Jamo-Level Subword Tokenization in Low-Resource Korean Machine Translation

Junyoung Lee[*,†]
Nanyang Technological University
junyounglee.k@gmail.com

Marco Cognetta[*]
Institute of Science Tokyo
cognetta.marco@gmail.com

Sangwhan Moon
Institute of Science Tokyo
sangwhan@iki.fi

Naoaki Okazaki
Institute of Science Tokyo
okazaki@c.titech.ac.jp

Abstract

Subword tokenization, where text is represented in an intermediate form between full words and characters, is ubiquitous in modern NLP due to its ability to represent any input sentence with a small vocabulary. However for Korean, where there are 11,172 base characters (*syllables*) in its alphabet, it is difficult to have a vocabulary large enough to succinctly encode text while fitting within parameter-budget constraints. This motivates us to explore an alternative representation for Korean which relies on the decompositional nature of Korean syllables, each of which can be uniquely decomposed into a sequence of two or three subcharacters (*jamo*), of which there are only 68.

Using jamo as the basis for subword tokenization (e.g., byte-pair encoding) leads to shorter tokenized sequences with fewer vocabulary parameters, exposes the model to sub-syllable-level morphological information, and increases the amount of augmentation gained from subword regularization. We evaluate jamo-level subword tokenization on several Korean translation tasks and find that jamo-level subword models consistently outperform syllable- and byte-level models in low-resource and restricted-vocabulary settings[1].

1 Introduction

Modern language models struggle with languages such as Chinese, Japanese, and Korean, where the large base character sets (Hanzi, Kanji/Kana, and Korean syllables, respectively) mean that a huge vocabulary parameter count is required to represent the base character set and the added subword tokens. However, unlike Hanzi and Kanji, the 11,172 modern Korean syllables have a compositional structure where each can be decomposed uniquely into a sequence of subcharacters called *jamo*. Jamo decompositions align to syllable-level sequences in a way that byte-level encoding does not and offer a linguistically-grounded, parameter-efficient mechanism for encoding Korean text. Several works have considered jamo-level subword tokenization, but only on the encoding side (Kim et al., 2021; Park et al., 2020a). To the best of our knowledge, this is the first study on the generation of Korean with jamo-level subword tokenization.

공부합니다

ㄱㅗㅇㅂㅜㅎㅏㅂㄴㅣㄷㅏ

공부합니다

Example 1: Example tokenizations for "공부합니다" (to study). From top to bottom: syllable-level, jamo-level, and the jamo-level tokenization mapped back to the syllable sequence. Note that the jamo-level tokenization is able to cross syllable boundaries, while the syllable-level tokenization cannot.

We hypothesize that jamo-level subword tokenization should improve over syllable-level subword tokenization for three reasons. Compared to syllable-level subword tokenization, jamo-level subword tokenization:

1. produces shorter tokenized sequences at a given vocabulary size
2. exposes sub-syllable morphological information
3. unlocks a larger space of tokenizations for subword regularization

In English↔Korean and Korean↔Jeju-eo (a very-low-resource Koreanic language spoken on Korea's Jeju Island) translation tasks, we find that jamo-level models do not improve upon syllable-level models in high resource settings. However, in low-resource and low-parameter settings (specifically, extremely small vocabulary budgets), jamo-

[1]Our full experimental code can be found at https://github.com/mcognetta/jamo-bpe-loresmt.
[*]Authors contributed equally.
[†]This work was done while the author was a visiting student at Institute of Science Tokyo.

Proceedings of the Eighth Workshop on Technologies for Machine Translation of Low-Resource Languages (LoResMT 2025), pages 66–80
May 3, 2025 ©2025 Association for Computational Linguistics

level subword models greatly improve over both syllable- and byte-level subword models, especially when dropout is used. This supports our hypothesis about the benefits of jamo-level subword models.

2 Byte-Pair Encoding and Subword Regularization

Subword tokenization transforms free-form text into a sequence of tokens drawn from a finite vocabulary. The tokens occupy an intermediate granularity between full words (e.g., `democratic` or 대한민국) and character sequences (e.g., `d␣e␣m␣o␣c␣r␣a␣t␣i␣c` or 대␣한␣민␣국), known as subwords (e.g., `demo␣cratic` or 대한␣민국). A key feature is that subword tokenizers are *open vocabulary*—any possible input text can be represented by them (i.e., they never produce out-of-vocabulary (OOV) tokens). This is realized by including a comprehensive atomic vocabulary in the final subword vocabulary (e.g., the full alphabet for English or the set of all Korean syllables).

Byte-pair Encoding (BPE) is a simple and popular subword tokenization technique first introduced by Gage (1994) and reintroduced by Sennrich et al. (2016) for neural machine translation. To construct the vocabulary, starting from the atomic-character-level representation of the training corpus, the model iteratively finds the most frequent co-occurring pair of tokens, merges them, and adds the merge to the subword vocabulary. This process repeats until the vocabulary reaches the desired size. For inference, starting from the character-level input, among all possible merges, the highest priority merge (the one that was added earliest to the merge vocabulary during training) is executed. This repeats until no possible merges remain.

Subword regularization, where tokenizations are sampled from a probability distribution, is often used during training to augment the dataset and improve model robustness, and has been shown to improve model quality across a wide variety of tasks, especially in low-resource machine translation (Kudo, 2018; Provilkov et al., 2020).

BPE-Dropout (Provilkov et al., 2020) is a variant of BPE's inference algorithm that injects randomness into the tokenization process as a form of subword regularization.

3 Hangul and Jamo Decomposition

Hangul, the modern Korean writing system, is represented in Unicode as a list of 11,172 unique *syllables*. Each syllable can be uniquely and deterministically decomposed into two or three *jamo*: an initial consonant, a vowel, and an (optional) final consonant, of which there are 19, 21, and 28, respectively (Table 1). Likewise, any initial consonant, vowel, final consonant triplet can be mapped directly to a unique syllable. For example, 신 = (ㅅ, ㅣ, ㄴ), where ㅅ is the initial consonant, ㅣ is the vowel, and ㄴ is the final consonant.

