Alignment-Based Decoding Policy for Low-Latency and Anticipation-Free Neural Japanese Input Method Editors

Armin Sarhangzadeh and **Taro Watanabe**
Nara Institute of Science and Technology
{sarhangzadeh.armin.rw0,taro}@naist.jp

Abstract

Japanese input method editors (IMEs) are essential tools for inputting Japanese text using a limited set of characters such as the kana syllabary. However, despite their importance, the potential of newer attention-based encoder-decoder neural networks, such as Transformer, has not yet been fully explored for IMEs due to their high computational cost and low-quality intermediate output in simultaneous settings, leading to high latencies. In this work, we propose a simple decoding policy to enable the use of attention-based encoder-decoder networks for simultaneous kana-kanji conversion in the context of Japanese IMEs. We demonstrate that simply decoding by explicitly considering the word boundaries achieves a fairly strong quality-latency trade-off, as it can be seen as equivalent to performing decoding on aligned prefixes and thus achieving an incremental anticipation-free conversion. We further show how such a policy can be applied in practice to achieve high-quality conversions with minimal computational overhead. Our experiments show that our approach can achieve a noticeably better quality-latency trade-off compared to the baselines, while also being a more practical approach due to its ability to directly handle streaming input. Our code is available at https://doi.org/10.5281/zenodo.11450159.

1 Introduction

Japanese input method editors (IMEs) allow users to input Japanese text, which may include thousands of Chinese characters called kanji, using an unsegmented sequence of a much more limited set of characters such as the kana syllabary or the Roman alphabet (rōmaji) loosely representing its pronunciations, a process commonly referred to as kana-kanji conversion. As kana-kanji conversion is by far the most common approach for inputting Japanese text, Japanese IMEs have become an integral part of hundreds of millions of people's daily interactions with mobile devices and computers.

Because of their importance, Japanese IMEs have a long history, dating back to 1978[1], with a lot of attempts at improving them throughout the years. However, despite the progress in traditional techniques, the potential of newer neural network based approaches for IMEs has not yet been fully explored, in spite of them achieving remarkable results in various context-dependent natural language processing tasks by exploiting, e.g., attention-based encoder-decoder networks such as Transformer (Vaswani et al., 2017). This is especially important since the limited phonemic inventory and mora-based phonology of Japanese lead to a very large number of homophones (and consequently homographs when written in kana or rōmaji), making context a vital factor.

The above-mentioned gap in the literature can be mainly attributed to two reasons. The first reason, is the noticeably high computational cost of conventional full-sentence encoder-decoder networks when it comes to inference, which makes them unsuitable for providing near real-time output in simultaneous settings. And the second reason, is their potentially low quality intermediate output in simultaneous settings, which can be attributed to train-test mismatch caused by different train and test settings (i.e. offline training vs online testing).

In this work, inspired by simultaneous machine translation (SimulMT), we propose a simple decoding policy to enable the use of attention-based encoder-decoder networks for simultaneous kana-kanji conversion in the context of Japanese IMEs. We demonstrate that simply decoding by explicitly considering the word boundaries achieves a fairly strong quality-latency trade-off, as it can

[1] In which Toshiba, JW-10, the first Japanese language word processor, was introduced.

Findings of the Association for Computational Linguistics ACL 2024, pages 8043–8054

be seen as equivalent to performing decoding on aligned prefixes and thus achieving an incremental anticipation-free conversion by leveraging the monotonic nature of kana-kanji conversion. We further show how such a policy can be applied in practice to achieve high-quality conversions with minimal computational overhead. More specifically, we utilize online word-boundary predictions from an auxiliary linear layer to perform decoding on aligned prefixes, while simultaneously employing an extra auxiliary linear layer incorporated in a wait-k fashion to allow backtracking and corrections in case of mismatches with previous online boundary predictions.

Our experiments on conversion quality, in addition to both computational and non-computational (i.e. the lag between source and target) latencies, show that our approach can achieve a noticeably better quality-latency trade-off compared to the baselines, while also being a more practical approach to IMEs due to its ability to directly handle streaming input.

2 Preliminaries

2.1 Kana-Kanji Conversion

Kana-kanji conversion, the core element of Japanese IMEs, involves converting a hiragana sequence representing the underlying phonetic form, to the corresponding written surface form, consisting of a kana-kanji mixed sequence. The complex and context-dependant relation between kana and kanji, in addition to the lack of delimiters in standard Japanese text and input, makes kana-kanji conversion a non-trivial task. For example, the very simple kana sequence "はる" (read as "ha ru") has more than 4000 potential surface forms based on two different possible segmentations, making educated guesses without considering context fairly challenging.

More formally, given an unsegmented sequence of hiragana characters $\boldsymbol{x} = (x_1, \ldots, x_n)$, we want to find the most likely corresponding kana-kanji mixed tokens $\boldsymbol{y} = (y_1, \ldots, y_m)$, $m \leq n$, where there is a monotonic many-to-one relation between their alignments. That is, given an alignment function $a(s) = t$, which maps the hiragana index s to the kanji index t where $y_{a(s)}$ corresponds to x_s, we have:

$$s_1 < s_2 \implies a(s_1) \leq a(s_2) \tag{1}$$

This monotonic and many-to-one relation is more clearly illustrated by the non-crossing converging connections in Figure 1, which depicts the alignments between the surface form of an example sentence and its underlying kana sequence.

