examples of $N = 2$ codebooks obtained with the proposed method in Japanese and German, respectively.

As in the English codebook, punctuation symbols in the German codebook tend to share the same second digit, reflecting similar backward contexts. Even in Japanese, despite the much larger character inventory, punctuation marks such as "。" and "、" are assigned to the same column (`b31`), indicating highly similar right-contexts. These observations suggest that the proposed method assigns codes in a consistent manner across languages.

In addition, the Japanese codebook clusters some syntactic particles in the same row. For example, "で" and "を" are both assigned to row `a14`, where characters that typically follow nouns are concentrated. While characters with similar meanings are occasionally assigned to the same row or column, such as "星" (star) and "月" (moon) in row `a1`, we do not observe strong clustering based on visual radicals or pronunciations. Instead, the codebooks primarily reflect character-level forward and backward contexts and syntactic roles.

A.2 Unigram Tokenizer

Figure 2 and Figure 3 in §4.3 report results obtained with the BPE tokenizer. This subsection presents corresponding results obtained with the unigram tokenizer in Figure 4 and Figure 5.

Relative to BPE, the performance gap between Random and Ours is narrower under unigram tokenization. Nevertheless, the overall trends remain consistent: Character-level encodings are generally outperformed by subcharacter-level encodings, including atom-level representations.

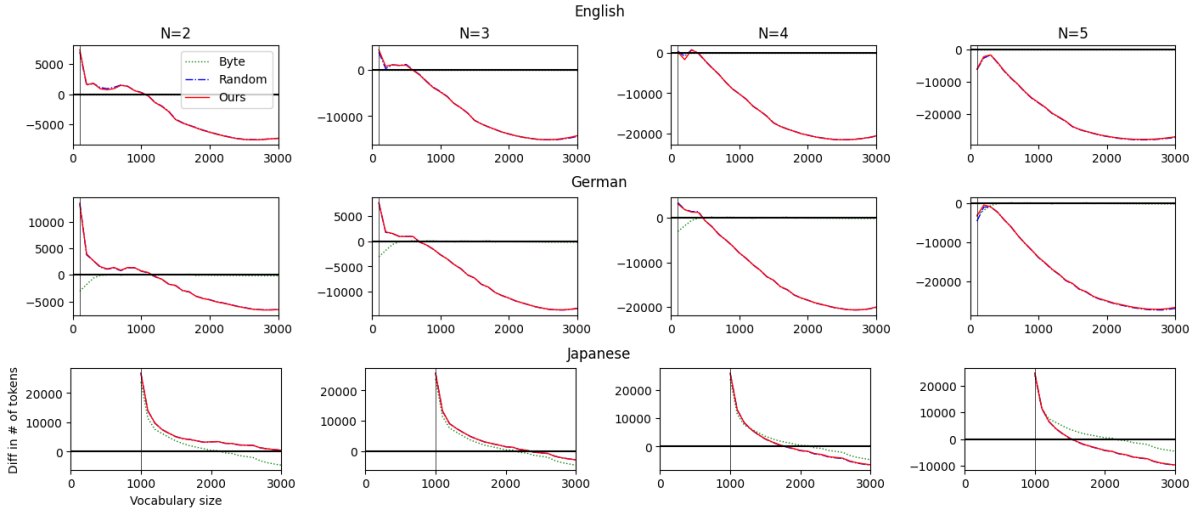

Figure 4: Token-count difference with the unigram tokenizer. **Higher is better**.

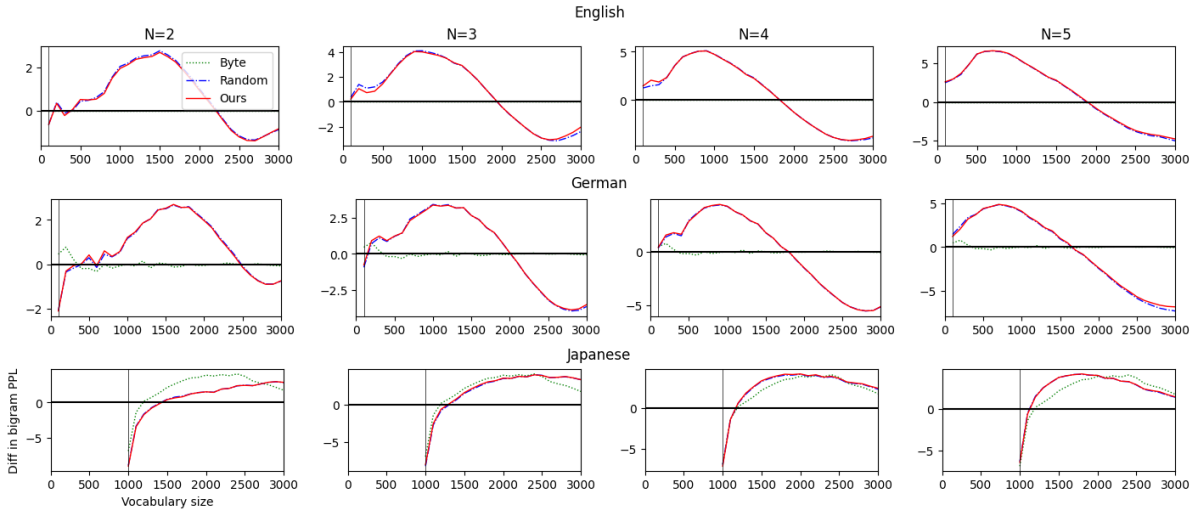

Figure 5: Bigram-perplexity difference with the unigram tokenizer. **Higher is better**.

A.3 Larger K with English Bible

In the main body of this paper, we used the smallest value of K that covers all characters in the corpus. This choice reflects a practical setting in which extensive hyperparameter tuning is undesirable. In this subsection, we explore larger values of K that introduce redundant code capacity.

We focus on the English Bible corpus with $N = 2$, where we originally used $K = 8$ to cover 60 characters. We additionally construct codebooks with $K \in \{9, \ldots, 15\}$. For example, when $K = 15$, the codebook has a capacity of $K^N = 15^2$ codes, providing 165 unused codes beyond those required for the observed character set.

Table 9 shows the codebook obtained with $K = 15$. The overall structure is similar to that of the $K = 8$ codebook in Table 3. Uppercase and lowercase letters are grouped into distinct ranges of first-digit symbols (i.e., `a0` to `a5` and `a6` to `a13`, respectively), and punctuation symbols are clustered into the same second-digit group (i.e., `b12`). In addition, the second-digit symbols `b13` and `b14` are unused, indicating that $K = 15$ provides sufficient capacity for this corpus.

Figure 6 shows token-count differences across values of K. Performance improves steadily for $K = 9$ and $K = 10$, while Figure 7 shows further improvements in bigram perplexity for $K = 11$ and $K = 12$. For clarity, Figure 8 summarizes both token count and bigram perplexity at $|V| = 1500$. These results suggest that suitable combinations of K and N can be selected based on corpus characteristics and language.

		2nd digit								
		b0	b1	b2	b3	b4	b5	b6	b7	b8
1st digit	a0	;	4	s	N	8	t	6	Ü	2
	a1	,	i	q	K	1	ß	G	5	z
	a2	.	b	P	ö	O	o	c	F	
	a3	)	ü	k	v	S	W	Ö	1	M
	a4		x	j	7	I	T	L	C	g
	a5	?	Z	R	-	H	p	:	a	ä
	a6		’	r	u	h	0	V	y	n
	a7	!	B	Q	w	E	m	f	e	J
	a8	Ä	9	_	U	D	(	d	A	3

Table 7: Example learned codebook for German with $N = 2$. Empty cells indicate unused codes, and "_" denotes the whitespace character.

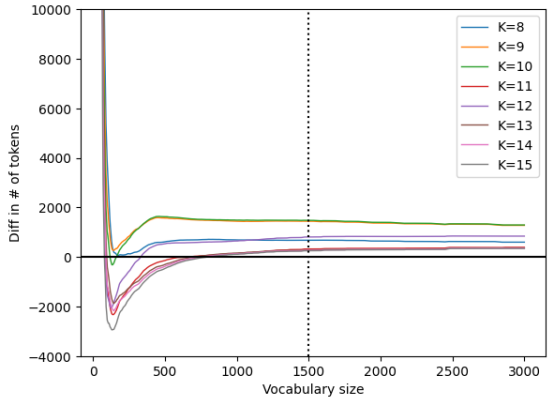

Figure 6: Token-count difference across values of K with $N = 2$ in the English Bible. **Higher is better**; values above zero indicate fewer tokens than the character-level baseline.