Initial consonant, $\mathcal{I}$	ㄱ ㄲ ㄴ ㄷ ㄸ ㄹ ㅁ ㅂ ㅃ ㅅ ㅆ ㅇ ㅈ ㅉ ㅊ ㅋ ㅌ ㅍ ㅎ
Vowel, $\mathcal{V}$	ㅏ ㅐ ㅑ ㅒ ㅓ ㅔ ㅕ ㅖ ㅗ ㅘ ㅙ ㅚ ㅛ ㅜ ㅝ ㅞ ㅟ ㅠ ㅡ ㅢ ㅣ
Final consonant, $\mathcal{F}$	∅(nil) ㄱ ㄲ ㄳ ㄴ ㄵ ㄶ ㄷ ㄹ ㄺ ㄻ ㄼ ㄽ ㄾ ㄿ ㅀ ㅁ ㅂ ㅄ ㅅ ㅆ ㅇ ㅈ ㅊ ㅋ ㅌ ㅍ ㅎ
Compatibility jamo	∅(nil) ㄱ ㄲ ㄳ ㄴ ㄵ ㄶ ㄷ ㄸ ㄹ ㄺ ㄻ ㄼ ㄽ ㄾ ㄿ ㅀ ㅁ ㅂ ㅃ ㅄ ㅅ ㅆ ㅇ ㅈ ㅉ ㅊ ㅋ ㅌ ㅍ ㅎ ㅏ ㅐ ㅑ ㅒ ㅓ ㅔ ㅕ ㅖ ㅗ ㅘ ㅙ ㅚ ㅛ ㅜ ㅝ ㅞ ㅟ ㅠ ㅡ ㅢ ㅣ

Table 1: All positional jamo, separated into their three distinct classes, and the the set of compatibility jamo. Note that there are visual overlaps between some initial and final consonants (e.g., ㄱ), and that compatibility jamo is the minimal set of visually distinct jamo.

To produce a decomposed jamo sequence from a syllable sequence, we concatenate the jamo from each triplet corresponding to the syllables. For a syllable without a final consonant, like 구 = (ㄱ, ㅜ, ∅), we omit the ∅ when writing out the jamo sequence[2] (i.e., 가 → (ㄱ, ㅏ, ∅) → ㄱㅏ). Thus, every syllable maps to a length of two or three jamo sequence. Example 1 demonstrates a syllable-level and jamo-level subword tokenization.

Hangul is represented in three different ways in Unicode: precomposed syllables[3], *positional* jamo[4], and *compatibility* jamo[5]. All jamo (including ∅, which denotes the absence of a final consonant), organized by class, are shown in Table 1, separated into positional jamo (partitioned into initial consonants, vowels, and final consonants) and compatibility jamo.

The syllable block (`U+AC00-U+D7AF`) contains all 11,172 modern syllables, each represented by a single codepoint. A syllable can be decomposed into either of the jamo representations, outlined here and discussed in detail in Appendix B.

[2] This is a design choice. Including the ∅ symbol in the decomposed jamo sequence is also possible and results in equivalent but slightly longer sequences.

[3] Wikipedia: Hangul Syllables

[4] Wikipedia: Hangul Jamo (Unicode Block)

[5] Wikipedia: Hangul (Compatibility Jamo)

3.1 Positional Jamo (`U+1100–U+11FF`)

Positional jamo are designed to perfectly match the compositional structure of jamo. The initial consonants, vowels, and final consonants are disjoint Unicode character sets, and a simple procedure based on modular arithmetic is sufficient to convert a single syllable codepoint to an initial consonant, vowel, and final consonant triplet, and vice-versa.

3.2 Compatibility Jamo (`U+3130–U+318F`)

Unlike positional jamo, compatibility jamo does not provide distinct codepoints for initial consonants and final consonants. Instead, visually ambiguous jamo are merged into one token, so that the final set of compatibility jamo is the set of all visually distinct jamo. Thus, there is a surjective mapping from positional jamo to compatibility jamo, but there are compatibility jamo which cannot be unambiguously mapped back to positional jamo (e.g., ㄱ, which could map to either the initial consonant ᄀ or final consonant ᆨ, depending on the surrounding context). A finite state machine is sufficient to recover the syllable sequence from valid compatibility jamo sequences.

3.3 Jeju-eo Orthography

Jeju-eo (제주어) is a Koreanic language that also uses Hangul as the modern writing system, but with the inclusion of two archaic vowels, 「ㆍ」 (*araea*, 아래아, `U+318D`), and its doubled form 「ᆢ」 (*ssang-araea*, 쌍아래아, `U+11A2`). Syllables in the Hangul Precomposed Syllable range do not contain this vowel, so we use a custom encoding described in Appendix B.3. Compatibility and positional jamo representations work like modern Korean by treating 「ㆍ」 and 「ᆢ」 as regular vowels.

4 Benefits of Jamo-level Tokenization

One may ask, why even consider jamo-level tokenization, as syllable-level tokenization is the canonical unit of text? Furthermore, byte-level BPE can be used to alleviate any space issues, and perhaps, as the vocabulary size grows, jamo-level tokens will just converge to syllable boundaries.

Tokenization Length and Small Vocabularies Especially at smaller vocabulary sizes, jamo-level BPE produces substantially shorter tokenized sequences compared to syllable level models, which is crucial to developing models that fit within some parameter or inference budget. As seen in Figure 1, compatibility and positional jamo produce shorter sequences than syllable-level subword tokenizers across all vocabulary sizes, until they eventually converge. When the vocabulary size is small, jamo-level models far outperform syllable-level models in compression ratio. This is especially true at the lowest end of the scale, where syllable-level models operate at essentially the character level (i.e., the entire vocabulary budget is taken up by the base syllable vocabulary, so no merges can be added). Even with a vocabulary budget of just 500, jamo-level subword models produce 5% shorter sequences than syllable-level models with $|V| = 2100$, the smallest possible in our corpus. At $|V| = 1000$ and 1500, jamo-level models produce nearly 25% and 40% shorter sequences than a $|V| = 2000$ syllable model, respectively.

Having shorter sequence lengths is a common metric for tokenizer performance and is also an important consideration with attention-based language models that scale quadratically with sequence length.

Sub-syllable Morphology Morphemes in Korean are not necessarily constrained to syllable boundaries. For example, the word 하기 (*ha-gi*/doing/gerund form) can be morphologically segmented into 하 (*ha*/to do/root) and 기 (*gi*/-ing/nominalization). On the other hand, 합니다 (*hab-ni-da*/to do/honorific form) is morphologically segmented into 하 (*ha*/to do/root) and -ㅂ니다 (*b-nida*/present tense honorific/ending). Notice that the second morpheme contains an incomplete syllable (the final consonant ㅂ) which is part of the first syllable in 합니다 (similar to Example 1).

When modeling at the syllable level, this and similar jamo-level morphology is impossible to capture, as the tokenization is constrained to syllable boundaries, so either the ㅂ jamo information is lost due to being split off from -니다 or the entire sequence 합니다 is represented as a single subword token, and the underlying morphological information is not able to be shared with other sequences that also include the -ㅂ니다 morpheme. However, with jamo-level BPE, which is not constrained to syllable boundaries, it is possible to capture such morphological patterns.