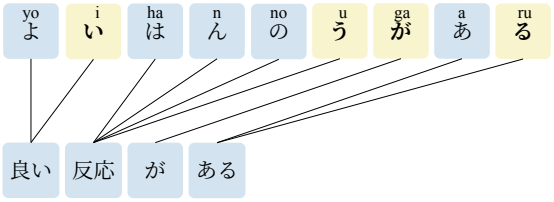


Figure 1: Alignment representation between the surface form of the sentence "There is a good response" and its underlying kana representation, with yellow/bold characters representing word-boundaries. Note that here the kana sequence is tokenized as characters while the kana-kanji mixed tokens represent subwords.

Considering the similarities between the two tasks, we proceed to formulate kana-kanji conversion as a translation from a kana source sequence to the corresponding kana-kanji mixed target sequence, similar to Huang et al. (2018). This interpretation is especially useful as NMT can be seen as the prototypical example of applying encoder-decoder networks to sequence transduction.

More specifically, given the simultaneous nature of kana-kanji conversion in IMEs, we will treat it as a case of simultaneous NMT within the prefix-to-prefix framework (Ma et al., 2019).

2.2 Full-Sentence NMT

Conventional full-sentence NMT involves first processing the input sequence $\boldsymbol{x} = (x_1, \ldots, x_n)$ using an encoder before passing it to a decoder, which then greedily predicts the next target token based on both the input sequence and the current partial hypothesis $\hat{\boldsymbol{y}}_{<t}$, building the final hypothesis $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m)$ in an autoregressive fashion, where:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y}} P(\boldsymbol{y} \mid \boldsymbol{x}) \tag{2}$$

$$= \operatorname*{argmax}_{\boldsymbol{y}} \prod_{t=1}^{|\boldsymbol{y}|} P(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x}) \tag{3}$$

2.3 Simultaneous NMT

Simultaneous NMT (Satija and Pineau, 2016; Cho and Esipova, 2016; Gu et al., 2017; Ma et al., 2019), in which (partial) translation is generated before reading the entire source sentence, can be seen as a very similar task to (simultaneous) kana-

kanji conversion. Simultaneous MT aims at minimizing latency while achieving the highest quality output possible which are also of importance in kana-kanji conversion.

Formally, simultaneous NMT within prefix-to-prefix framework (Ma et al., 2019), models the probability $P(\boldsymbol{y} \mid \boldsymbol{x})$ as the following:

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y}}{\text{argmax}}\, P(\boldsymbol{y} \mid \boldsymbol{x}) \tag{4}$$

$$= \underset{\boldsymbol{y}}{\text{argmax}} \prod_{t=1}^{|\boldsymbol{y}|} P(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq g(t)}) \tag{5}$$

where the monotonic non-decreasing function $g(t)$ denotes a decoding policy, that is, a policy which decides the number of source tokens processed when predicting the target token y_t. In other words, at decoding time t, $g(t)$ source tokens must have been READ before conducting a WRITE operation.

Generally, the decoding policies are divided into fixed and adaptive policies. Fixed policies, as the name suggests, incorporate a prescribed READ/WRITE decision-making which does not rely on the exact input. One of the simplest and most widely used fixed policies is the wait-k policy (Ma et al., 2019) which initially waits for k input, before starting to generate an output on every new input. Adaptive policies on the other hand, usually rely on complex external agents to decide between READ/WRITE operations based on the input. Adaptive policies constitute most of the early work on simultaneous NMT (Satija and Pineau, 2016; Gu et al., 2017) and they can usually be seen as offering better quality-latency (especially non-computational latency) trade-off at the cost of added complexity.

One of the biggest hurdles when it comes to simultaneous MT is anticipation. Anticipation occurs when the model is required to make predictions representing input yet to be seen, forcing it to make predictions based on hallucinated future input, at the cost of translation quality. This is more apparent in Table 1, which illustrates the wait-3 policy failing to keep up with the high fidelity of kanji when applied to kana-kanji conversion, and being forced to anticipate after time step $t \geq 3$.

s	t	Input	Output
1	-	あ	-
2	-	あし	-
3	1	あした	明日
4	2	あしたは	明日は
5	3	あしたはあ	明日は**雨**
6	4	あしたはあめ	明日は**雨**？
7	5	あしたはあめで	明日は**雨**??
8	6	あしたはあめです	明日は**雨**???

Table 1: A hypothetical example for converting the kana sequence “あしたはあめです” (“a shi ta ha a me de su”), meaning “tomorrow will be rainy”, using the wait-3 policy. We can see that after only a few initial inputs, the model is forced to anticipate and generate outputs for input either partially seen (orange/bold) or not seen at all (red question marks).

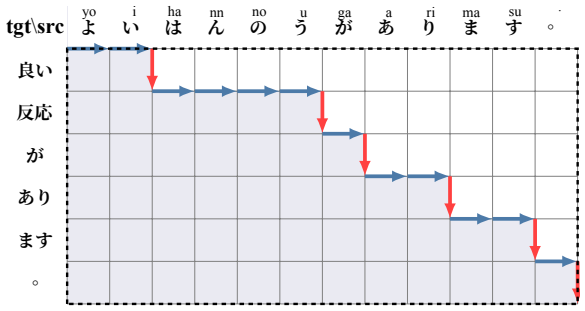


Figure 2: An overview of the alignment-based decoding policy on a source-target grid, showing READ (horizontal) and WRITE (vertical) operations on a sample sentence translating to “There is a good response”.

3 Alignment-Based Decoding Policy

3.1 Policy Definition

As briefly discussed, factors such as high-variance fidelity and different word orders (in case of natural languages) make avoiding anticipation highly challenging when dealing with different source and target languages. Nevertheless, considering the monotonic and many-to-one nature of alignments between kana and kanji, knowing the word-boundaries is sufficient for aligning both sequences when producing intermediate outputs, and consequently, achieving anticipation-free kana-kanji conversion. Figure 1 demonstrates how word-boundaries reveal alignment between those sequences.