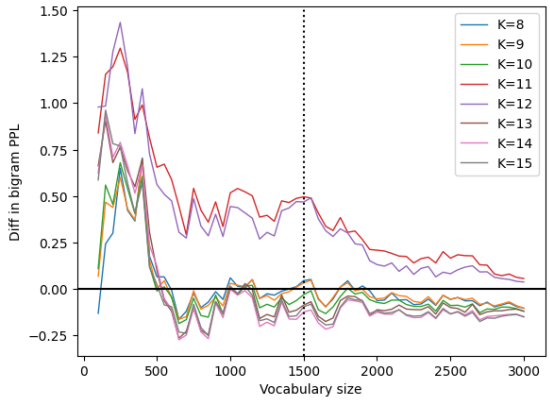

Figure 7: Bigram-perplexity difference across values of K with $N = 2$ in the English Bible. **Higher is better**; values above zero indicate lower perplexity than the character-level baseline.

A.4 In More Languages

For readability, §4 focuses on three languages: English, German, and Japanese. To examine whether the observed trends generalize beyond these settings, Figure 9 and Figure 10 report text compression and bigram perplexity results obtained with BPE tokenizers across 11 languages.

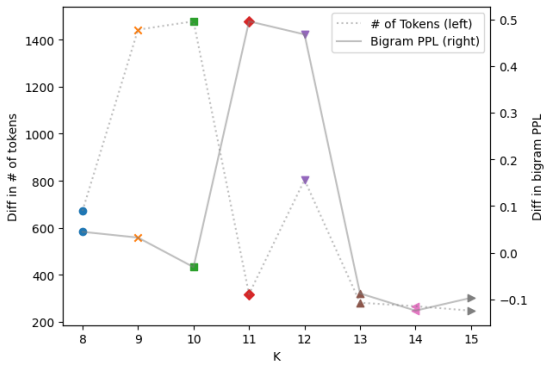

Figure 8: Token count (left y-axis) and bigram perplexity (right y-axis) across values of K at $|V| = 1500$.

A.5 Multilingual Setup

The intrinsic evaluation (§4) focuses on monolingual settings. However, purely monolingual corpora are rare in practical applications. This subsection reports results from multilingual setups constructed by merging Bible corpora from two languages.

The first setup concatenates English and German, where the two languages share many characters and have relatively balanced vocabulary sizes. The second setup concatenates English and Japanese, where the languages do not share alphabets and the vocabulary sizes are highly imbalanced.

Figure 11 and Figure 12 report token-count and bigram-perplexity differences in these multilingual settings. In the English–German setup, the results exhibit characteristics of both English and German observed in the monolingual experiments. Notably, we observe a stable improvement in bigram perplexity with the proposed method, which is less pronounced in the monolingual setting.

In contrast, results in the English–Japanese setup are largely influenced by Japanese, reflecting the larger character inventory. This observation suggests that codebook structure in multilingual settings can be dominated by the language with the larger character set.

A.6 Experiments with Additional Encoding Methods

A.6.1 Methods

In addition to the three baselines described in §4.1.3, we consider three additional encoding meth-

1st digit	2nd digit																															
	b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12	b13	b14	b15	b16	b17	b18	b19	b20	b21	b22	b23	b24	b25	b26	b27	b28	b29	b30	b31
a0	至	刻	長	臨	支	預	リ	訓	素	姿	肉	証	施		首	忘	深	争	霊	北		雨	断	完	処	隊		設	奴	港		や
a1	角	逆	際	薬	償	都	会	樹	変	月	知	矢	命	ボ	沸	離	星	茎	ろ	言	久	平	囚	は	子	別	探	る	肥	養		泉
a2	否	類	嫌	得	器	小	」	位	務	叫	飢	束	風	木	努	礼	土	給	え	音	鎖	舟	連	じ	煙	量	黄	ニ	嘆	酔		れ
a3	袋	孫	並	常	ラ	ヨ	黒	番	神	淵	逃	故	へ		君	激	ヂ	時	全	次	家	力	ル	昼	記	鼓	後	ソ	水	す		射
a4	承	由	気	ど	騎	国	飲	敵	馬	狡	マ	紐	切	折	農	情	糧	ハ	送	求	表	黙	世	高	占	千	陸	確	末	多		十
a5	継	嗣	更	ユ	輪	羊	イ	他	姉	葬	ゾ	明	刃		金	収	ヒ	パ	恨	初	賜	誠	囲	頭	ザ	侍	ガ	入	娘	夜		乳
a6	許	も	召	灌	財	怒	』	姦	疲	永	申	細	能	昨	加	留	撃	天	横	帰	骨	岸	立	ら	配	・	管	注	裸	べ		プ
a7	放	サ	鉄	西	惑	幅	母	訴	婆	勧	銀	認	暮	期	職	信	暑	復	法	大	先	率	花	死	経	威	欠	柔	難	田	芽	泣
a8	組	走	畑	鳥	万	尽	し	傾	暴	元	異	買	三	豊	色	砕	災	破	隠	緒	め	準	武	殺	息	汗	半	福	路	移		回
a9	招	衰	考	火	略	試	治	弟	救	結	夢	び	妻	煮	贈	思	温	男	露	酬	誕	休	督	霜	洩	喪	基	共	去	度		戸
a10	疫	狩	限	本	あ	右	て	笛	裕	ネ	成	攻	役	鍛	郷	空	盟	替	苦	楽	歯	解	服	墓	麦	腰	選	糸	嫁	ジ	扱	携
a11	創	終		周	師	裂	理	『	沈	達	海	与	へ		仰	換	部	雲	造	置	腕	刈	レ	野		額		地	強	焼		食
a12	六	コ	涼	ロ	制	惜	公	齢	宝	御	滅	増	取		オ	唱	面	牧	陪	葦	議	歳	杯	再	緋	計		ケ	安	わ		及
a13	悪	グ	咲	止	兵	九	作	陣	浮	う	声	氏	誤	巻	照	衣	旅	参	同	間	指	統	ホ	幼	麻	栄		閉	付	尋	捕	互
a14	形	受	粉	守	潔	特	で	印	盾	り	肩	園	胎	蜜	襲	責	ァ	女	を	よ	方	話	ン	耕	向	抱	暗	姻	草	汁	聖	浪
a15	遠	非	根	穂	不	決	炎	新	消	術	穫	二	穏	猾	恐	陰	事	盗	雇	左	弓	ざ	諸	川	呼	燔	違	砂	ぎ	何		授
a16	無	王	促	血	雄	自	ャ	戦	執	相	洗	納	出	央	請	醜	語	訳	笑	遊	種	石	用	上	布	ペ	退	幾	寝	打	。	住
a17	乗	ミ	行	飾	春	祖	慕	豆	交	七	か	そ	デ	「	屋	白	塚	婦	祭	営	痛	光	料	築	ダ	厚		弱	滅	五	邦	べ
a18	丘	端	晴	重	当	拠	群	流	ご	季	欺	医	ぬ	四	除	望	最	始	バ	悩	発	魚	ず	油	第	に	漂	枝	掛	図		ぜ
a19	這	琴	業	虐	渡	軽	供	進	境	憶	困	真	壇	ゴ	階	迷	頂	生	被	調	拝	残	意	ト	使	眠		有	答	柱		泊
a20	穴	過	覧	々	育	働	顧	驚	盛	待	代	顔	場	保	罪	ふ	葉	勝	活	ば	果	忍	見	対	隔	ブ	太	週	問	好	幻	味
a21	偽	メ	騒	が	年	皮	セ	衛	原	伸	カ	ナ	満		朝	腹	投	報	着	け	荒	ュ	兄	ク		老	熟	正	寒	こ		埋
a22	戯	悔	下	前	絶	権	所	汚	赤	硫	託	動	避	従	可	恵	近	荷	獣	飛	来	外	隷	拒	祈	殺	ド	せ	産	名	垂	富
a23	要	告	源	張	ズ	振	寄	愚	貨	憎	畜	道	軍	毎	象	触	害	慰	げ	ひ	魔	犠	建	灰	塩	持		き	良	通	、	お
a24	裏	犯	塗	づ	井	妹	誓	目	牲	免	府	谷	町	病	届	悲	包	タ	幕	起	集	親	押	岩	浜	飼	ツ	分	の	傷		割
a25	亜	ぐ	ゲ	系	沿	フ	巧	賢	恋	村	南	忌	ん	整	ゅ	な	む	酒	少	身	凝	諾	ゆ	列	銅	獲	足	ヅ	東	者		冬
a26	差	散	節	祝	体	棺	つ	幸	途	歩	鉢	ギ	赴	倉	ム	さ	引	備	運	掘	ま	物	日	助	士	聞	品	約	た	実		青
a27	勢	ぶ	辞	価	魂	数	善	負	植	領	乱	ピ	ポ	伏	宿	キ	友	失	ョ	ウ	戒	述	潤	香	官	恥		一	ぞ	ほ		契
a28	越	現	伝	美	皆	捨	合	縛	耳	門	ス	筋	寡	開	奪	讐	窓	シ	脱	売	返	以	歌	浸	塔	内	労	監	毛	払		勇
a29	エ	み	憤	箱	ビ	急	ゼ		費	夏	今	若	ぼ		ヤ	獄	示	願	続	ノ	虜	チ	ア	ッ	タ	載	洪	父	彼	ょ		主
a30	ワ	巡	練	必	翼	仕	人	淫	心	雌	八	捜	倍	く	導	車	清	義	禁	愛	儀	迎	だ	山	夫	と	手	早	迫	広	席	商
a31	落	モ	喜	降	婚	中	民	口	吹	震	覚	っ	昔	床	任	定	ね	百	牛	鼻	罰	い	低	族	頼	司	益	ち	追	猟		テ