Increased Subword Regularization Subword regularization improves model robustness by sampling tokenizations during training, which augments the training set and breaks the model's conditioning on an exact, canonical tokenization. A

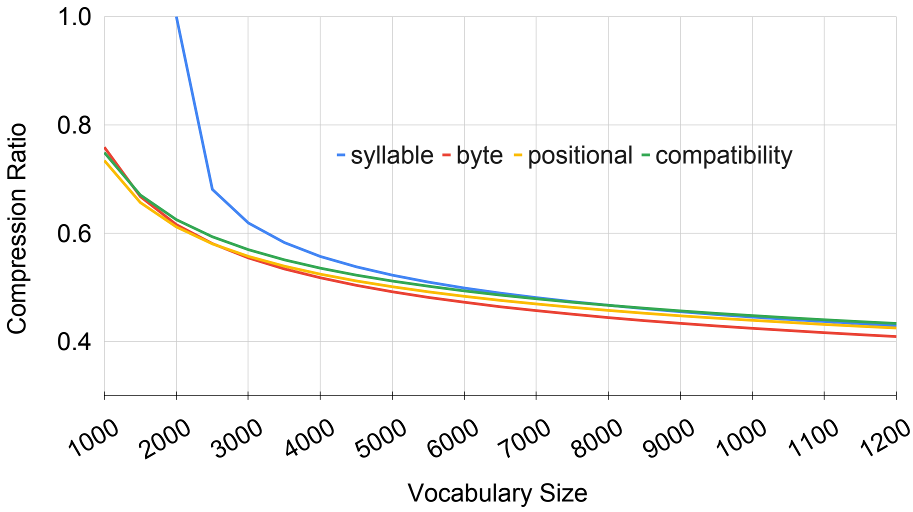

Figure 1: A comparison of the tokenization compression ratio of each of the jamo representations at different vocabulary sizes. The number of tokens in the tokenized corpus is compared to the total number of syllables in the corpus. The syllable subword tokenizer falls back to a character-level tokenizer at $|V| \approx 2000$. However, the jamo and byte subword tokenizers can have much smaller vocabularies and better compression ratios.

key factor is the amount of augmentation (that is, the number of unique tokenized sequences) that are produced during stochastic tokenization, with the implication that a larger number of tokenizations should lead to a larger improvement in model quality and robustness (Cognetta et al., 2024).

Let w_s be a sequence of Korean syllables and w_j the same sequence decomposed into jamo form (the argument works identically for positional and compatibility jamo). For simplicity, assume every subsequence of w_s and w_j is a token in the respective subword vocabularies. Then, the *stars and bars* theorem says that tokenizing a sequence of length n into k tokens can be done in $\binom{n-1}{k-1}$ ways (Wikipedia contributors, 2024) and the total number of ways to tokenize a sequence w is

$$\sum_{k=1}^{|w|} \binom{|w|-1}{k-1} = 2^{|w|-1}.$$

The maximum tokenized sequence lengths for a syllable and jamo sequence are $|w_s|$ and $|w_j|$, respectively, corresponding to just their character-level sequences, which must be a valid tokenization. As $2^{|w_s|-1} \leq 2^{2|w_s|-1} \leq 2^{|w_j|-1} \leq 2^{3|w_s|-1}$, there are at least

$$\frac{2^{2|w_s|-1}}{2^{|w_s|-1}} = 2^{|w_s|}$$

times more ways to tokenize a jamo-level sequence than a syllable-level sequence, meaning there is exponentially larger space of tokenizations available for subword regularization when representing Korean text at the jamo level.

4.1 Arguments Against Jamo Tokenization

Here, we consider some arguments *against* jamo-level tokenization, but argue that they are not fatal.

Why Not Byte-Level BPE? An obvious question is "why not just use byte-level BPE, which should do the same thing?" The primary reason is that there is no clear alignment between bytes, jamo, and syllables in Unicode. For example, a byte-level representation for the token 긱 = (ㄱ, ㅣ, ㄱ) is `<0xEA><0xB8><0xB1>` while the representation for 깋 = (ㄱ, ㅣ, ㅎ) is `<0xEA><0xB9><0x8B>`, which differ in the second and third bytes despite the two having the same initial consonant and vowel. Thus, byte-level representations cannot losslessly capture jamo-level morphologies. Explicitly modeling with jamo provides a linguistically motivated way to represent Korean text which can be combined with a byte-level fallback in the presence of other languages.

Invalid Sequence Generation When operating at the jamo level, the jamo sequences produced by concatenating jamo-level BPE tokens may not follow the canonical `IVF` order. In this case, they cannot be recomposed back into syllable sequences, and the overall sequence becomes invalid (Moon and Okazaki, 2020). For example, if the model produced the token sequence [ㄱ ㅏ] [ㅓ ㅎ], one cannot recompose it to a syllable sequence, since there are two vowels in a row.

In our experiments (Section 5), we observed no instances of invalid sequence generation once the models had gone through a small number of training epochs (and far before they fully converged). Some probability mass is still allocated to invalid sequences, but it can be eliminated by masking out the logits corresponding to invalid tokens (i.e., tokens which would cause an invalid sequence when appended to the current text).

This is not a problem with syllable-level BPE, as it operates only on valid syllables so any sequence of syllables is, at least syntactically, valid, but does affect byte-level tokenization, as a sequence of bytes does not necessarily correspond to a valid Unicode sequence.

Convergence to Syllable-Level Tokens As the vocabulary size grows, it is possible that jamo-level tokenizations eventually converge to just syllable-level tokens — i.e., after a certain point, the vast majority of new merges form unambiguous, full-syllable sequences. Then, if we have to form intermediate syllable-level tokens anyway, it may be better just to start with a syllable base vocabulary.

However, as shown in Figure 1, starting at the syllable level is less space efficient (from a parameter count and tokenized-sequence-length perspective) than jamo-level encoding, and results in less available augmentation from subword regularization. Further, even if they both converge to syllable-level tokens, they likely will not converge to the same set of tokens, and the jamo-level tokenizers will still have a larger number of merged tokens (compared to atomic characters) in the vocabulary than a syllable-level subword tokenizer.

Ambiguity Between Decomposed Syllables and Jamo Literals It is possible that the input or output of a model should be literal jamo, which is difficult to disambiguate from jamo tokens that are produced from decomposition. This is especially true for compatibility jamo representations, as isolated compatibility jamo are often used in colloquial Korean (e.g., ㅋㅋㅋ/"kekeke" for "laughing").

The ambiguity is lessened for positional jamo, as it is not possible to directly input positional jamo on most modern IMEs. Additionally, since compatibility jamo is a distinct set from positional jamo, the set of compatibility jamo can be included in a positional jamo tokenizer's base vocabulary to allow such colloquialisms to be processed directly.

5 Experiments

We compare syllable-level, byte-level, and jamo-level BPE tokenization on two translation tasks: English↔Korean using the AI-Hub News Translation corpus[6] and Korean↔Jeju-eo (Park et al., 2020a) (Section 6.3). The full English↔Korean dataset contains 800k sentence pairs, but we use a 200k sentence-pair subset given our focus on lower-resource settings (Section 6.1). However, experimental results for the full corpus are given in Appendix C and an analysis of a restricted-vocabulary experimental setting on the larger corpus is given in Section 6.2.1. The Korean↔Jeju-eo corpus contains 180k sentence pairs.