Leveraging this fact, we propose a word-boundary-based decoding policy for anticipation-free kana-kanji conversion. This policy is depicted in Figure 2, which shows the source-target grid for a sample conversion. As illustrated, in this policy we perform consecutive READ operations until reaching a word boundary, at which we conduct a WRITE operation, eliminating the need for any anticipation.

Following Equation 5, we define our adaptive decoding policy $g(t)$ as:

$$g(t) = \underset{s}{\text{argmin}} \left(\sum_{1}^{s} c_s = t \right) \tag{6}$$

$$c_s = \mathbf{1}_{a(s) \neq a(s+1)} \tag{7}$$

where c_s indicates the word boundary status of the corresponding hiragana input x_s. Taking the first output y_1 in the example illustrated in Figure 2, the corresponding input, which should be up to after the first input including $t = 1$ boundaries, is $\boldsymbol{x}_{\leq 2}$. This is because $c_1 = 0$ and $c_2 = 1$ due to only x_2 being a word boundary, which results in $g(t) = 2$

3.2 Policy Classifier

To obtain word-boundaries $\hat{\boldsymbol{c}} = \{\hat{c}_1, \ldots, \hat{c}_n\}, \hat{c}_s \in \{0, 1\}$ in an online fashion at inference time, we define $P(\boldsymbol{c} \mid \boldsymbol{x})$ as:

$$\hat{\boldsymbol{c}}^{(0)} = \underset{\boldsymbol{c}}{\text{argmax}}\, P(\boldsymbol{c} \mid \boldsymbol{x}) \tag{8}$$

$$= \underset{\boldsymbol{c}}{\text{argmax}} \prod_{s=1}^{n} P(c_s \mid \boldsymbol{x}_{\leq s}) \tag{9}$$

However, basing boundary predictions only on past context can make accurate boundary predictions, especially at the beginning of the sentence, challenging. To alleviate that problem by also incorporating future context into our boundary predictions, we can redefine $P(\boldsymbol{c} \mid \boldsymbol{x})$ in a wait-k fashion (Ma et al., 2019) as:

$$\hat{\boldsymbol{c}}^{(1)} = \underset{\boldsymbol{c}}{\text{argmax}}\, P(\boldsymbol{c} \mid \boldsymbol{x}) \tag{10}$$

$$= \underset{\boldsymbol{c}}{\text{argmax}} \prod_{s=1}^{n} P(c_s \mid \boldsymbol{x}_{\leq s+k-1}) \tag{11}$$

Nevertheless, the wait-k boundary predictor cannot be seen as a replacement due to online requirements (i.e. considerable lag between input and output being unacceptable), and thus needs to be incorporated alongside $\hat{\boldsymbol{c}}^{(0)}$. That is, we train two separate classifiers, $\hat{\boldsymbol{c}}^{(0)}$, making predictions based on past observations, and $\hat{\boldsymbol{c}}^{(1)}$, making predictions based on future observations, once observed.

In practice, to obtain word-boundaries $\hat{\boldsymbol{c}}^{(0)}$, we add a linear binary classifier with the weights $\boldsymbol{w}^{(0)} \in \mathbb{R}^{d_{model}}$, where d_{model} refers to the output dimensions of the encoder, on top of the encoder stack. This layer makes boundary predictions on each input character x_s, and if the prediction is positive, the model will make predictions in accordance with Equation 7 until it reaches an end-of-word token. That is:

$$\hat{c}_s^{(0)} = \mathbf{1}_{\text{sigmoid}(u^{(0)}(s)) \geq 0.5} \tag{12}$$

$$u^{(0)}(s) = \boldsymbol{w}^{(0)} \cdot \boldsymbol{e}_s + b^{(0)} \tag{13}$$

where $\boldsymbol{e}_s$ refers to encoders hidden state output for the input hiragana x_s.

To calculate $\hat{\boldsymbol{c}}^{(1)}$, we add another linear layer on top of the encoder, this time with weights $\boldsymbol{w}^{(1)} \in \mathbb{R}^{k \times d_{model}}$ which runs in a wait-k fashion, which can also attend future input tokens, making more accurate predictions. Namely:

$$\hat{c}_s^{(1)} = \mathbf{1}_{\text{sigmoid}(u^{(1)}(s)) \geq 0.5} \tag{14}$$

$$u^{(1)}(s) = \boldsymbol{w}^{(1)} \cdot \text{concat}(\boldsymbol{e}_{s:s+k-1}) + b^{(1)} \tag{15}$$

This second classifier is used backtracking and triggering *corrections* if there are any mismatches between our boundary predictors, in which case the boundary prediction results will be overridden by $\hat{\boldsymbol{c}}^{(1)}$. Assuming conversion being at the input step x_ϕ, we have:

$$\hat{c}_s = \begin{cases} \hat{c}_s^{(0)} & \phi - k + 1 < s \leq \phi \\ \hat{c}_s^{(1)} & 0 \leq s \leq \phi - k + 1 \end{cases} \tag{16}$$

This correction mechanism is illustrated in Figure 3. In this example, a prediction mismatch is detected for x_6 while processing the input x_9. This triggers a correction of all the outputs which had attended to x_6, shown in crossed gray blocks.