Table 8: Example learned codebook for Japanese with $N = 2$. Empty cells indicate unused codes.

1st digit	2nd digit														
	b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12	b13	b14
a0	F	T	M	U	S	H	W	Y	B	N	I	A			
a1	E	G	O	C											
a2		K	L												
a3					Z	D									
a4	P												?		
a5	R														
a6		j	x	p	e			b					!		
a7	z	m	a	i	y	w			_						
a8	u	J	n			t							)		
a9		h	d		c								;		
a10		v	q		-		f						:		
a11	l	r	s		(								,		
a12		g			o										
a13					k										
a14													.		

Table 9: Example learned codebook for English with $N = 2$ and $K = 15$. Empty cells indicate unused codes, and "_" denotes the whitespace character.

ods for comparison.

Radical Chinese characters can be decomposed into meaningful components known as radicals. For example, "海" (sea) consists of "氵" (water) and "毎" (every). Radicals are sometimes used as subcharacter features in Chinese and Japanese NLP. Accordingly, we compare the proposed method with radical-level encoding. We use the CHISE-IDS dictionaries[9] to decompose characters in the Chinese and Japanese datasets into their smallest radical components.

Stroke Beyond radical-level representations, Chinese characters can also be expressed as sequences of strokes. For example, "大" can be represented as "一丿㇏". We include stroke-level encoding as another existing subcharacter representation. Stroke directions are extracted from an open-source dictionary[10].

MYTE In addition to subcharacter-based approaches, we also consider recent work that proposes alternatives to byte-level encoding. We select MYTE (Limisiewicz et al., 2024) as a representative method. We use the official MYTE implementation to encode the raw text[11].

[9] https://www.chise.org/ids/

[10] https://www.skishore.me/makemeahanzi/

[11] https://github.com/tomlimi/MYTE/

A.6.2 Experimental Results

Figure 13 and Figure 14 show token-count and bigram-perplexity results for Japanese and Chinese. Stroke-level decomposition performs poorly on both metrics, while radical-level encoding sometimes achieves lower bigram perplexity than the proposed method. In contrast, the proposed method consistently yields the smallest token counts, highlighting the importance of selecting encoding strategies that align with corpus statistics. MYTE also performs well in terms of bigram perplexity in both languages.

A.7 Intrinsic Evaluation on Unseen Dataset

The intrinsic evaluation in §4 constructs and evaluates codebooks on the same corpus, which is sufficient for analyzing the basic behavior of the proposed method. However, evaluating performance on unseen data is also important. Accordingly, we conduct additional experiments on the Amazon Reviews dataset introduced in §5.

In this subsection, we construct codebooks with $N \in \{2, 3\}$ and the smallest feasible K using the training split of the Amazon dataset. We then evaluate compression performance and bigram perplexity on the validation split, considering vocabulary sizes $|V| \in \{|C|, 8000, 16000, 24000, 32000\}$.

Figure 15 reports token counts after tokenization. Consistent with the intrinsic evaluation, all subcharacter-level methods reduce token counts under the $|V| = |C|$ setting. The code-based methods (Ours and Random) generally outperform character- and byte-level encodings, except for Japanese and Chinese when $N = 2$.

Figure 16 reports bigram perplexity after tokenization. Across languages, perplexity decreases as $|V|$ increases. The proposed method performs stably better than character- and byte-level encodings for $|V| \geq 16000$. These results suggest that corpus-dependent atomic encodings generalize beyond the training corpus and remain effective on unseen data.

B Supplemental Information of Extrinsic Evaluation

B.1 Hyperparameters

This section provides detailed information on the hyperparameters used in §5.

	$\lvert C\rvert$	K^2	K^3
English (En)	302	18^2	7^3
Spanish (Es)	275	17^2	7^3
German (De)	403	21^2	8^3
French (Fr)	375	20^2	8^3
Japanese (Ja)	4169	65^2	17^3
Chinese (Zh)	5565	75^2	18^3

Table 10: Minimum values of K for each code length N such that $K^N \geq |C|$ in the Amazon Reviews dataset.

B.1.1 Text Classification

Table 10 lists the values of K used in the experiments. We selected the minimum K that covers all characters in the training dataset.

For the BiLSTM classifier, we used a three-layer bidirectional LSTM followed by a linear layer. A sentence representation is obtained by applying max pooling over all time steps to the output of the linear layer. The resulting sentence vector is then passed to another linear layer for label prediction. During training, we applied dropout with $p = 0.5$.

For the Transformer classifier, we used a three-layer Transformer encoder with four attention heads. As with the BiLSTM classifier, a sentence vector is obtained via max pooling and passed to a linear layer to compute label probabilities. The maximum input length was set to 5,000 tokens, which was sufficient for all samples in the training, validation, and test splits. We applied dropout with $p = 0.2$.

For both architectures, the token embedding size and hidden state size were set to 256. We used the Adam optimizer (Kingma, 2014) with a learning rate of 0.001 for the BiLSTM classifier and 0.0001 for the Transformer classifier.

B.1.2 Language Modeling

We used a four-layer Transformer decoder with four attention heads and an embedding size of 256. During training, we used training sequences of length 128 tokens. The language model was trained with AdamW (Loshchilov and Hutter, 2017) using a learning rate of 0.0003.

During text generation, we used a temperature of 1.0 with top-50 sampling.

B.2 Implementation Details

All experimental code was implemented in Python. We used `pomegranate` for HMM training and `scipy` for the Hungarian algorithm. Neural network models were implemented using PyTorch.

All experiments were conducted on NVIDIA A40 GPUs in a cloud environment. The total GPU cost for the experiments was approximately USD 2,400, most of which was incurred by the text classification experiments with multiple configurations.

The runtime of the proposed method depends on the size of the input corpus and the values of N and K. For code assignment on the English Bible corpus, the process took less than three minutes. For the Amazon Reviews dataset, code assignment required approximately 1.5 hours.

B.3 Completed Experimental Results of Text Classification

The baselines in Table 5 focus on three decodable encoding methods and a limited set of vocabulary sizes. This subsection extends the comparison to additional encoding methods described in §A.6.1, namely radical, stroke, and MYTE, and to a wider range of vocabulary sizes ($|V| \in \{8000, 16000, 32000\}$). In addition, we report results obtained with the unigram tokenizer and without tokenization.

Table 14 and Table 15 present the complete text classification results with BPE and unigram tokenizers, respectively. Across both tokenization methods, similar trends are observed: the proposed method tends to improve performance in European languages at smaller vocabulary sizes, while improvements are more pronounced at larger vocabulary sizes for Chinese and Japanese.

Table 11 reports results obtained without tokenization. Because decomposition-based encodings substantially increase input length (e.g., $N = 3$ increases sequence length by a factor of three), Transformer-based classifiers often encounter out-of-memory errors. We therefore report results only for the BiLSTM classifier in this setting.

Overall, the results without tokenization exhibit trends similar to those observed with tokenizers. Classifiers that incorporate subcharacter information, including the proposed method, tend to perform better in European languages under smaller vocabulary sizes.

B.4 Standard Deviation

For all extrinsic evaluation results, we report the standard deviation over five runs. Table 16, Table 17, and Table 12 report standard deviations for text classification results. In addition, Table 13 reports standard deviations for decoding error rates.