We use `SentencePiece` (Kudo and Richardson, 2018) as the BPE tokenizer implementation and `fairseq` (Ott et al., 2019) for training our language models. For each experiment, we fix an underlying Transformer architecture and only vary the tokenizer according to the Korean representation and vocabulary sizes. The source and target side tokenizers are trained separately and their parameters are not shared. All other training configurations are held equal in all experiments. The complete model and training information is given in Appendix A. For each task, we compare the BLEU, CHRF (both via SACREBLEU (Post, 2018)), and COMET (Rei et al., 2020) scores of each model (using an average of 3 runs). All metrics are computed at the syllable level (after the jamo-level model outputs are recomposed to syllables).

In the English↔Korean corpus, there are ∼4600 unique characters. However, about ∼2500 of these are Chinese, Japanese, and Korean (CJK) ideographs (Hanzi/Kanji/Hanja), which appear in only a small fraction of sentences and with median frequency 1, but take up an outsized amount of the vocabulary if we include all observed symbols to avoid OOV. To isolate our main focus (the representation of Korean text), we remove all sentences with these ideographs.

We analyze three axes: 1) input representation, 2) subword regularization, and 3) vocabulary size, and corpus size via the following experiments:

[6] `https://www.aihub.or.kr/`

- English↔Korean
  1) Syllable, Positional, Compatibility[7], Byte
  2) No dropout, English-only dropout, Korean-only dropout, Both dropout
  3) Full 8k (En)/8k (Kr) vocabulary, Restricted 8k (En)/2.1k (Kr) vocabulary
- Korean↔Jeju-eo
  1) Syllable, Positional, Compatibility[7], Byte
  2) No dropout, Korean-only dropout, Jeju-eo-only dropout, Both dropout
  3) Full 4k (Kr)/4k (Je) vocabulary, Restricted 2k (Kr)/2k Je vocabulary

For vocabulary size, the "full" vocabulary size was chosen arbitrarily without a hyperparameter sweep for a fair comparison. However, the "restricted" vocabulary size was chosen as the number of unique characters in the syllable-level corpus. In this setting, the syllable-level models act as character-level models (since there is no room in the vocabulary for merged tokens), while the other representations can still form additional subwords.

6 Results

6.1 English↔Korean

On the left of Table 2 are the English→Korean results. We are particularly interested in this direction as it requires *outputting* Korean, which, as described in Section 4.1, could be difficult when using jamo-level BPE. However, we see that positional jamo-level model performs the best in all three metrics. In the double-dropout case, positional jamo outperforms the best syllable-level model by 0.3 BLEU. Conversely, byte-level BPE performs the worst across the board, indicating that generating Korean with byte-level subwords is difficult.

In the restricted vocabulary setting, positional jamo again performs the best (this time when dropout is applied to the English side only). In the same setting, byte-level BPE comes close, but otherwise generally underperforms, similar to the full vocabulary setting. Syllable-level models also perform much worse than positional jamo-level models (-0.45 change in COMET), which is likely because they are character-level models and the output sequence length becomes too long.

[7]Compatibility jamo performed similarly to positional jamo, so we only report it in Appendix C for space reasons.

On the right side are the Korean→English results. In both the non-restricted and restricted vocabulary settings, dropout does not consistently improve the model. Particularly, in the full vocabulary setting, applying dropout to the English side actually degrades the model performance across all representation types. Syllable-level models also perform the best across the board, and we attribute this to it being easier to *encode* than to *decode* at the character level. Since the models are outputting English and use an 8k English vocabulary, they are able generate coherent text even when the source side uses character-level Korean representations.

6.2 Aside: Full English↔Korean Corpus

Most of the full English↔Korean corpus experiments had uninteresting results, so they are moved to Appendix C, but we cover one interesting experimental result in Section 6.2.1.

In general, in the full corpus setting, we observed no advantage to using jamo- or byte-level modeling over syllable-level BPE. In both directions, the difference in performance of the best positional jamo-level model and the best syllable-level model was within 0.3 BLEU, with the best byte-level models being only slightly worse. Adding dropout to any of the models did not noticeably improve model performance, and sometimes *degraded* quality, in line with past research about subword regularization in high-resource settings (Provilkov et al., 2020).

6.2.1 English→Korean Full Corpus, Restricted Vocabulary

We now turn to an interesting experimental result from the full English↔Korean corpus: the En→Kr restricted-vocabulary setting (Table 3).

Given that syllable-level models are acting essentially as character-level models at this setting, we expect the byte- and positional jamo-level models to perform better than them. We see this to be true as the syllable-level models lose nearly 4 BLEU, showing how difficult it is for the model to learn robust embeddings for such a small vocabulary as the syllables appear in such diverse contexts. Since this is a large corpus, we also expect dropout to have little effect on performance of byte- and positional jamo-level models, and we see that they have similar BLEU, CHRF, and COMET scores.

6.3 Korean↔Jeju-eo

This experiment, shown in Table 4, is a low-resource translation task between two highly re-

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable	None	15.50	38.43	87.47
	English	15.99	38.87	87.86
	Korean	16.42	39.63	88.14
	Both	16.31	39.53	88.18
Byte	None	15.19	38.23	87.32
	English	15.77	38.87	87.89
	Korean	15.50	38.60	87.33
	Both	15.24	38.20	87.07
Positional	None	15.53	38.33	87.36
	English	15.74	38.60	87.63
	Korean	16.10	39.23	87.91
	Both	**16.63**	**39.77**	**88.28**
Syllable (restricted)	None	15.78	38.80	87.37
	English	16.05	38.20	87.68
Byte (restricted)	None	15.82	38.80	87.50
	English	16.47	39.67	88.07
	Korean	15.44	38.30	87.05
	Both	15.59	38.47	87.12
Positional (restricted)	None	15.91	39.00	87.54
	English	**16.52**	**39.77**	**88.13**
	Korean	16.20	39.37	87.84
	Both	16.43	39.57	**88.13**

(a) English→Korean

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable	None	32.30	59.83	80.49
	Korean	**33.36**	**60.73**	**81.02**
	English	32.13	59.27	80.23
	Both	32.74	59.73	80.67
Byte	None	32.21	59.80	80.37
	English	32.31	59.83	80.46
	Korean	31.56	58.73	79.83
	Both	31.31	58.23	79.53
Positional	None	32.25	59.87	80.34
	Korean	33.01	60.40	81.01
	English	31.67	58.73	79.87
	Both	32.74	59.70	80.70
Syllable (restricted)	None	32.93	60.40	80.78
	English	**33.31**	**60.60**	**80.97**
Byte (restricted)	None	32.81	60.17	80.63
	Korean	32.32	59.93	80.43
	English	32.99	60.27	80.72
	Both	32.58	60.00	80.53
Positional (restricted)	None	32.84	60.37	80.71
	Korean	32.84	60.23	80.81
	English	33.08	60.30	80.74
	Both	32.71	59.93	80.49

(b) Korean→English

Table 2: Experimental results[7] for the truncated English↔Korean corpus (Section 6.1). Models marked "(restricted)" are for the restricted Korean vocabulary setting.