As for training, the classifiers are trained in multitask settings, in which the model is trained on the objectives of correct kana-kanji conversions and boundary predictions. That is, we aim at minimizing the following loss function:

$$\mathcal{L} = \mathcal{L}_{\text{NMT}} + \lambda \cdot (\mathcal{L}_{\hat{c}^{(0)}} + \mathcal{L}_{\hat{c}^{(1)}}) \tag{17}$$

where $\mathcal{L}_{\text{NMT}}$, $\mathcal{L}_{\hat{c}^{(0)}}$ and $\mathcal{L}_{\hat{c}^{(1)}}$ indicate cross entropy losses for the conversion and boundary predictions respectively, and $\lambda = 20$ denotes the weight for the boundary prediction losses.

4 Experiments and Results

In this section, we will review various experiments to compare both the conversion quality and latencies of our model against various baselines such as the previously-mentioned wait-k policy (Ma et al., 2019), which initially waits for k READs before

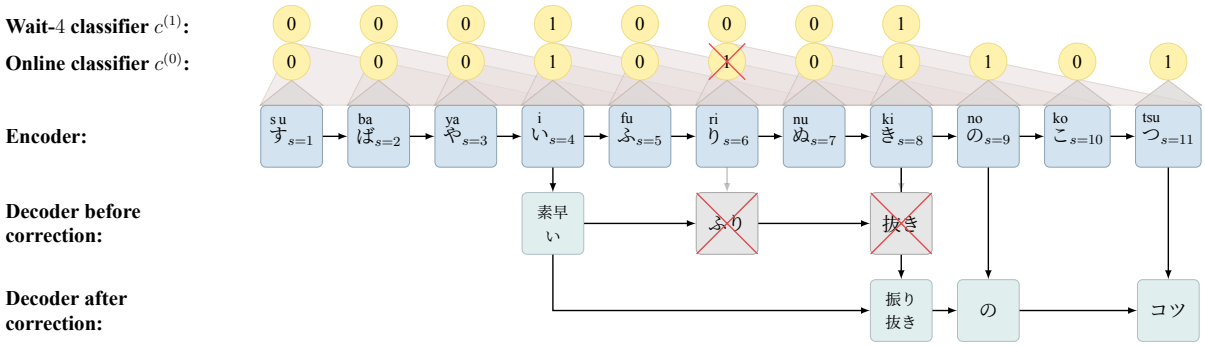


Figure 3: Overview of the incremental encoding/decoding incorporating an alignment-based decoding policy on a sample phrase translating to "Tips for fast (golf) swings". The figure demonstrates a correction step triggered by the mismatch of the outputs of the online and wait-4 classifiers at $s = 6$. Note that some details such as subwords has been omitted for brevity.

conducting a WRITE, the re-translation (Arivazhagan et al., 2020) approach, which treats the conversion as a full-sentence offline conversion on each new input, and our modified version of the wait-k policy, which continuously waits for k input before making an output instead of waiting only before the initial output to provide a stronger baseline by better handling the high fidelity of kanji.

Unlike offline sentence-level kana-kanji conversion, kana-kanji conversion in the context of IMEs has additional requirements besides the final conversion quality, due to its simultaneous nature. Firstly, non-computational latency, which reflects how closely the output follows the input, is a major factor in evaluation. In addition. the computational cost and the intermediate inference latency caused by it becomes a much bigger factor due to near real-time requirements. Finally, intermediate conversion quality, which can be reflected in non-computational latency metrics, is also of importance for this task.

4.1 Experiment Settings

4.1.1 Dataset

The Balanced Corpus of Contemporary Written Japanese (BCCWJ) (Maekawa et al., 2014) is chosen as our dataset. BCCWJ can be seen is the standard dataset used for training and evaluating Japanese IMEs, as it provides a rich, balanced selection of various sources, encompassing a wide range of language varieties. Furthermore, BCCWJ provides a relatively large subset of manually reviewed samples which is crucial for an unbiased evaluation, making it one of the very few, if not the only, viable option.

As briefly mentioned, the annotations provided by BCCWJ contain both automatically generated samples alongside samples annotated by humans, which constitutes around 1% of the corpus, labeled as Core. We limit our test data to the Core section of BCCWJ for a more representative evaluation, in addition the size of the test split is reduced to make evaluation computationally feasible. The details about our splits can be found in Table 2.

Data Splits	#Samples	Avg SRC Length (char)	Avg TGT Length	
			Character	Subword
Train	4,797,944	126.278	99.704	24.966
Val	252,524	135.404	105.913	26.696
Test	2,000	130.277	99.204	21.801

Table 2: Number of samples and their average lengths per our data splits.

To enable incremental encoding, kana sequences are tokenized as characters, on the other hand for the kana-kanji sequences we apply BPE (Sennrich et al., 2016) on top of BCCWJ's short unit word (SUW) tokenization to decrease the vocabulary size, which is set to 16000 in our experiments. To be able to continue to use the word-boundary information provided by the corpus, we employ end-of-word suffixes when conducting subword tokenization. These suffixes later help us detect the equivalent multi-step WRITE operation for each single-step WRITE operation in the case of SUW tokenization.

4.1.2 Implementation Details

Although the word-boundary-based decoding policy can be applied to almost any encoder-decoder network, here we have adopted the Transformer

architecture, as it has been shown to outperform other architectures in various tasks, including NMT.

Besides the auxiliary layers, our implementation mostly follows the standard Transformer architecture, with three small differences:

- We use a causal encoder, which allows us to encode the input incrementally without having to recalculate the hidden states of the previous input tokens, and also make training much more efficient (Elbayad et al., 2020).
- We use a cross-attention mask on training so that each target only attends to previous and its own underlying kana sequences to avoid train-test mismatch.
- We adopt a deep encoder and shallow decoder consisting of 10 and 2 layers respectively. This has two benefits: Higher accuracy for our boundary classifiers due to the increased number of available parameters, and less variance in inference time, especially in the case of corrections which can lead to an increased number of decoding steps.