W/O Tokenizer ↑		BiLSTM					
		De	En	Es	Fr	Ja	Zh
	Character	60.42	59.82	56.11	56.74	56.71	53.87
	Byte	**60.64**	**59.97**	**57.44**	56.28	55.24	**54.03**
	Myte	**60.63**	**60.43**	**56.67**	**57.10**	54.75	53.08
	Radical	-	-	-	-	55.37	**54.42**
	Stroke	-	-	-	-	54.32	50.16
	$N = 2$						
	Random	**61.14**	**60.75**	**57.11**	**57.23**	55.78	**54.17**
	Ours	**60.56**	**61.17**	**57.11**	**57.52**	56.55	**54.41**
	$N = 3$						
	Random	**61.07**	**60.63**	**57.26**	56.67	55.49	**54.42**
	Ours	**61.06**	**60.66**	**56.81**	**57.29**	56.03	**54.34**

Table 11: Text classification results without tokenization. Scores higher than *Character* are shown in bold, and scores higher than *Byte* are underlined.

W/O Tokenizer ↑		BiLSTM					
		De	En	Es	Fr	Ja	Zh
	Character	0.60	0.52	0.29	0.71	0.34	0.74
	Byte	0.96	0.69	0.45	0.66	0.31	0.52
	Myte	0.43	0.51	0.37	0.23	0.32	0.42
	Radical	-	-	-	-	0.37	1.14
	Stroke	-	-	-	-	0.71	0.74
	$N = 2$						
	Random	0.50	0.49	0.54	0.16	0.38	0.35
	Ours	0.28	0.38	0.55	0.34	0.50	0.65
	$N = 3$						
	Random	0.59	0.28	0.79	0.45	0.33	0.57
	Ours	0.51	0.38	0.35	0.45	0.30	0.47

Table 12: Standard deviation corresponding to Table 11.

±	En	Es	De	Fr	Ja	Zh
Character	0.00	0.00	0.00	0.00	0.00	0.00
Byte	0.04	0.07	0.11	0.05	0.06	0.11
$N = 2$						
Random	0.03	0.13	0.04	0.03	0.05	0.10
Ours	0.05	0.09	0.10	0.06	0.03	0.14
$N = 3$						
Random	0.21	0.18	0.35	0.16	0.35	0.22
Ours	0.18	0.32	0.28	0.52	0.35	0.20

Table 13: Standard deviation corresponding to Table 6.

↑		BiLSTM						Transformer					
		De	En	Es	Fr	Ja	Zh	De	En	Es	Fr	Ja	Zh
$\|V\| = \|C\|$	Character	61.41	58.99	56.06	57.35	57.23	55.24	48.89	52.22	50.14	50.01	52.27	52.31
	Byte	59.52	**60.71**	**56.14**	56.88	56.14	54.35	**57.49**	**55.54**	**53.30**	**53.30**	**54.33**	**53.41**
	Myte	60.33	**60.36**	**56.53**	56.30	55.15	54.08	**56.26**	**55.44**	**52.91**	**52.96**	51.82	**52.68**
	Radical	-	-	-	-	55.15	54.08	-	-	-	-	51.82	**52.68**
	Stroke	-	-	-	-	54.69	51.89	-	-	-	-	**53.09**	50.50
	$N = 2$												
	Random	**62.04**	**61.36**	**57.95**	**57.75**	56.59	54.69	**57.28**	**56.59**	**53.92**	**54.00**	**55.08**	**54.12**
	Ours	61.26	**61.67**	**57.18**	**58.00**	56.71	54.34	**57.59**	**56.82**	**54.10**	**54.56**	**55.74**	**54.09**
	$N = 3$												
	Random	60.52	**61.53**	**57.68**	**57.58**	57.05	55.10	**57.48**	**56.63**	**53.83**	**54.25**	**55.06**	**55.17**
	Ours	61.11	**61.69**	**57.33**	**57.87**	**57.39**	55.04	**57.40**	**56.75**	**53.94**	**54.51**	**55.19**	**54.33**
$\|V\| = 8000$	Character	60.74	61.43	56.51	57.35	56.18	54.74	59.71	59.59	56.04	55.97	54.80	53.49
	Byte	59.74	59.43	54.49	56.37	54.65	52.81	**59.74**	58.59	54.94	54.44	54.30	**53.54**
	Myte	58.28	60.37	55.81	56.12	55.23	53.25	59.13	58.85	55.65	55.64	51.88	51.42
	Radical	-	-	-	-	55.23	53.25	-	-	-	-	51.88	51.42
	Stroke	-	-	-	-	55.45	51.28	-	-	-	-	53.40	50.67
	$N = 2$												
	Random	60.27	60.52	**57.05**	56.39	55.78	54.45	**59.73**	59.46	**56.25**	**56.17**	**55.38**	**54.32**
	Ours	**60.87**	60.46	56.26	57.05	56.04	53.97	**60.36**	**59.79**	**56.45**	**56.23**	**55.31**	**54.65**
	$N = 3$												
	Random	60.06	60.05	**57.02**	56.47	**56.19**	54.24	**60.24**	59.56	**56.55**	55.66	**55.82**	**53.71**
	Ours	60.65	60.08	**56.99**	56.85	56.07	54.07	**60.22**	**59.71**	**56.27**	**56.12**	**55.47**	**54.25**
$\|V\| = 16000$	Character	58.62	60.12	55.60	56.31	56.19	54.15	59.50	58.84	55.97	56.36	55.59	53.56
	Byte	**58.98**	58.11	**55.71**	54.20	54.50	52.68	58.76	58.59	**56.40**	54.38	53.49	52.37
	Myte	**59.13**	59.06	54.58	54.72	53.90	53.41	58.95	58.44	55.13	54.91	51.50	51.82
	Radical	-	-	-	-	53.90	53.41	-	-	-	-	51.50	51.82
	Stroke	-	-	-	-	54.88	50.19	-	-	-	-	52.18	51.62
	$N = 2$												
	Random	**59.54**	59.49	**55.65**	55.87	55.61	53.30	58.82	**59.23**	55.70	56.04	54.93	53.29
	Ours	**60.33**	59.48	**55.99**	55.57	55.35	54.13	**59.75**	**59.71**	**56.19**	56.04	55.40	**53.94**
	$N = 3$												
	Random	**58.84**	59.45	55.21	54.83	55.07	53.63	**59.75**	**59.47**	**56.03**	55.78	55.41	**53.68**
	Ours	**60.58**	59.47	55.23	55.58	55.52	53.66	59.37	**59.01**	**56.84**	**56.50**	**55.68**	**53.71**
$\|V\| = 32000$	Character	59.15	59.58	54.85	54.42	54.43	51.59	58.75	58.00	56.42	55.52	54.06	53.22
	Byte	58.26	57.86	**55.31**	53.90	52.93	51.35	57.79	**58.11**	54.85	53.36	54.03	52.82
	Myte	57.92	58.68	53.65	53.89	**55.21**	**52.81**	57.94	**58.37**	54.77	54.28	51.83	52.57
	Radical	-	-	-	-	**55.21**	**52.81**	-	-	-	-	51.83	52.57
	Stroke	-	-	-	-	53.76	50.56	-	-	-	-	51.92	51.34
	$N = 2$												
	Random	58.45	58.85	**55.09**	**55.13**	**55.07**	**53.11**	58.31	**58.74**	55.37	54.85	**55.20**	53.18
	Ours	58.68	58.48	54.10	**55.30**	**55.04**	**53.60**	**58.85**	**59.07**	55.53	55.34	**54.75**	**53.42**
	$N = 3$												
	Random	58.69	58.83	54.35	**54.82**	**54.92**	**52.68**	57.98	**58.51**	54.95	55.02	53.63	**53.30**
	Ours	58.61	58.75	54.07	**54.96**	54.19	**53.35**	**59.47**	**59.24**	55.45	**55.68**	**55.32**	**53.89**

Table 14: Text classification results on the Amazon Reviews dataset measured by weighted F1 score (%) using BPE tokenization. Results for the BiLSTM model are shown on the left and those for the Transformer model on the right. Scores higher than *Character* are shown in bold, and scores higher than *Byte* are underlined.