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable (restricted)	None	16.75	40.17	86.63
	English	16.78	40.27	86.63
Byte (restricted)	None	20.61	44.65	90.36
	English	20.59	44.50	90.31
	Korean	20.16	44.15	90.03
	Both	19.46	43.13	89.65
Positional (restricted)	None	20.65	44.63	90.33
	English	**20.77**	**44.67**	**90.37**
	Korean	20.50	44.43	90.28
	Both	20.14	44.23	90.19

English→Korean

Table 3: Experimental results[7] for the restricted vocabulary full English→Korean corpus (Section 6.2.1).

lated languages. In both language directions, positional jamo-level BPE far outperforms syllable- and byte-level BPE in both BLEU and CHRF. In the most extreme case, for Korean→Jeju-eo, the positional jamo-level model with dropout on both sides outperforms the best syllable-level and byte-level models by roughly 0.6 and 1.0 BLEU, respectively.

The same is seen in the restricted vocabulary setting, where again positional jamo-level model with dropout on both sides is the best performing model. In Jeju-eo→Korean, the syllable-level model slightly outperforms the positional jamo-level model without dropout. However, the syllable-level model is not able to utilize dropout (since it uses character-level vocabulary), while applying dropout on both sides on the positional jamo-level model improves the model by more than 2.3 BLEU. The Korean→Jeju-eo restricted vocabulary setting is similar. However, here, the no-dropout syllable-level model performs worse than both the byte- and positional jamo-level base models without dropout. When dropout is applied, positional jamo-level model outperforms the best byte-level model by 0.4 BLEU and the syllable-level model by nearly 3 BLEU. We observe similar results for CHRF, where positional jamo outperforms syllable- and byte-level models.

7 Analysis

Overall, we observe that using jamo-level subword tokenization performed on-par with or better than syllable-level tokenization in non-dropout settings and with large vocabularies. This is to be expected, especially at large vocabulary sizes where the jamo-level tokenizers converge to syllable-level tokens.

One hypothesized benefit of using jamo-level subwords is that the increased amount of subword regularization would lead to better modeling. This was most clearly observed in the lowest-resource setting (Korean↔Jeju-eo), but we also observed it in the English→Korean tasks. In the highest resource setting, dropout did not improve modeling quality at all, but this is in line with other research (Provilkov et al., 2020). For Korean↔Jeju-eo, not only did positional jamo-level models with dropout score better than all other models (with or without

Representation	Dropout ($p = 0.1$)	BLEU	CHRF
Syllable	None	69.52	79.67
	Jeju-eo	70.76	80.60
	Korean	70.40	80.37
	Both	71.58	81.30
Byte	None	68.02	78.30
	Jeju-eo	70.21	80.10
	Korean	69.69	79.65
	Both	71.62	81.30
Positional	None	68.97	79.03
	Jeju-eo	71.17	80.87
	Korean	70.27	80.13
	Both	**72.06**	**81.67**
Syllable (restricted)	None	69.67	79.86
Byte (restricted)	None	68.81	78.93
	Jeju-eo	70.51	80.33
	Korean	69.74	79.67
	Both	71.44	81.10
Positional (restricted)	None	69.50	79.60
	Jeju-eo	70.85	80.70
	Korean	70.31	80.23
	Both	**71.82**	**81.56**

(a) Jeju-eo→Korean

Representation	Dropout ($p = 0.1$)	BLEU	CHRF
Syllable	None	43.27	56.37
	Korean	44.54	57.40
	Jeju-eo	44.06	56.97
	Both	45.33	58.13
Byte	None	42.70	55.57
	Jeju-eo	44.17	57.13
	Korean	43.81	56.53
	Both	44.95	57.73
Positional	None	43.49	56.47
	Korean	44.36	57.10
	Jeju-eo	44.27	57.07
	Both	**45.96**	**58.50**
Syllable (restricted)	None	42.83	56.03
Byte (restricted)	None	43.70	56.50
	Jeju-eo	44.46	57.20
	Korean	43.77	56.57
	Both	45.32	58.00
Positional (restricted)	None	43.91	56.60
	Jeju-eo	44.88	57.63
	Korean	44.70	57.30
	Both	**45.72**	**58.33**

(b) Korean→Jeju-eo

Table 4: Experimental results[7] for the Korean↔Jeju-eo corpus (Section 6.3). Models marked "(restricted)" are for the restricted vocabulary setting. Jeju-eo is not supported by COMET, so that metric is omitted.

dropout), but also the gain in performance over the non-dropout baseline was larger, suggesting that the additional tokenizations available for subword regularization is truly beneficial.

A qualitative analysis found that most tokens in the jamo-level vocabularies essentially syllable-level tokens and that these tokens made up the vast majority of actually-observed tokens in the tokenized corpora. This suggests that the additional morphological information available in jamo-level subwords may not be very useful, or it may also just be an artifact of the BPE tokenization algorithm, which merges tokens greedily, and another tokenization algorithm like UnigramLM (Kudo and Richardson, 2018) might make better use of tokens that do not fit in syllable boundaries. Further, in all of the small vocabulary settings with Korean as the output, positional jamo-level models performed the best. This demonstrates that positional jamo representations form more useful tokens than byte-level models and that the ability to form subword tokens in a way that syllable models cannot is beneficial.

Across the board, byte-level models underperformed compared to jamo-level models, despite having similar advantages over syllable-level models. Indeed, in *every* experimental setting, the jamo-level model outperformed the equivalent byte-level model (irrespective of corpus, vocabulary size, or dropout). This suggests that the salient difference is that byte-level models fail to preserve sub-syllable morphological information that is captured by jamo-level models and leads to better modeling.

8 Related Work

Park et al. (2020b) investigate Korean tokenization in various natural language understanding tasks. The strategies compared include jamo and syllable character-level modeling, morphological segmentation provided by MeCab-ko (Kudo, 2006), syllable-level BPE, and word-level segmentation.

In another work, Park et al. (2020a) consider jamo-level byte-pair encoding for Jeju-eo in a similar way to what we explore here. However, this is in the context of text-to-speech, and the jamo-level subword encoding is only applied to the *input* side but not to generation. For a translation task, Park and Zhao (2020) used hierarchical syllable and jamo-level features, but also only for encoding. Jamo-level modeling has been applied to many encoder-only tasks such as named entity recognition (Stratos, 2017; Kim et al., 2021) and sentence classification (Cho et al., 2019). For decoding, both Song et al. (2018) and Cognetta et al. (2023) used jamo-level representations in character-level Korean language-modeling tasks. However, their approaches do not apply to subword-level modeling.