4.2 Final Conversion Quality

For the evaluation of the final conversion quality, we use precision (P), recall (R), character error rate (CER), and sentence-level accuracy (Acc_{sent}) defined as the following, mostly similar to previous work on IMEs (Mori et al., 1998; Tokunaga et al., 2011; Okuno and Mori, 2012):

$$P = \frac{\text{len}(\text{lcs}(\boldsymbol{y}, \hat{\boldsymbol{y}}))}{\text{len}(\boldsymbol{y})} \quad (18)$$

$$R = \frac{\text{len}(\text{lcs}(\boldsymbol{y}, \hat{\boldsymbol{y}}))}{\text{len}(\hat{\boldsymbol{y}})} \quad (19)$$

$$CER = \frac{\text{lev}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\text{len}(\boldsymbol{y})} \quad (20)$$

$$Acc_{sent} = 1_{(\text{lev}(\boldsymbol{y}, \hat{\boldsymbol{y}})=0)} \quad (21)$$

where $\text{lcs}(\cdot)$ refers to the longest common subsequence, and $\text{lev}(\cdot)$ to the standard Levenshtein distance, both of which are applied on character units, while $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ refer to the reference and predicted kana-kanji mixed sequences respectively.

The results for final conversion quality, provided in Table 3, show that the re-translation framework provides the best final conversion quality. This is as expected, considering that the final quality of the re-translation framework is identical to an offline full-sentence model. On the other hand, the wait-k policy achieves a considerably low conversion quality, something which can be attributed to anticipation caused by the high fidelity of kana-kanji mixed tokens. Finally, following the re-translation framework's results, the modified wait-k policy and our approach provide a close and fairly acceptable final conversion qualities.

4.3 Non-Computational Latency

To assess the non-computational latency, which measures how out of sync the input and output sequences are, we use a slightly modified version of revision-aware average lagging (Zheng et al., 2020) as our metric:

$$\text{RAL}(\boldsymbol{x}, \hat{\boldsymbol{y}}) = \frac{1}{\tau(|\boldsymbol{x}|)} \sum_{t=1}^{\tau(|\boldsymbol{x}|)} LR(t) - \frac{t-1}{r} \quad (22)$$

where $\tau(|\boldsymbol{x}|)$ refers to the cut-off step and $LR(t)$ is defined as the source time step s which corresponds to the last revision of t^{th} target token. However different from Zheng et al. (2020), we define r as $\max(|\hat{\boldsymbol{y}}|, |\boldsymbol{y}^*|)/|\boldsymbol{x}|$ instead of $|\boldsymbol{y}|/|\boldsymbol{x}|$, where $\boldsymbol{y}^*$ represents the reference prediction. This is to better handle under-/over-generations, which are common in some of our baselines.

Referring to the results at Table 3, the wait-k approach, especially at lower k, provides the best latency results, despite a bad conversion quality. This can be attributed to the tendency of wait-k to under-generate, which is caused by too much anticipation due to the high fidelity of the target sequences. Next, we can see the results for our policy followed by the results for the re-translation approach, both of which also offer high-quality conversions. The reason re-translation shows higher latency than our model, is due to its sometimes lower intermediate conversion quality, which can also be reflected in the RAL metric. Finally, we can see that the modified version of wait-k entails quite high latency numbers, as opposed to its good conversion quality.

4.4 Computational Latency

Computational latency is one of the most important factors in IMEs, especially when it comes to on-device processing. To measure the computational latency, we track the time spent between each READ operation on each test sample, and then calculate the total, maximum, and average time spent between READs based on those. Maximum time

Model	Conversion Quality					Non-Computational latency	Computational latency		
	P	R	F_1	CER	Acc_{sent}	RAL	Mean	Max	Total
Re-translation	95.58	95.68	95.61	5.00	54.95	3.44	68	188	4943
Wait-3	35.40	34.11	30.96	143.51	0.04	1.38	8	32	314
Wait-6	52.13	52.83	48.81	97.46	11.85	3.07	8	31	348
Wait-9	63.80	66.47	62.40	63.56	19.65	4.60	8	24	321
Mod-wait-2	72.88	72.35	71.57	65.89	18.65	5.38	10	35	444
Mod-wait-3	93.37	93.34	93.31	7.76	44.3	11.08	9	43	438
Mod-wait-6	95.29	95.41	95.33	5.32	53.2	19.08	9	65	454
Ours	94.47	94.74	94.58	6.17	47.4	2.72	10	22	496

Table 3: Experiment results of various approaches (single runs). Results for computational latency are provided in milliseconds, with tests conducted on two Xeon Gold 6230R CPUs using the PyTorch 2.1 (Paszke et al., 2019) library in online settings without batch-processing to be more representative of real-world usage. Note that unlike the original wait-k approach (Ma et al., 2019), we use a causal encoder similar to Elbayad et al. (2020) alongside a deep encoder accompanied by a shallow decoder for our wait-k implementation to allow direct comparison. Likewise, our re-translation baselines also incorporates a (bidirectional) deep encoder and a shallow decoder.

spent is an important factor when it comes to measuring tangible lags by the user, while mean/total times spent provide a basis for evaluating the overall efficacy of the approaches.

Table 3 also provides the computational latency of our approach and other baselines. We can see that simultaneous incremental approaches all provide a much more efficient conversion compared to the re-translation approach. While the mean times spent are mostly similar within those approaches, the standard wait-k policy provides the smallest total time spent. With regard to the maximum time spent, our approach demonstrates the best results.

Lastly, Figure 4 allows us to better see the latency and quality trade-offs between all the approaches. Here, we can see that both our and the re-translation approaches give very good results for non-computational latency and conversion qualities. However, taking the computational latency into account, our approach provides a better solution with a substantial margin.