↑		BiLSTM						Transformer					
		De	En	Es	Fr	Ja	Zh	De	En	Es	Fr	Ja	Zh
$\|V\| = \|C\|$	Character	61.41	58.99	56.06	57.35	57.23	55.24	48.89	52.22	50.14	50.01	52.27	52.31
	Byte	59.75	**60.96**	**56.80**	56.34	57.04	54.73	**55.71**	**55.85**	**52.15**	**53.43**	**54.48**	**52.55**
	Myte	60.60	**60.86**	**57.00**	55.62	55.40	53.04	**56.20**	**56.27**	**53.18**	**53.21**	52.18	50.87
	Radical	-	-	-	-	55.40	53.04	-	-	-	-	52.18	50.87
	Stroke	-	-	-	-	56.14	54.45	-	-	-	-	**52.33**	**53.23**
	$N = 2$												
	Random	**61.68**	**61.40**	**57.83**	**57.59**	56.99	54.17	**57.48**	**56.90**	**53.84**	**54.20**	**55.20**	**53.94**
	Ours	61.38	**61.64**	**57.92**	**57.59**	56.37	54.37	**57.48**	**57.63**	**53.89**	**54.24**	**55.32**	**54.01**
	$N = 3$												
	Random	60.90	**61.73**	**57.21**	**57.47**	57.14	54.77	**57.10**	**56.95**	**53.38**	**54.01**	**55.92**	**53.49**
	Ours	60.92	**61.54**	**57.69**	**57.50**	**57.85**	54.39	**57.29**	**57.55**	**53.89**	**54.53**	**55.96**	**54.41**
$\|V\| = 8000$	Character	60.52	60.36	56.78	56.71	56.90	54.33	60.10	60.09	56.29	56.98	56.38	53.39
	Byte	60.23	60.26	56.54	56.42	56.20	53.46	59.85	59.07	56.28	56.62	54.67	**54.12**
	Myte	**60.59**	59.71	55.42	56.49	56.19	53.35	59.45	59.50	55.62	56.16	51.82	51.60
	Radical	-	-	-	-	56.19	53.35	-	-	-	-	51.82	51.60
	Stroke	-	-	-	-	55.56	52.55	-	-	-	-	52.55	**53.45**
	$N = 2$												
	Random	**60.81**	**61.41**	**57.46**	**56.84**	56.74	53.91	**60.74**	59.01	**57.00**	56.61	55.49	**53.86**
	Ours	**61.07**	**61.29**	**57.24**	**57.58**	**57.01**	53.80	**60.41**	**60.68**	**57.53**	56.93	55.73	**54.09**
	$N = 3$												
	Random	60.12	**60.56**	56.78	**56.74**	56.48	54.17	59.75	59.90	**56.55**	56.21	55.30	**53.65**
	Ours	**60.72**	**60.47**	**57.08**	**56.89**	56.55	**54.58**	**60.29**	**60.41**	**57.15**	**57.06**	55.94	**54.60**
$\|V\| = 16000$	Character	59.68	60.00	55.30	55.35	55.00	53.50	59.61	58.49	56.70	55.76	55.75	52.68
	Byte	58.86	58.36	**55.36**	**56.06**	54.29	**53.51**	59.55	**59.18**	56.05	53.96	54.21	**53.68**
	Myte	59.17	59.30	**55.79**	52.91	**55.24**	**53.85**	**59.63**	**59.41**	**57.05**	55.38	51.91	51.97
	Radical	-	-	-	-	**55.24**	**53.85**	-	-	-	-	51.91	51.97
	Stroke	-	-	-	-	**55.35**	53.06	-	-	-	-	51.85	**52.80**
	$N = 2$												
	Random	59.29	59.99	**56.24**	**56.73**	**55.73**	**53.55**	**60.34**	**59.50**	56.50	**56.24**	55.27	**53.21**
	Ours	**60.24**	59.65	**56.03**	**56.42**	**56.16**	**53.90**	**60.25**	**59.80**	56.65	**56.74**	55.23	**54.29**
	$N = 3$												
	Random	**60.16**	59.85	**56.23**	**55.60**	**55.72**	**53.70**	59.56	**58.75**	56.67	**56.32**	54.99	**54.03**
	Ours	59.34	59.90	**56.15**	**56.04**	**55.38**	**53.88**	**59.66**	**60.31**	56.32	**56.30**	55.07	**53.46**
$\|V\| = 32000$	Character	59.33	58.24	55.41	54.95	55.03	52.29	58.76	59.30	55.61	54.99	54.62	52.27
	Byte	57.24	**58.87**	53.73	53.96	53.95	**52.67**	**58.89**	58.54	**55.82**	**55.11**	54.22	52.07
	Myte	58.05	**58.76**	**56.40**	54.36	**55.37**	**54.42**	**59.18**	58.66	54.84	54.29	50.84	51.95
	Radical	-	-	-	-	**55.37**	**54.42**	-	-	-	-	50.84	51.95
	Stroke	-	-	-	-	53.95	**52.59**	-	-	-	-	52.18	**53.01**
	$N = 2$												
	Random	59.10	**58.33**	**56.05**	54.80	54.91	**52.46**	**59.29**	58.71	55.36	**55.68**	53.68	**52.99**
	Ours	59.11	**58.53**	55.38	**56.18**	**55.21**	**53.43**	**59.17**	58.90	55.49	**55.65**	**55.01**	**54.04**
	$N = 3$												
	Random	58.95	**59.26**	55.02	**55.08**	54.95	**52.52**	58.72	59.27	55.31	**55.88**	53.42	**53.46**
	Ours	58.98	**59.57**	55.16	**55.79**	54.17	**53.18**	**59.50**	59.25	**55.99**	**55.65**	54.25	**52.77**

Table 15: Text classification results on the Amazon Reviews dataset measured by weighted F1 score (%) using unigram tokenization. Results for the BiLSTM model are shown on the left and those for the Transformer model on the right. Scores higher than *Character* are shown in bold, and scores higher than *Byte* are underlined.

↑		BiLSTM						Transformer					
		De	En	Es	Fr	Ja	Zh	De	En	Es	Fr	Ja	Zh
$\|V\| = \|C\|$	Character	0.36	0.03	0.19	0.34	0.32	0.74	0.33	0.42	0.69	0.56	0.19	0.25
	Byte	0.30	0.35	0.06	0.30	0.79	0.68	0.29	0.55	0.25	0.29	0.20	0.27
	Myte	0.21	0.25	0.19	0.19	0.30	0.21	0.24	0.05	0.13	0.21	0.14	0.31
	Radical	-	-	-	-	0.30	0.21	-	-	-	-	0.14	0.31
	Stroke	-	-	-	-	0.50	0.58	-	-	-	-	0.17	0.36
	$N = 2$												
	Random	0.65	0.28	0.45	0.25	0.89	0.77	0.25	0.22	0.23	0.29	0.39	0.26
	Ours	0.21	0.19	0.24	0.58	0.56	0.80	0.23	0.17	0.17	0.27	0.13	0.16
	$N = 3$												
	Random	0.27	0.16	0.32	0.39	0.93	1.31	0.28	0.28	0.42	0.24	0.31	0.20
	Ours	0.25	0.22	0.32	0.40	0.75	0.57	0.19	0.18	0.24	0.20	0.29	0.28
$\|V\| = 8000$	Character	0.79	0.79	0.68	0.71	1.02	0.92	0.19	0.17	0.21	0.23	0.11	0.13
	Byte	0.63	0.40	0.25	1.07	0.91	1.04	0.15	0.40	0.36	0.15	0.16	0.20
	Myte	0.48	0.69	0.83	0.83	0.34	0.20	0.10	0.15	0.22	0.45	0.16	0.07
	Radical	-	-	-	-	0.34	0.20	-	-	-	-	0.16	0.07
	Stroke	-	-	-	-	0.37	0.39	-	-	-	-	0.32	0.21
	$N = 2$												
	Random	0.58	0.80	1.31	0.70	1.02	0.88	0.25	0.08	0.38	0.36	0.12	0.30
	Ours	0.62	0.59	0.57	0.68	0.62	0.66	0.42	0.08	0.09	0.25	0.28	0.25
	$N = 3$												
	Random	0.50	0.62	0.59	0.48	0.61	0.82	0.22	0.15	0.12	0.28	0.29	0.28
	Ours	1.22	0.55	1.23	0.92	0.41	1.15	0.30	0.15	0.13	0.30	0.07	0.15
$\|V\| = 16000$	Character	0.33	0.85	0.89	0.80	0.75	1.45	0.42	0.24	0.22	0.23	0.43	0.11
	Byte	0.72	0.73	0.93	0.66	0.45	0.84	0.34	0.22	0.40	0.35	0.35	0.21
	Myte	0.64	0.72	0.75	0.35	0.33	0.38	0.14	0.16	0.08	0.32	0.37	0.18
	Radical	-	-	-	-	0.33	0.38	-	-	-	-	0.37	0.18
	Stroke	-	-	-	-	0.53	0.76	-	-	-	-	0.45	0.20
	$N = 2$												
	Random	1.21	0.82	0.76	0.73	0.49	1.23	0.31	0.10	0.42	0.19	0.25	0.32
	Ours	0.94	0.47	0.75	0.94	0.71	0.42	0.46	0.17	0.25	0.18	0.53	0.27
	$N = 3$												
	Random	0.91	0.93	0.68	0.37	0.81	1.05	0.23	0.17	0.26	0.14	0.20	0.29
	Ours	0.76	0.58	0.44	0.50	0.53	1.09	0.16	0.19	0.14	0.10	0.31	0.24
$\|V\| = 32000$	Character	0.85	1.02	0.68	0.75	0.87	0.83	0.29	0.22	0.30	0.07	0.15	0.24
	Byte	0.61	0.72	1.35	0.58	0.75	0.38	0.17	0.27	0.16	0.12	0.30	0.08
	Myte	0.86	0.81	0.96	0.35	0.45	0.43	0.09	0.38	0.23	0.20	0.31	0.39
	Radical	-	-	-	-	0.45	0.43	-	-	-	-	0.31	0.39
	Stroke	-	-	-	-	0.32	0.47	-	-	-	-	0.25	0.17
	$N = 2$												
	Random	0.72	0.78	0.97	0.67	0.52	0.63	0.36	0.26	0.24	0.58	0.39	0.41
	Ours	0.85	1.00	0.76	1.08	0.65	1.08	0.28	0.18	0.30	0.22	0.30	0.16
	$N = 3$												
	Random	0.77	0.72	0.61	1.18	0.93	0.85	0.40	0.03	0.18	0.28	0.33	0.20
	Ours	1.01	0.56	0.76	0.75	1.09	1.76	0.18	0.19	0.37	0.14	0.19	0.13

