Non-Autoregressive Grammatical Error Correction Toward a Writing Support System

Hiroki Homma and **Mamoru Komachi**
Tokyo Metropolitan University
homma-hiroki@ed.tmu.ac.jp komachi@ed.tmu.ac.jp

Abstract

There are several problems in applying grammatical error correction (GEC) to a writing support system. One of them is the handling of sentences in the middle of the input. Till date, the performance of GEC for incomplete sentences is not well-known. Hence, we analyze the performance of each model for incomplete sentences. Another problem is the correction speed. When the speed is slow, the usability of the system is limited, and the user experience is degraded. Therefore, in this study, we also focus on the non-autoregressive (NAR) model, which is a widely studied fast decoding method. We perform GEC in Japanese with traditional autoregressive and recent NAR models and analyze their accuracy and speed.

1 Introduction

Grammatical error correction (GEC) is a writing support method for language learners. In recent years, neural GEC has been actively researched owing to its ability to produce fluent text. For example, in Kiyono et al. (2019), state-of-the-art correction accuracy was achieved by using a Transformer (Vaswani et al., 2017), which is a powerful neural machine translation (NMT) model. Because the neural model can see the entire sequence, it can correct errors with long-range dependencies; these errors cannot be corrected by a statistical method that uses n-grams.

However, considering the application of GEC in a writing support system, we must consider how to handle incomplete sentences. It is easy to present the GEC result when the user finishes writing a sentence. However, in case of an incomplete sentence, the user will not know how to fix the sentence while writing it. If the system can perform GEC correctly for incomplete sentences, the results can be presented to the user. In most previous studies, complete sentences have been evaluated, and the performance of GEC for incomplete sentences has not been researched.

In addition, there is a problem that inference speed is slow in a conventional autoregressive (AR) decoder of a sequence-to-sequence model. Considering the application of GEC in a writing support system, a slower inference speed would restrict its utility or lower the usability of the model. In Gu et al. (2018), a non-autoregressive (NAR) decoder that speeds up inference time by outputting all tokens simultaneously was proposed. Following the success of NAR models, in Gu et al. (2019), Levenshtein Transformer, an NAR NMT model that iteratively deletes and inserts inputs, was proposed. Its usefulness was verified in machine translation and document summarization tasks.

Moreover, fast GEC methods with sequence tagging using an NAR model have been proposed. In Awasthi et al. (2019), GEC was regarded as a local sequence conversion task, and high-speed GEC was achieved by using an NAR model that iteratively adapted editing tags in parallel. In Omelianchuk et al. (2020), NAR GEC was performed by repetitive tagging of editing operations on each token of an input sentence, and higher correction accuracy and faster correction speed than in previous studies were achieved. However, these methods exhibited good performance by narrowing down the target language to English and preparing the editing operations as tags using language knowledge in advance.

In this study, we focus on the NAR model as a method for high-speed GEC. We perform GEC in Japanese using the NAR model that does not need to prepare editing operations in advance. We analyze the proposed method considering its application to writing support systems. In particular, we analyze the relationship between the correction accuracy and the inference speed, focusing on incomplete sentences, and evaluate the impact of hyperparameters on NAR models. The contributions of this study can be summarized as follows.

- We evaluate the performance of NAR and AR models for incomplete sentences in terms of accuracy and speed, aiming for the construction of a writing support system.
- We show that the Levenshtein Transformer that performs one-time iterative refinements can achieve fast and stable GEC by reducing the worst inference time by 6.0 seconds and the average inference time by 0.3 seconds compared with the method based on convolutional neural networks (CNNs).
- Using the NAR model for Japanese GEC, we find that it is better to present the GEC result when the number of input words is six or more because the accuracy is significantly reduced when the number is less than five.

2 Related Work

2.1 AR NMT

AR NMT is a standard decoding method in the encoder–decoder model (Kalchbrenner and Blunsom, 2013) for sequence-to-sequence learning. This method uses a recurrent language model (Mikolov et al., 2010) during inference.

Given an original sentence, $X = \{x_1, \ldots, x_{T'}\}$, and an objective sentence, $Y = \{y_1, \ldots, y_T\}$, an AR NMT model calculates the target sentence as

$$p(Y|X;\theta) = \prod_{t=1}^{T+1} p(y_t|y_{0:t-1}, x_{1:T'};\theta), \quad (1)$$

where y_0 and y_{T+1} are special tokens representing the beginning and end of the sentence, respectively, and θ is the model's parameter.