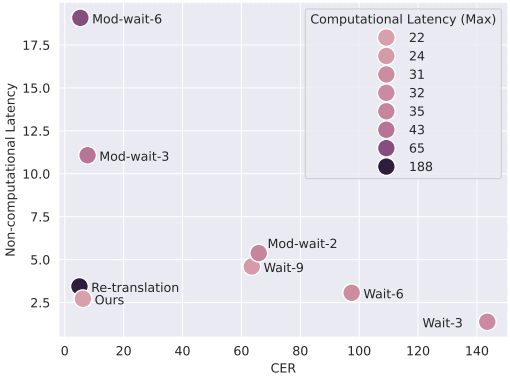


Figure 4: Latencies and CER trade-off between baselines.

5 Analysis of Boundary Predictors

Boundary predictions play a central role in the implementation of our policy, as conversion errors caused by a wrong boundary prediction can propagate to the next tokens and drastically affect the final conversion quality of our model. In this section, we will evaluate the performance of our boundary predictors in isolation, and further analyze its effects on end-to-end results.

Model	P	R	F_1	Acc_{sent}	#cor
Online	93.92	94.02	93.77	20.90	0
$k=2$	98.63	98.27	98.37	63.20	2.184
$k=4$	99.48	99.48	99.45	86.75	2.352
$k=6$	99.58	99.63	99.58	89.90	2.363

Table 4: Evaluation of the boundary predictors with respect to reference boundary information. #cor denotes the average number of corrections happening per sentence if the given predictor was used for corrections alongside the online predictor.

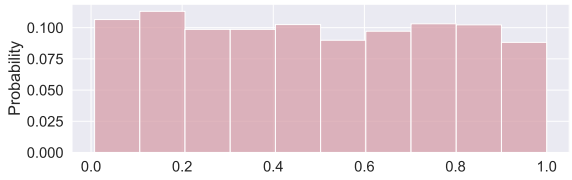


Figure 5: Relative position of corrections happening in a sentence.

Boundary Prediction Quality Table 4 shows the boundary prediction results of our online and wait-k predictors with various k values. We can observe that even for low values of k, the wait-k predictor greatly improves prediction accuracy over the online predictor. Interestingly, the number of average corrections per sentence in the case of using two predictors as discussed previously, stays mostly the same regardless of k, and appears to stabilize for $k \geq 4$.

Effects on End-to-End Results While boundary predictor results offer insights, they don't reveal their impact on overall model performance. We thus proceed to have a look at the end-to-end results of our model concerning our predictors in different settings, and also compare those with the model's performance on reference boundary data. The results, represented in Table 5, demonstrate that there is a noticeable gap between conversion qualities when using an online predictor versus using the reference boundary data, and that incorporating wait-k predictors alongside the online predictor helps us to greatly close that gap, achieving better conversion quality. As for non-computational latency, we can observe that incorporating a wait-k predictor for backtracking can actually slightly reduce the latency potentially due to better conversion quality, however, the latency keeps increasing as k increases, overtaking the benefit margin. Finally, as expected, the correction mechanism can increase the computation time, although the effects seem to be quite marginal.

Relative Position of Corrections Figure 5 shows the relative positions of a sentence in which boundary prediction mismatches, and consequently corrections, occur. We can observe that corrections tend to happen mostly uniformly across sentences, with a slightly higher change to happen at the beginning of a sentence, enabling a consistent experience for the end user with regard to possible corrections made.

6 Previous Work

In the conventional approach to Japanese IMEs, kana-kanji conversion is conducted within the noisy channel framework which incorporates a (either statistical or neural) language model, and a translation model, to find the best kana-kanji representation for a kana sequence (Mori et al., 1998; Kudo, 2011; Yao et al., 2018).

Besides the conventional framework, other approaches such as using discriminative methods (Tokunaga et al., 2011) and incorporating a joint source channel model (Okuno and Mori, 2012) have been proposed to further improve the results.

Although limited to Chinese, more recently there have been some attempts to incorporate modern neural architectures in IMEs. Huang et al. (2018) implements a Chinese IME by formulating pinyin to Chinese conversion as a neural machine translation (NMT) task and uses an attention-based encoder-decoder model. Nonetheless, using a full-sentence model and a bi-directional encoder makes it computationally inefficient. Tan et al. (2022) explores using GPT (Radford and Narasimhan, 2018; Radford et al., 2019) for pinyin to Chinese conversion, however, their solutions are also either incompatible with Japanese or computationally inefficient for real-world scenarios.

7 Conclusion and Future Work

In this work we proposed the alignment-based decoding policy for achieving high-quality low-latency simultaneous kana-kanji conversion using neural encoder-decoder networks. Our results demonstrate that neural encoder-decoder networks can be a viable option for high-quality Japanese IMEs, and we thus encourage more exploration in that area, especially with regards to efficiently adding common IME features such as next-word prediction.

In the future, we hope to further address remaining efficacy questions such as memory consumption when it comes to utilizing neural encoder-decoders for IMEs to enable more viable on-device solutions.