Table 16: Standard deviation of the text classification results reported in Table 14.

$\uparrow$		BiLSTM						Transformer					
		De	En	Es	Fr	Ja	Zh	De	En	Es	Fr	Ja	Zh
$\|V\| = \|C\|$	Character	0.36	0.03	0.19	0.34	0.32	0.74	0.33	0.42	0.69	0.56	0.19	0.25
	Byte	0.39	0.33	0.14	0.26	0.74	0.90	0.28	0.28	0.27	0.25	0.46	0.08
	Myte	0.44	0.45	0.21	0.47	0.30	0.43	0.18	0.23	0.13	0.43	0.27	0.14
	Radical	-	-	-	-	0.30	0.43	-	-	-	-	0.27	0.14
	Stroke	-	-	-	-	0.39	0.95	-	-	-	-	0.21	0.25
	$N = 2$												
	Random	0.43	0.32	0.14	0.32	0.45	0.52	0.12	0.15	0.26	0.15	0.13	0.07
	Ours	0.09	0.33	0.25	0.44	0.47	0.93	0.31	0.21	0.18	0.14	0.35	0.20
	$N = 3$												
	Random	0.35	0.11	0.18	0.38	0.59	1.01	0.21	0.16	0.11	0.31	0.15	0.19
	Ours	0.40	0.22	0.30	0.39	0.92	0.52	0.35	0.09	0.33	0.06	0.41	0.46
$\|V\| = 8000$	Character	0.74	0.52	1.26	0.47	0.61	0.52	0.16	0.23	0.11	0.39	0.34	0.28
	Byte	0.64	0.85	0.47	0.87	0.50	0.43	0.40	0.16	0.21	0.23	0.16	0.29
	Myte	0.99	0.41	0.39	0.69	0.81	0.56	0.19	0.08	0.22	0.29	0.27	0.20
	Radical	-	-	-	-	0.81	0.56	-	-	-	-	0.27	0.20
	Stroke	-	-	-	-	0.57	0.99	-	-	-	-	0.27	0.29
	$N = 2$												
	Random	0.60	0.56	0.59	0.99	0.98	0.50	0.24	0.40	0.16	0.30	0.08	0.45
	Ours	0.42	0.81	0.84	1.08	0.55	0.81	0.12	0.31	0.06	0.22	0.13	0.27
	$N = 3$												
	Random	0.58	0.62	0.71	1.09	0.74	0.97	0.15	0.15	0.17	0.09	0.31	0.04
	Ours	0.72	0.73	1.00	0.83	0.60	0.96	0.32	0.08	0.23	0.08	0.32	0.29
$\|V\| = 16000$	Character	0.42	0.89	0.80	0.71	0.57	0.78	0.18	0.15	0.31	0.17	0.28	0.29
	Byte	0.49	0.40	0.50	0.88	0.60	1.34	0.14	0.23	0.31	0.20	0.34	0.55
	Myte	0.69	0.69	0.74	0.39	0.26	0.58	0.21	0.24	0.38	0.12	0.11	0.23
	Radical	-	-	-	-	0.26	0.58	-	-	-	-	0.11	0.23
	Stroke	-	-	-	-	0.50	0.80	-	-	-	-	0.37	0.24
	$N = 2$												
	Random	0.97	0.74	0.93	0.78	0.83	0.84	0.17	0.17	0.18	0.28	0.24	0.33
	Ours	0.75	0.80	0.45	0.91	0.33	0.99	0.34	0.32	0.15	0.18	0.18	0.32
	$N = 3$												
	Random	0.76	0.42	0.92	0.72	0.91	0.47	0.10	0.25	0.08	0.14	0.19	0.33
	Ours	0.82	0.77	1.14	0.36	0.72	0.82	0.31	0.17	0.27	0.05	0.20	0.17
$\|V\| = 32000$	Character	1.39	0.79	0.97	1.00	0.80	0.99	0.26	0.19	0.15	0.17	0.21	0.39
	Byte	0.61	1.03	0.49	0.60	1.07	0.69	0.26	0.21	0.25	0.29	0.28	0.24
	Myte	1.06	1.08	1.30	0.73	0.37	0.66	0.46	0.29	0.18	0.07	0.34	0.28
	Radical	-	-	-	-	0.37	0.66	-	-	-	-	0.34	0.28
	Stroke	-	-	-	-	0.37	1.09	-	-	-	-	0.22	0.36
	$N = 2$												
	Random	0.82	0.68	0.86	1.05	1.13	1.17	0.48	0.22	0.45	0.37	0.24	0.29
	Ours	1.04	0.76	1.06	1.43	0.71	1.23	0.36	0.44	0.15	0.27	0.25	0.22
	$N = 3$												
	Random	0.96	0.59	0.70	0.82	1.14	0.94	0.20	0.09	0.22	0.23	0.37	0.14
	Ours	0.34	1.12	0.83	0.94	0.81	1.13	0.34	0.20	0.20	0.16	0.26	0.55

Table 17: Standard deviation of the text classification results reported in Table 15.