2.2 NAR NMT

NAR NMT (Gu et al., 2018) is a decoding method that generates each token independently and simultaneously. This method is attracting attention as a method to increase the speed of decoding.

In Gu et al. (2018), the concept of fertility, which predicts how many words on the target side correspond to each word in the source side, was introduced. The decoding is performed as follows:

$$p(Y|X;\theta) = \sum_{f_1,\ldots,f_{T'} \in \mathcal{F}} \left(\prod_{t'=1}^{T'} p_F(f_{t'}|x_{1:T'};\theta) \cdot \prod_{t=1}^{T} p(y_t|x_1\{f_1\}, \ldots, x_{T'}\{f_{T'}\};\theta) \right), \quad (2)$$

where $\mathcal{F}$ is the set of all fertility sequences that sum into the length of Y, and $x\{f\}$ represents token x repeated f times. As described earlier, it is necessary to predict the target sentence length in the NAR decoding method.

Furthermore, NAR NMT involves a problem named the multimodality problem (Gu et al., 2018). This problem causes errors (such as token repetitions and a lack of tokens) and significantly deteriorates accuracy compared with an AR decoder. To solve this problem, in recent studies, iteratively refining the output (Lee et al., 2018; Gu et al., 2019) and partially autoregressively outputting the sentence divided into segments (Ran et al., 2020) have been proposed. Knowledge distillation (KD) (Kim and Rush, 2016) is also used to address this problem (Zhou et al., 2020). The output of the AR model is known to mitigate multimodality problems because diversity is suppressed such that the model can be easily learned (Ren et al., 2020).

2.3 Levenshtein Transformer

Levenshtein Transformer (Gu et al., 2019) is one of the most recent NAR NMT models[1] that introduces a workaround for the aforementioned multimodality problems. In Gu et al. (2019), the usefulness of the Levenshtein Transformer in machine translation and summarization tasks was verified; however, its usefulness in GEC has not been verified yet.

This model has a Transformer (Vaswani et al., 2017) block (T-block) as a primary component, and the original text is given to each T-block. First, the states coming from the lth T-block are as fol-

[1]In the original paper, it is called a "partially autoregressive model"; however, in this paper, we call it an NAR model because it is a model that outputs all tokens simultaneously when decoding.

lows:

$$\boldsymbol{h}_0^{(l+1)}, \boldsymbol{h}_1^{(l+1)}, \ldots, \boldsymbol{h}_n^{(l+1)} = \begin{cases} E_{y_0} + P_0, E_{y_1} + P_1, \ldots, E_{y_n} + P_n, & l = 0 \\ \text{T-block}_l \left(\boldsymbol{h}_0^{(l)}, \boldsymbol{h}_1^{(l)}, \ldots, \boldsymbol{h}_n^{(l)}\right), & l > 0 \end{cases} \tag{3}$$

where E and P are token and position embeddings, respectively; y_0 and y_n are boundary tokens representing the start and end, respectively. Next, we use these decoder outputs, $(\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_n)$, to classify deletions, placeholders, and tokens. The deletion classifier uses $\mathrm{softmax}\left(\boldsymbol{h}_i \cdot A^\top\right), (i = 1, \ldots n-1)$ to perform binary classification of "deleted" or "kept" for tokens other than boundary tokens. Next, it deletes corresponding tokens. The placeholder classifier uses $\mathrm{softmax}\left(\mathrm{concat}\left(\boldsymbol{h}_i, \boldsymbol{h}_{i+1}\right) \cdot B^\top\right), (i = 0, \ldots n-1)$ to classify how many placeholders to insert from 0 to $K_{\max}$ at every consecutive position pair. Subsequently, it inserts the corresponding number of special tokens `<PLH>`, where $K_{\max}$ is the maximum number of tokens that can be inserted at one time in one place, and we set it to 255. The token classifier uses $\mathrm{softmax}\left(\boldsymbol{h}_i \cdot C^\top\right), (\forall y_i = \texttt{<PLH>})$ to classify and replace all special tokens `<PLH>` into words that are elements of vocabulary $\mathcal{V}$. Here, A, B, and C are matrices for linearly transforming the number of dimensions of a state or a combination of two states into the number of classes.