8 Limitations

Although our proposed approach is not limited to Japanese, its advantages might no longer be tangible when applied to other languages due to some language-specific assumptions such as unsegmented input and a many-to-one alignment be-

Model	Conversion Quality					Non-Computational latency	Computational latency		
	P	R	F_1	CER	Acc_{sent}	RAL	Mean	Max	Total
Reference (w/o correction)	95.07	95.27	95.15	5.51	50.25	2.55	10	18	486
Online (w/o correction)	88.07	88.35	88.12	14.28	21.9	2.62	10	20	486
$k = 2(w/correction)$	92.95	93.05	92.97	8.09	40.00	2.61	10	20	477
$k = 4(w/correction)$	94.48	94.74	94.58	6.17	47.4	2.72	10	21	496
$k = 6(w/correction)$	94.58	94.81	94.67	6.04	48.6	2.92	10	24	495

Table 5: End-to-end results for various boundary predictor settings.

tween input and output. For example, in the case of some types of simultaneous pinyin to Chinese conversion, we are provided with the segmented input and we can assume a fidelity of one, which could potentially make the conversion possible with a simpler model.

Moreover, despite their advantages, neural-network-based approaches (especially approaches based on encoder-decoder architectures) can have some disadvantages compared to conventional methods. For instance, in addition to the higher computational cost, providing customizations such as modifiable vocabulary and online learning are not trivial in deep learning approaches, requiring further investigation regarding their viability.

References

N. Arivazhagan, Colin Cherry, Wolfgang Macherey, and George F. Foster. 2020. Re-translation versus Streaming for Simultaneous Translation. In *International Workshop on Spoken Language Translation*.

Kyunghyun Cho and Masha Esipova. 2016. Can neural machine translation do simultaneous translation? *ArXiv*, abs/1606.02012.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020. Efficient Wait-k Models for Simultaneous Machine Translation. Publisher: arXiv Version Number: 2.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. 2017. Learning to Translate in Real-time with Neural Machine Translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1053–1062, Valencia, Spain. Association for Computational Linguistics.

Yafang Huang, Zuchao Li, Zhuosheng Zhang, and Hai Zhao. 2018. Moon IME: Neural-based Chinese Pinyin Aided Input Method with Customizable Association. In *Proceedings of ACL 2018, System Demonstrations*, pages 140–145, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo. 2011. Tōkei-teki kanakanji henkan shisutemu Mozc [Statistical kana-kanji conversion system Mozc]. *The 17th Annual Conference of the Association for Natural Language Processing, 2011*.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. 2019. STACL: Simultaneous Translation with Implicit Anticipation and Controllable Latency using Prefix-to-Prefix Framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, Florence, Italy. Association for Computational Linguistics.

Kikuo Maekawa, Makoto Yamazaki, Toshinobu Ogiso, Takehiko Maruyama, Hideki Ogura, Wakako Kashino, Hanae Koiso, Masaya Yamaguchi, Makiro Tanaka, and Yasuharu Den. 2014. Balanced corpus of contemporary written Japanese. *Language Resources and Evaluation*, 48(2):345–371.

Shinsuke Mori, Masatoshi Tsuchiya, Osamu YAMAJI, and Makoto Nagao. 1998. Kana-Kanji Conversion by A Stochastic Model. *IPSJ SIG Notes*, 21:75–81. Publisher: Information Processing Society of Japan (IPSJ).

Yoh Okuno and Shinsuke Mori. 2012. An Ensemble Model of Word-based and Character-based Models for Japanese and Chinese Input Method. In *Proceedings of the Second Workshop on Advances in Text Input Methods*, pages 15–28, Mumbai, India. The COLING 2012 Organizing Committee.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Harsh Satija and Joelle Pineau. 2016. Simultaneous machine translation using deep reinforcement learning.

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.

Minghuan Tan, Yong Dai, Duyu Tang, Zhangyin Feng, Guoping Huang, Jing Jiang, Jiwei Li, and Shuming Shi. 2022. Exploring and Adapting Chinese GPT to Pinyin Input Method. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1899–1909, Dublin, Ireland. Association for Computational Linguistics.

Hiroyuki Tokunaga, Daisuke Okanohara, and Shinsuke Mori. 2011. Discriminative Method for Japanese Kana-Kanji Input Method. In *WTIM@IJCNLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need.

Jiali Yao, Raphael Shu, Xinjian Li, Katsutoshi Ohtsuki, and Hideki Nakayama. 2018. Real-time Neural-based Input Method.

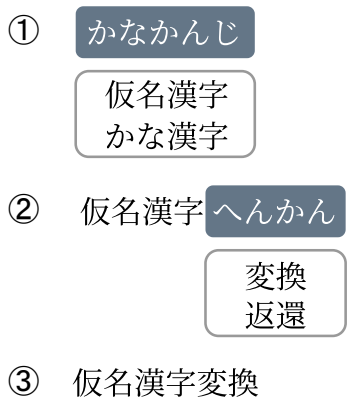
Renjie Zheng, Mingbo Ma, Baigong Zheng, Kaibo Liu, and Liang Huang. 2020. Opportunistic Decoding with Timely Correction for Simultaneous Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 437–442, Online. Association for Computational Linguistics.

A User Experience of Japanese IMEs

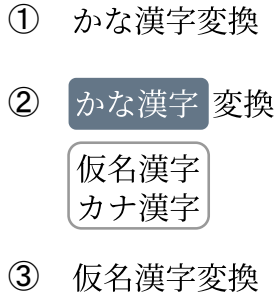
Although most Japanese IMEs provide a similar user experience (UX) for inputting text via kana-kanji conversion, the small differences can potentially have great implications on the underlying approach and also the evaluation strategy. Two approaches can generally be found within the design of Japanese IMEs from the view point of UX paradigms, *opt-in conversion*, in which an IME expects explicit user actions for kana-kanji conversion, and *opt-out conversion*, in which the aim is to conduct the conversion with as little interruption as possible.

Opt-in Conversion The UX of most common approaches to kana-kanji conversion can be considered to fall into the opt-in conversion paradigm, which heavily relies on explicit user actions.