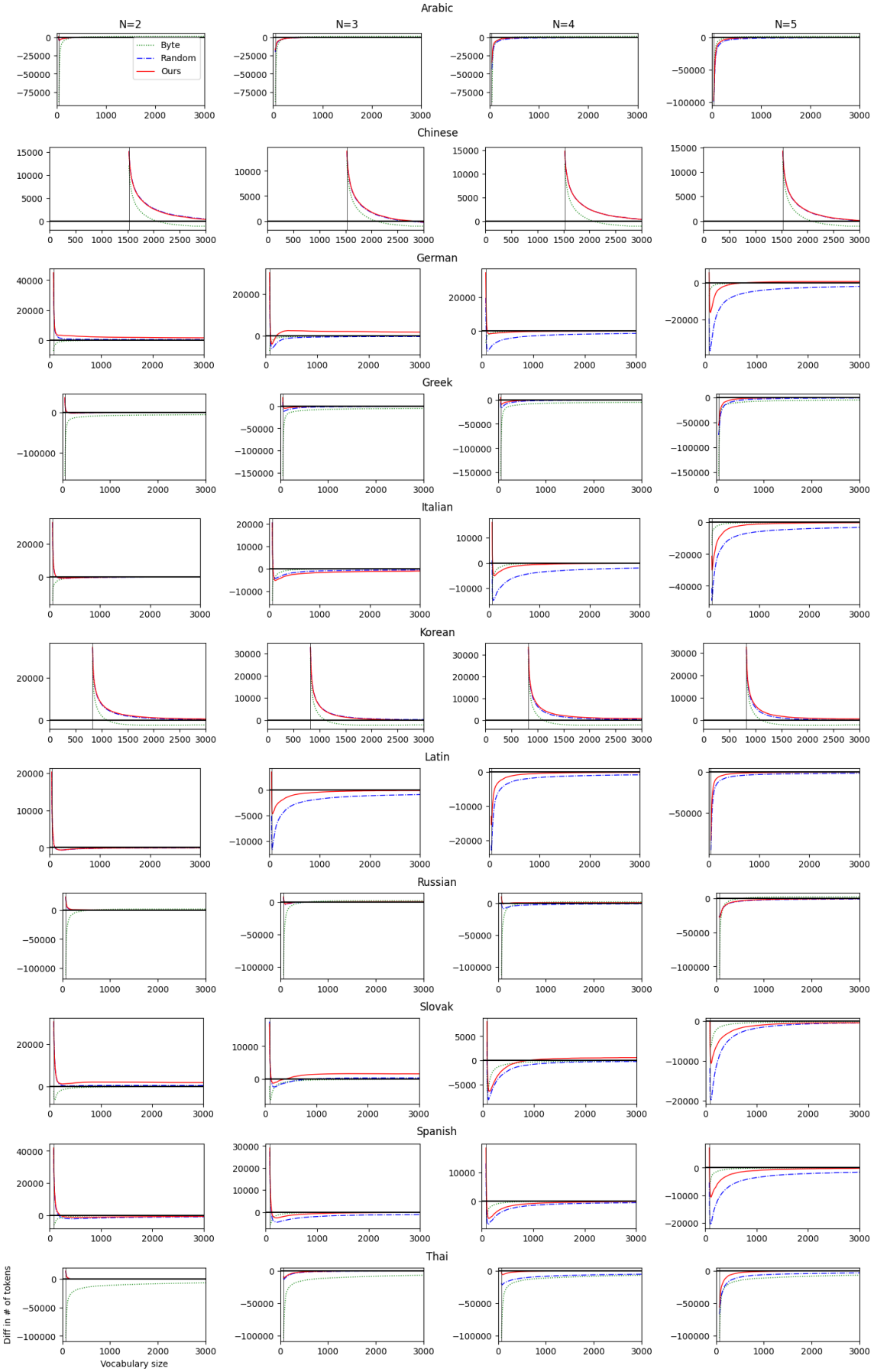

Figure 9: Token-count difference across multiple languages with the BPE tokenizer. **Higher is better**; values above zero indicate fewer tokens than the character-level baseline.

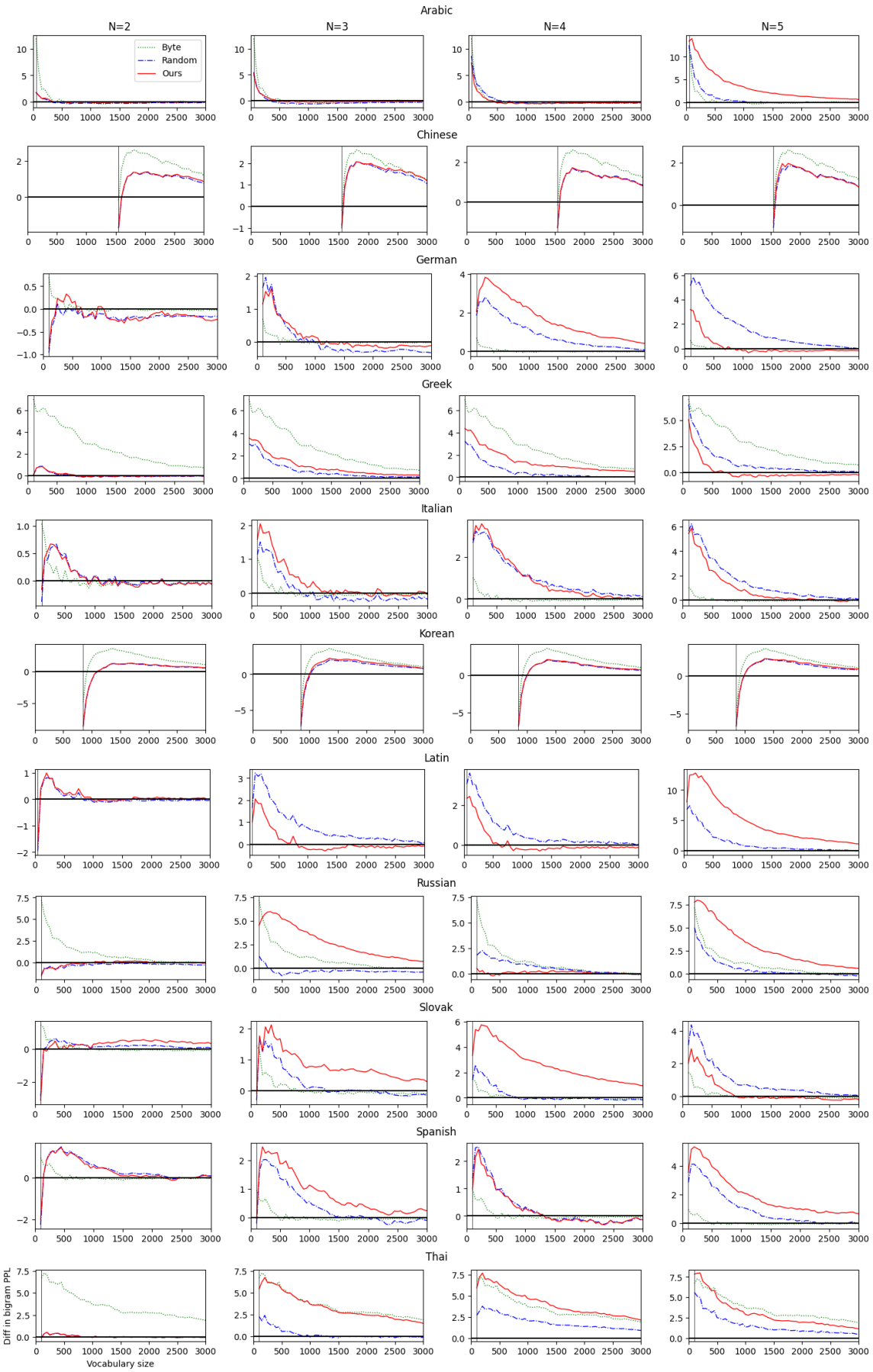

Figure 10: Bigram-perplexity difference across multiple languages with the BPE tokenizer. **Higher is better**; values above zero indicate lower perplexity than the character-level baseline.

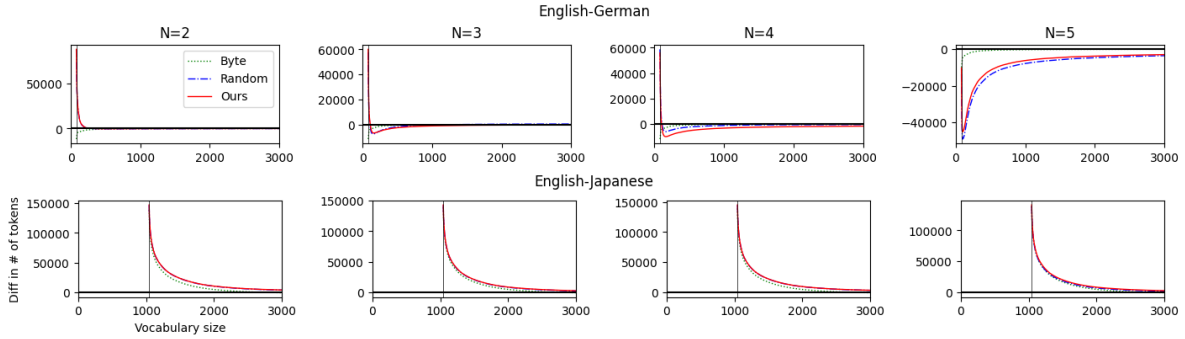

Figure 11: Token-count difference in multilingual setups. **Higher is better**; values above zero indicate fewer tokens than the character-level baseline.

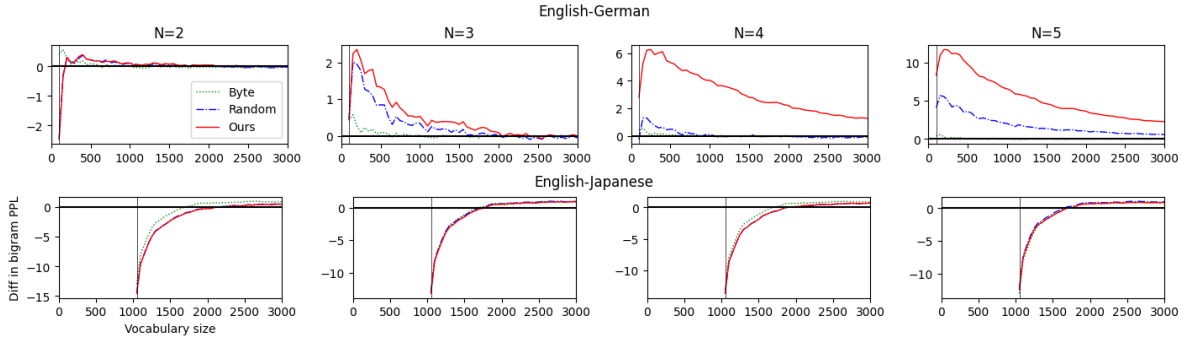

Figure 12: Bigram-perplexity difference in multilingual setups. **Higher is better**; values above zero indicate lower perplexity than the character-level baseline.