For Chinese and Japanese, which have large base vocabularies due to their use of ideographs, radical-based decomposition has been explored as a possible sub-character method to reduce the required vocabulary budget and improve modeling (Shi et al., 2015; Nguyen et al., 2017; Saunders et al., 2020; Si et al., 2023). However, unlike Korean, radical-based decomposition is not lossless.

9 Conclusion

We investigate jamo-level subword tokenization for Korean machine translation based on three theoretical benefits—shorter tokenized sequences and better vocabulary allocation, exposure to sub-syllable morphological information, and larger space of tokenizations for subword regularization—and show that in two translation tasks and across multiple experimental settings, jamo-level models outperform syllable-level models and byte-level models.

Our experimental results support the hypothesized advantages of jamo-level subword modeling in that: 1) in small-vocabulary settings, jamo-level models far outperform syllable-level models (which essentially act like character-level models), 2) jamo-level models outperform byte-level models across the board (with the primary difference between them being that jamo models preserve sub-syllable information that byte-level models do not and syllable-level models cannot), and 3) with the same dropout hyperparameters, jamo-level models both perform better than analogous syllable- and byte-level models *and also* exhibit a larger increase in performance over the non-dropout models as compared to syllable- and byte-level models.

We conclude with the recommendation against simply defaulting to syllable-level or byte-level subword tokenization for Korean NLP, as it can lead to poor tokenizations and loss of performance, especially in low-resource settings. We hope to have provided sufficient basis for further exploration into jamo-level subword tokenization with our work.

Limitations

One limitation is that we used only two datasets, and that the Korean↔Jeju-eo pair is an extremely closely related language pair. It would be better to compare with other language pairs, especially others from the CJK family. However, few high quality datasets with Korean parallel sentences exist, so this was not possible.

Another is that we did not do a large vocabulary hyperparameter sweep. While the restricted vocabulary settings are inherently fixed (since we choose the vocabulary size to be the number of unique characters in the corpus so that the syllable-level models are forced to be as small as possible), the other vocabulary sizes were picked arbitrarily (though the same vocabulary sizes were used for all models within a language pair). It is possible that our results would change with different vocabulary sizes. However, finding the best vocabulary size is prohibitively expensive, and by arbitrarily choosing a size and using it across all models, we hoped to provide a fair comparison.

We also did not experiment with using large, pretrained models as our base encoders and decoders, which has become common in modern NLP. The primary reason for this is that pretrained models come with their own tokenizers. In order to experiment with a variety of tokenizers like we did in this paper, we would need to train our own large base models from scratch for each tokenizer, which is prohibitively expensive and beyond the scope of this paper.

A final limitation is that our corpora had at most 200k sentence pairs, which should be considered low-resource. However, the full English↔Korean had 800k sentence pairs, which is substantially higher resource than the Korean↔Jeju-eo corpus, but might be considered low-resource in the modern, data-rich NLP era. Our results do not scale perfectly to the larger dataset, suggesting that as the dataset grows, the benefits of our proposed technique diminish. However, this is not unexpected, as many of these techniques have diminishing returns as the vocabulary size grows (in particular, dropout is not effective and can even be harmful in large corpus settings (Provilkov et al., 2020)).

References

Won Ik Cho, Seok Min Kim, and Nam Soo Kim. 2019. Investigating an effective character-level embedding in Korean sentence classification. In *Proceedings of the 33rd Pacific Asia Conference on Language, Information and Computation*, pages 10–18, Hakodate, Japan. Waseda Institute for the Study of Language and Information, Waseda University, Tokyo, Japan.

Marco Cognetta, Sangwhan Moon, Lawrence Wolf-sonkin, and Naoaki Okazaki. 2023. Parameter-efficient Korean character-level language modeling. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational*

Linguistics, pages 2350–2356, Dubrovnik, Croatia. Association for Computational Linguistics.

Marco Cognetta, Vilém Zouhar, and Naoaki Okazaki. 2024. Distributional properties of subword regularization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10753–10763, Miami, Florida, USA. Association for Computational Linguistics.

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

Gyeongmin Kim, Junyoung Son, Jinsung Kim, Hyunhee Lee, and Heuiseok Lim. 2021. Enhancing Korean named entity recognition with linguistic tokenization strategies. *IEEE Access*, 9:151814–151823.

Taku Kudo. 2006. MeCab: Yet Another Part-of-Speech and Morphological Analyzer.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

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.

Sangwhan Moon and Naoaki Okazaki. 2020. Jamo pair encoding: Subcharacter representation-based extreme Korean vocabulary compression for efficient subword tokenization. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3490–3497, Marseille, France. European Language Resources Association.

Viet Nguyen, Julian Brooke, and Timothy Baldwin. 2017. Sub-character neural language modelling in Japanese. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 148–153, Copenhagen, Denmark. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Jeonghyeok Park and Hai Zhao. 2020. Korean neural machine translation using hierarchical word structure. In *2020 International Conference on Asian Language Processing (IALP)*, pages 294–298.

Kyubyong Park, Yo Joong Choe, and Jiyeon Ham. 2020a. Jejueo datasets for machine translation and speech synthesis. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2615–2621, Marseille, France. European Language Resources Association.

Kyubyong Park, Joohong Lee, Seongbo Jang, and Dawoon Jung. 2020b. An empirical study of tokenization strategies for various Korean NLP tasks. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 133–142, Suzhou, China. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Danielle Saunders, Weston Feely, and Bill Byrne. 2020. Inference-only sub-character decomposition improves translation of unseen logographic characters. In *Proceedings of the 7th Workshop on Asian Translation*, pages 170–177, Suzhou, China. Association for Computational Linguistics.

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, Berlin, Germany. Association for Computational Linguistics.

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.

Chenglei Si, Zhengyan Zhang, Yingfa Chen, Fanchao Qi, Xiaozhi Wang, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. 2023. Sub-character tokenization for Chinese pretrained language models.

Transactions of the Association for Computational Linguistics, 11:469–487.

Chisung Song, Myungsoo Han, Hoon Young Cho, and Kyong-Nim Lee. 2018. Sequence-to-sequence autoencoder based Korean text error correction using syllable-level multi-hot vector representation. In *Proceedings of HCLT (in Korean)*, pages 661–664.

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.

Wikipedia contributors. 2024. Stars and bars (combinatorics) — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Stars_and_bars_(combinatorics)&oldid=1240643785`. [Online; accessed 26-August-2024].

A Architecture and Training Details

We used `fairseq` for the training. For English↔Korean, we used the base `transformer` architecture. For Korean↔Jeju-eo, we used the smaller `transformer-iwslt` architecture. Table 5 gives the model configurations and Table 6 gives the optimization and training settings.