The opt-in paradigm, does not conduct any implicit conversions on the input, rather it requires the user to opt in for the desired conversion by explicitly choosing from a candidate list provided separately from the input, as demonstrated in Figure 6a.



(a) The opt-in approach. Here, in Step 1, the user has to explicitly select a candidate for the input kana to conduct conversion. Step 2 shows the same process repeated for the second chunk to get the final output of Step 3.



(b) The opt-out approach. Here, in Step 1, the conversion is already implicitly applied on top the user input without any explicit actions. Step 2 shows an optional user interaction to fix a hypothetical mistake to get the final output of Step 3.

Figure 6: Kana-kanji conversion within the two different UX paradigms for the user input phrase “かなかんじへんかん” (“ka na ka n ji he n ka n”), meaning “kana-kanji conversion”.

The opt-in approach encourages the users to conduct conversions in short chunks rather than longer phrases or sentence units, which can result in many manual interactions by the user to conduct the conversion. In addition, conducting conversions in short chunks leads to more limited context for the IME, and subsequently lower quality suggestions.

Opt-out Conversion The alternative paradigm, namely the opt-out conversion, tries to provide a more seamless and interruption-free experience by reducing the required number of user actions.

The opt-out conversion paradigm, illustrated in Figure 6b, simultaneously applies the best conver-

sion prediction on top of the input field as the user types without requiring any explicit actions. In this approach the suggestions are usually hidden by default to encourage the user to continue typing uninterrupted while also providing more context to the IME for better conversion, and are only shown by explicit user actions or in situations such as low confidence conversions by the IME (i.e Step 2 in Figure 6b).

The opt-out conversion paradigm can lead to a more efficient experience in practice as most conversions will not require explicit corrections, especially that longer context can help the IME provide higher quality conversions. Due to this, we propose our approach with the opt-out paradigm in mind, meaning focusing on sentence-level and top-1 conversion results instead of ranking-based results of shorter chunks.

B Prediction Errors Caused by Our Approach

Generally the errors occurring within our approach can be categorized in two main groups, namely:

- Errors caused by wrong boundary predictions
- Errors caused by alternative writing styles

Errors Caused by Wrong Boundary Predictions This type of error is mostly unique to our approach as it heavily relies on word boundary predictions to conduct conversion. Although subword tokenization appears to allow our approach to recover from some wrong boundary predictions, at points such mistakes lead to actual conversion errors, as shown in the below example:

Source: e n ze tsuwo chū shi
えんぜつをちゅうし

Predicted Boundaries: e n ze tsuwo ch u shi
えんぜつをちゅうし

Prediction: enzetsu wo chūshi
...演説 を 中止
(...stop the speech)

Reference: enzetsu wo chūshi
...演説 を 注視
(...pay close attention to the speech)

in which the model predicts an extra boundary between "ちゅう" ("chū") and "し" ("shi"), leading to an incorrect conversion for the word "注視".

Errors Caused by Alternative Writing Styles This type of error is not really unique to our proposed approach but rather originates from the characteristic of the Japanese language, in which one word can have multiple acceptable surface forms (with potentially variable dependency on context). This issue is usually mitigated by presenting the user with a ranked list of conversion candidates instead or alongside the best prediction. Below shows an example of such a conversion error made by our model:

Source: ze hi ze hi ha i ke n shi ta i
ぜひぜひ はいけんしたい

Prediction: zehizehi haiken shitai
ぜひぜひ、拝見したい...
(...would really like to see...)

Reference: zehizehi haiken shitai
是非是非、拝見したい...
...would really like to see...)

here the alternative writing of "是非" has lead to a different prediction compared to the reference.

In addition two the two main groups, another notable error category, although less frequent, consists of words unseen in the training data, particularly proper nouns which can have a much greater degree of irregularities, making it harder for the model to guess the correct surface form based on contextual cues. However, similar to the preceding category, the challenge of *out-of-vocabulary* (OOV) words is not unique to our approach.

This category of errors can potentially be mitigated by standard online learning approaches.

C Training Environment and Hyperparameters

In this section we briefly discuss the training environment details and hyperparameters not mentioned in the main body of this paper. For finer details please refer to our code.

Hardware Training was conducted on two 40GB NVIDIA A100 GPUs, totalling around 6.2 GPU hours per model. Although even much shorter training time of around 2.0 GPU hours showed fairly acceptable results in our experiments.

Software Table 6 includes some of the main software and packages used in our work.

Hyperparameters Table 7 represents some of the main details and hyperparameters used for our

Name	Version
cuda-runtime	11.8.0
libcublas	11.11.3.6
libblas	3.9.0
numpy	1.26.0
pandas	2.1.1
python	3.11.6
pytorch	2.1.0
pytorch-lightning	2.0.9
sentencepiece	0.1.99
tokenizers	0.13.3

Table 6: Package versions.

training. The choices for hyperparameters are made mostly based on the standard Transformer implementation (Vaswani et al., 2017), task requirements (e.g. deep encoder and shallow decoder for faster inference), and manual tuning. The values are kept consistent between various models as much as possible to enable direct comparison of evaluation results.

Hyperparameter	Value
Activation	ReLU
d_{model}	512
Dropout	0.1
Max Len	1024
Epochs	9
#Attention Heads	8
#Decoder Layers	2
#Encoder Layers	10
Source Vocab Size	300
Target Vocab Size	16000
Optimizer	AdamW
LR	0.0005
LR Scheduler	Linear + Cosine Decay
Warmup Steps	2500
Weight Decay	0.0001
Betas	(0.9, 0.98)
Epsilon	1.0e-09
#Total Parameters	$\sim$48M

Table 7: Model details and hyperparameters.