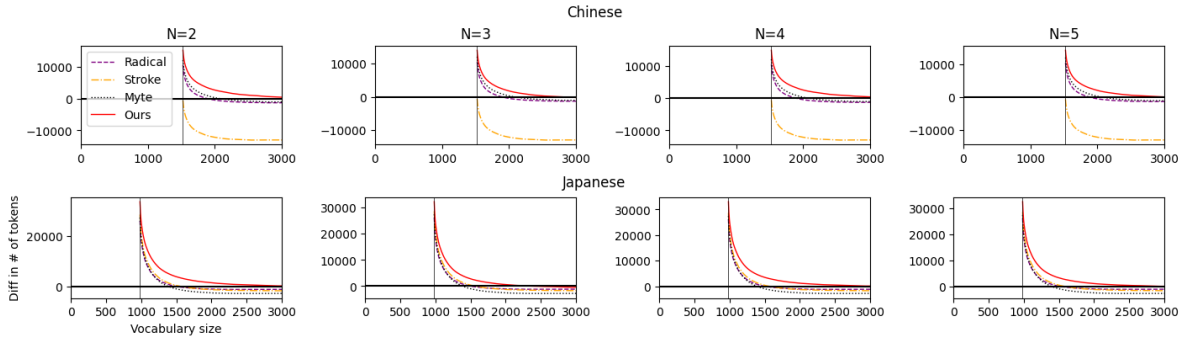

Figure 13: Token-count difference across three additional encoding methods, radical, stroke, and MYTE, in Japanese and Chinese. **Higher is better**; values above zero indicate fewer tokens than the character-level baseline.

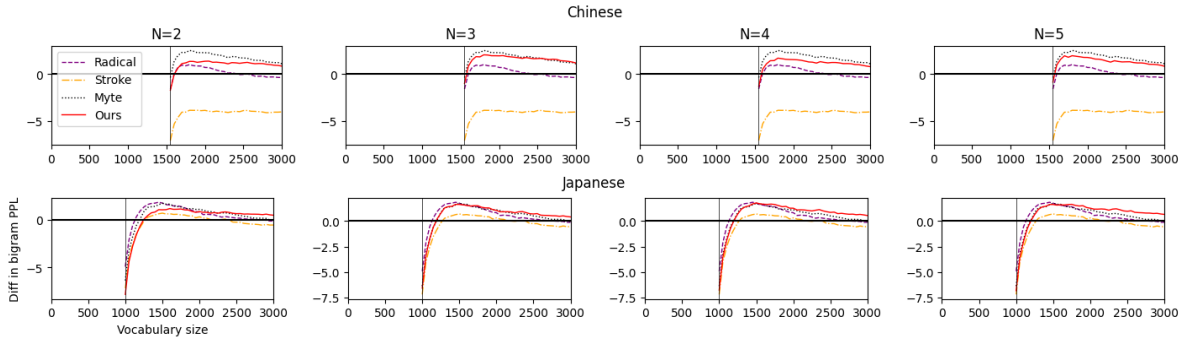

Figure 14: Bigram-perplexity difference across three additional encoding methods, radical, stroke, and MYTE, in Japanese and Chinese. **Higher is better**; values above zero indicate lower perplexity than the character-level baseline.

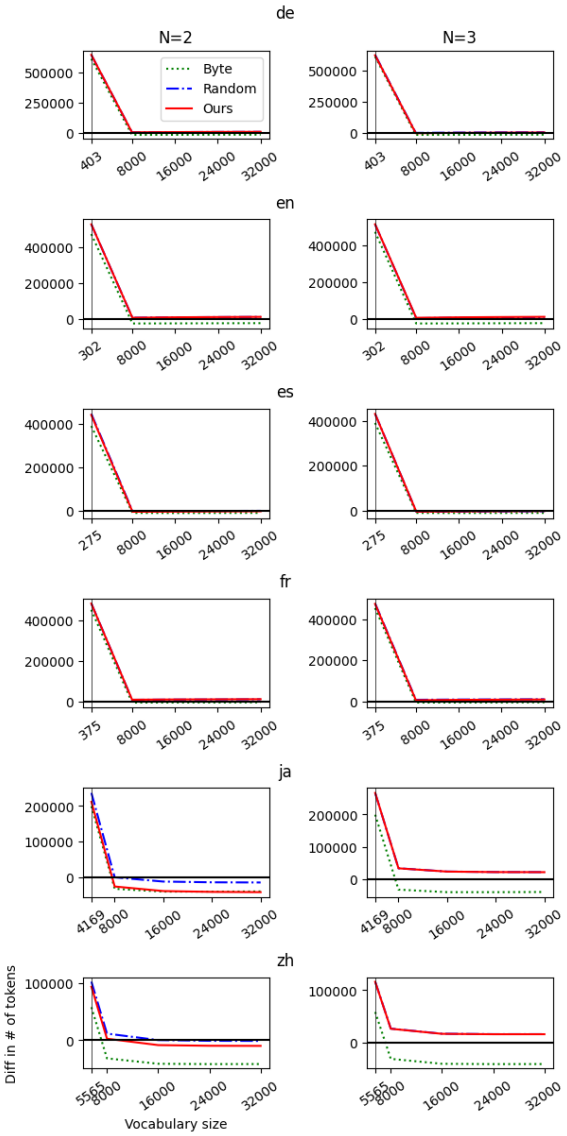

Figure 15: Token-count difference across six languages on the validation split of the Amazon Reviews dataset. **Higher is better**; values above zero indicate fewer tokens than the character-level baseline.

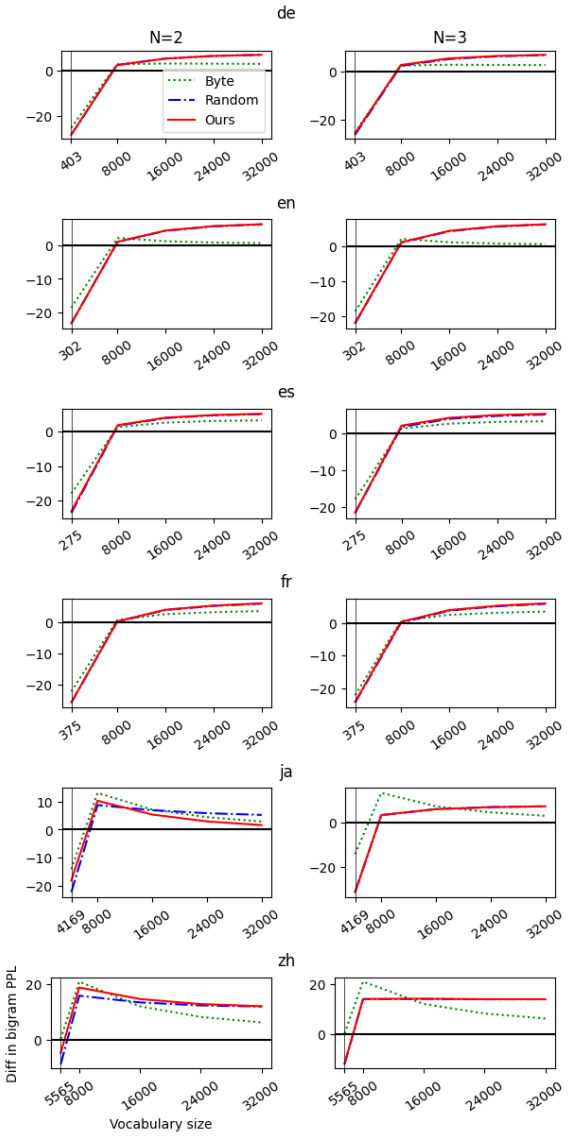

Figure 16: Bigram-perplexity difference across six languages on the validation split of the Amazon Reviews dataset. **Higher is better**; values above zero indicate lower perplexity than the character-level baseline.