`transformer`	
Embedding Dimension	512
FFN Dimension	2048
Number of Heads	6
Number of Layers	8
Dropout	0.1

`transformer-iwslt`	
Embedding Dimension	512
FFN Dimension	1024
Number of Heads	4
Number of Layers	6
Dropout	0.1

Table 5: The configurations for the `transformer` and `transformer-iwslt` architectures.

A.1 Tokenization and Segmentation

We use `SentencePiece` for BPE tokenization. Table 8 gives the flags for the Korean and English tokenizers.

We used `SacreBLEU` to compute the metrics. The text was pre-segmented by whitespace and punctuation with `SacreMoses`.

Optimizer	ADAM
β_1, β_2	$(0.9, 0.98)$
Learning Rate	5×10^{-4}
Warmup	4000 steps (Korean↔Jeju-eo) 20000 steps (English↔Korean)
Scheduler	Inverse Square Root
Tokens-per-batch	4096
Patience	5 (English↔Korean) 8 (Korean↔Jeju-eo)

Table 6: The optimizer and training parameters.

Corpus	Train	Test	Valid
English↔Korean	750k	25k	25k
English↔Korean (truncated)	200k	8k	10k
Korean↔Jeju-eo	160k	5k	5k

Table 7: The size of the corpora used for each experiment.

B Jamo Decomposition

For clarity, to disambiguate visually identical initial and final positional jamo, we mark them with a subscript denoting their position. For example, 국 would decompose to ㄱ$_i$, ㅜ$_v$, ㄱ$_f$. For compatibility jamo, we always leave it unmarked (e.g, ㄱ, ㅜ, ㄱ).

B.1 Positional Jamo

Let c be the codepoint of any Hangul syllable, and $c' = c - \texttt{0xAC00}$ be its offset from the start of the Hangul syllable range. Then, we compute i, v, f (the initial consonant, vowel, and final consonant positional jamo codepoint offsets, respectively) as:

$$\begin{aligned} i &= \frac{c'}{588} \\ v &= \frac{c' - (588 \cdot i)}{28} \\ f &= (c' - (588 \cdot i)) - 28 \cdot v \end{aligned} \tag{1}$$

For example, the syllable 름 (U+B984) gives $c' = 3460$, $i = 5$ ($\mathcal{I}_5 =$ ㄹ$_i$) , $v = 18$ ($\mathcal{V}_{18} =$ ㅡ$_v$), and $f = 16$ ($\mathcal{F}_{16} =$ ㅁ$_f$).

A triplet with $f = 0$ signifies that the syllable does not have a final consonant. For example, 르 (U+B974), which does not have a final consonant gives $i = 5 =$ ㄹ$_i$, $v = 18 =$ ㅡ$_v$, but $f = 0 = \varnothing_f$.

Recomposition of positional jamo triplets back to syllable follows the inverse of the same algorithm. Given i, v, and f:

$$c = i \times 588 + v \times 28 + f + 44032$$

Language	Flag
English	`--character_coverage=1.0` `--normalization_rule="identity"`

Korean
&
Jeju-eo
	`--character_coverage=1.0`
	`--normalization_rule="identity"`
	`--split_by_whitespace=false`

Table 8: `SentencePiece` tokenizer settings for each language. All flags not listed here are set to the defaults.

produces the original syllable codepoint. Thus, to convert a positional jamo sequence back to a syllable sequence, we simply iterate through each (i, v, f) triplet and recover the original syllables.

B.2 Compatibility Jamo

Decomposition of syllables to compatibility jamo is a simple two-step process of first decomposing syllables into positional jamo and then converting the resulting positional jamo to the corresponding compatibility jamo with the surjective mapping. For example, 왕 = ᄋ$_i$, ᅪ$_v$, ᆼ$_f$, which is mapped to the compatibility jamo `ㅇ, ㅘ, ㅇ` (in the latter, the ㅇ's are the same codepoint, while in the former, they are distinct codepoints).

A difficulty comes in the recomposition of a compatibility jamo sequence back into syllables, which requires disambiguating the compatibility jamo by converting them back to positional jamo. Without context, since the mapping is surjective, this is not possible. However, since syllables always follow (i, v, f) order, compatibility jamo can be disambiguated by greedily decoding the jamo sequence from left to right via a simple state machine.

B.3 Jeju-eo Syllable Decomposition

To convert from a Unicode Private Use Area (PUA) representation of a syllable containing *araea*, we extract the initial and final like in Equation 1. Let p be the start of the PUA range and c be the codepoint in the PUA range we wish to decompose. Then $i = \lfloor \frac{c-p}{|\mathcal{I}|} \rfloor$ and $f = (c-p) \mod |\mathcal{I}|$. This produces positional jamo for the initial and final consonants, which can be mapped to compatibility jamo as usual.

To reverse the process, given an initial and final (positional) consonant, we compute $i \times \mathcal{I} + f + p$ to recover the PUA-indexed codepoint corresponding to the syllable $(\mathcal{I}_i, \cdot, \mathcal{F}_f)$.

The same is done for *ssang-araea*, but with a separate PUA.

C Full Results

Tables 9, 10, and 11 contain the full experimental results. Specifically, they all contain the Compatibility Jamo experiments (which were omitted due to space from the main paper's tables) and the full-sized English↔Korean corpus results (only a subset of this corpus was presented in Table 3).

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable	None	15.50	38.43	87.47
	English	15.99	38.87	87.86
	Korean	16.42	39.63	88.14
	Both	16.31	39.53	88.18
Byte	None	15.19	38.23	87.32
	English	15.77	38.87	87.89
	Korean	15.50	38.60	87.33
	Both	15.24	38.20	87.07
Compatibility	None	15.32	38.00	87.13
	English	15.74	38.50	87.66
	Korean	16.25	39.50	88.06
	Both	16.23	39.27	87.99
Positional	None	15.53	38.33	87.36
	English	15.74	38.60	87.63
	Korean	16.10	39.23	87.91
	Both	**16.63**	**39.77**	**88.28**
Syllable (restricted)	None	15.78	38.80	87.37
	English	16.05	38.20	87.68
Byte (restricted)	None	15.82	38.80	87.50
	English	16.47	39.67	88.07
	Korean	15.44	38.30	87.05
	Both	15.59	38.47	87.12
Compatibility (restricted)	None	15.79	38.73	87.53
	English	16.18	39.10	87.72
	Korean	16.00	39.10	87.68
	Both	16.21	39.30	87.88
Positional (restricted)	None	15.91	39.00	87.54
	English	**16.52**	**39.77**	**88.13**
	Korean	16.20	39.37	87.84
	Both	16.43	39.57	**88.13**

(a) English→Korean

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable	None	32.30	59.83	80.49
	Korean	**33.36**	**60.73**	**81.02**
	English	32.13	59.27	80.23
	Both	32.74	59.73	80.67
Byte	None	32.21	59.80	80.37
	English	32.31	59.83	80.46
	Korean	31.56	58.73	79.83
	Both	31.31	58.23	79.53
Compatibility	None	32.18	59.77	80.25
	Korean	32.85	60.20	80.81
	English	31.71	58.97	79.98
	Both	31.97	58.90	80.08
Positional	None	32.25	59.87	80.34
	Korean	33.01	60.40	81.01
	English	31.67	58.73	79.87
	Both	32.74	59.70	80.70
Syllable (restricted)	None	32.93	60.40	80.78
	English	**33.31**	**60.60**	**80.97**
Byte (restricted)	None	32.81	60.17	80.63
	Korean	32.32	59.93	80.43
	English	32.99	60.27	80.72
	Both	32.58	60.00	80.53
Compatibility (restricted)	None	32.65	60.07	80.56
	Korean	32.89	60.20	80.76
	English	32.87	60.07	80.69
	Both	32.81	60.07	80.47
Positional (restricted)	None	32.84	60.37	80.71
	Korean	32.84	60.23	80.81
	English	33.08	60.30	80.74
	Both	32.71	59.93	80.49

(b) Korean→English

Table 9: Full experimental results for the truncated English↔Korean corpus (Section 6.1) including Compatibility Jamo, which was omitted in Table 2. Models marked "(restricted)" are for the restricted Korean vocabulary setting.

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable	None	20.27	44.23	90.29
	English	20.22	44.13	90.26
	Korean	**20.52**	**44.67**	**90.44**
	Both	20.23	44.27	90.32
Byte	None	19.96	44.00	90.22
	English	19.88	43.87	90.19
	Korean	19.19	43.10	89.70
	Both	18.22	41.77	89.01
Compatibility	None	20.23	44.13	90.24
	English	19.88	43.70	90.12
	Korean	20.26	43.33	90.31
	Both	20.18	43.43	89.97
Positional	None	20.19	44.13	90.27
	English	19.86	43.77	90.15
	Korean	20.35	44.40	90.33
	Both	19.88	43.70	90.09
Syllable (restricted)	None	16.75	40.17	86.63
	English	16.78	40.27	86.63
Byte (restricted)	None	20.61	44.65	90.36
	English	20.59	44.50	90.31
	Korean	20.16	44.15	90.03
	Both	19.46	43.13	89.65
Compatibility (restricted)	None	20.62	44.53	90.25
	English	20.65	44.53	90.31
	Korean	20.20	44.20	90.18
	Both	19.95	43.83	90.04
Positional (restricted)	None	20.65	44.63	90.33
	English	**20.77**	**44.67**	**90.37**
	Korean	20.50	44.43	90.28
	Both	20.14	44.23	90.19

(a) English→Korean

Representation	Dropout ($p = 0.1$)	BLEU	CHRF	COMET
Syllable	None	37.74	63.93	83.37
	Korean	**38.04**	**64.17**	**83.57**
	English	36.77	62.73	82.78
	Both	36.73	62.73	82.77
Byte	None	37.48	63.73	83.27
	English	37.17	63.57	83.14
	Korean	36.25	62.23	82.52
	Both	35.69	61.73	82.24
Compatibility	None	37.60	63.80	83.27
	Korean	37.68	63.83	83.33
	English	36.38	62.43	82.52
	Both	36.78	62.70	82.77
Positional	None	37.72	63.90	83.37
	Korean	37.55	63.80	83.34
	English	36.46	62.57	82.64
	Both	36.52	62.43	82.66
Syllable (restricted)	None	37.53	63.77	83.32
	English	37.50	63.63	83.29
Byte (restricted)	None	37.82	63.97	83.43
	Korean	37.12	63.57	83.11
	English	37.42	63.57	83.20
	Both	36.72	63.07	82.86
Compatibility (restricted)	None	37.78	63.97	**83.44**
	Korean	37.62	63.83	83.38
	English	37.33	63.47	83.11
	Both	37.20	63.30	83.06
Positional (restricted)	None	**37.87**	**64.03**	83.40
	Korean	37.65	63.87	83.34
	English	37.51	63.60	83.26
	Both	37.20	63.30	83.08

(b) Korean→English

Table 10: Full experimental results for the full English↔Korean corpus (Section 6.1) including Compatibility Jamo, which was omitted in Table 3. Models marked "(restricted)" are for the restricted Korean vocabulary setting.

Representation	Dropout ($p = 0.1$)	BLEU	CHRF
Syllable	None	69.52	79.67
	Jeju-eo	70.76	80.60
	Korean	70.40	80.37
	Both	71.58	81.30
Byte	None	68.02	78.30
	Jeju-eo	70.21	80.10
	Korean	69.69	79.65
	Both	71.62	81.30
Compatibility	None	68.79	78.70
	Jeju-eo	70.56	80.47
	Korean	69.90	79.80
	Both	71.98	81.53
Positional	None	68.97	79.03
	Jeju-eo	71.17	80.87
	Korean	70.27	80.13
	Both	**72.06**	**81.67**
Syllable (restricted)	None	69.67	79.86
Byte (restricted)	None	68.81	78.93
	Jeju-eo	70.51	80.33
	Korean	69.74	79.67
	Both	71.44	81.10
Positional (restricted)	None	69.50	79.60
	Jeju-eo	70.85	80.70
	Korean	70.31	80.23
	Both	**71.82**	**81.56**

(a) Jeju-eo→Korean

Representation	Dropout ($p = 0.1$)	BLEU	CHRF
Syllable	None	43.27	56.37
	Korean	44.54	57.40
	Jeju-eo	44.06	56.97
	Both	45.33	58.13
Byte	None	42.70	55.57
	Jeju-eo	44.17	57.13
	Korean	43.81	56.53
	Both	44.95	57.73
Compatibility	None	43.05	55.87
	Korean	44.11	56.97
	Jeju-eo	43.97	56.73
	Both	45.65	58.30
Positional	None	43.49	56.47
	Korean	44.36	57.10
	Jeju-eo	44.27	57.07
	Both	**45.96**	**58.50**
Syllable (restricted)	None	42.83	56.03
Byte (restricted)	None	43.70	56.50
	Jeju-eo	44.46	57.20
	Korean	43.77	56.57
	Both	45.32	58.00
Positional (restricted)	None	43.91	56.60
	Jeju-eo	44.88	57.63
	Korean	44.70	57.30
	Both	**45.72**	**58.33**

(b) Korean→Jeju-eo

Table 11: Full experimental results for the Korean↔Jeju-eo corpus (Section 6.3) including Compatibility Jamo, which was omitted in Table 4. Models marked "(restricted)" are for the restricted vocabulary setting. Jeju-eo is not supported by COMET, so that metric is omitted.