2.4 GEC

GEC is a task to correct errors, such as punctuation, grammar, and word selection errors. Various methods have been studied for this task. In recent years, owing to the development of NMT, GEC is often interpreted as a machine translation task. Almost all studies using the BEA Shared Task-2019 datasets (Bryant et al., 2019) used Transformer-based models (Omelianchuk et al., 2020; Kiyono et al., 2019; Kaneko et al., 2020; Grundkiewicz et al., 2019; Choe et al., 2019; Li et al., 2019). For example, in Li et al. (2019), a system that combined a CNN-based model with a Transformer-based model was used, and the method in Chollampatt and Ng (2018) was adopted as the CNN architecture.

The following are previous studies on high-speed GEC using an NAR model. In Awasthi et al. (2019), GEC was regarded as a local sequence conversion task, and it was rapidly solved by using a parallel iterative editing model. In Omelianchuk et al. (2020), the same task was solved by iterative sequence tagging. However, both methods applied linguistic knowledge prepared in advance (such as suffix conversion rules and verb conjugation dictionaries). Thus, it is not easy to apply them to another language. In this study, we propose a method that uses only a training corpus.

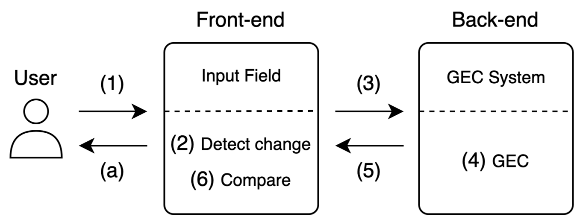


Figure 1: Schematic diagram of a writing support system.

3 Proposed Method

This study aims to analyze the effectiveness of an NAR model for Japanese GEC in terms of accuracy and speed, assuming that NAR is used as a back-end of a writing support system.

3.1 Writing Support System

In this section, we explain the workflow of a writing support system. Figure 1 shows a schematic diagram of the system, which consists of a front-end with a text input field and a back-end with a GEC system, and it works as follows. (1) The user inputs or deletes the text in the input field[2]. (2) The front-end detects the change; (3) it sends the changed sentence to the back-end. (4) The back-end performs GEC; (5) it sends the correction to the front-end. (6) The front-end checks for changes; (a) it suggests changes to the user if there are changes; (b) otherwise, it does nothing.

3.2 Challenges for the System

In this section, we consider the system input ((1) and (2)) and response time ((2) to (6a)).

System Input We consider two problems with input from users of the system.

The first is how to process a sentence in the middle of input. When the user finishes writing a sentence (in other words, when the user enters a line),

[2]For simplicity, we assume that the user enters one sentence per line. In other words, line breaks divide the sentences.

the GEC result of the sentence should be presented to the user. However, it is unclear how to deal with an incomplete sentence. This is because the back-end system may not perform accurate GEC owing to its incompleteness or shortness. Thus, we propose the following hypothesis: if the incomplete sentence is short, the correction accuracy deteriorates; however, if it is long, the correction accuracy approaches that of the complete sentence. We verify this hypothesis in Subsection 4.2.

The second is the problem of the Japanese input method. In Japanese, unlike English, user inputs are processed through a kana–kanji conversion system[3]. In other words, when the front-end is receiving the text through the kana–kanji conversion, it is not evident in what unit (character, word, phrase, or whole sentence) the errors can be appropriately detected[4]. In this study, we assume that users are intermediate Japanese learners and treat the input string as words.

Response Time The processing speed from (2) to (6a) in the system flow dominates the response time of the system, which affects the user experience. It is known that not only is responsiveness required, but also users prefer a system with constant response speed over a system with variable response speed (Shneiderman, 1979). We analyze the processing time of GEC in Subsection 4.3.

4 Experiment

4.1 Experimental Settings

Dataset We use data from the Lang-8 learner corpus (Mizumoto et al., 2011). We use the TMU Evaluation Corpus for Japanese Learners (Koyama et al., 2020) for the validation and test sets[5]. All data, including the training set, are preprocessed as in Koyama et al. (2020). Table 1 presents the number of sentences in the data. We use the same training set in our experiments with both complete and incomplete sentences.

To evaluate the performance of GEC for incomplete sentences, we segment the test data to the word level and then create incomplete sentences by increasing the number of words from the beginning. For example, 10 sentences are created from a 10-word sentence. Next, based on the word alignment between the source and target sentences, we create parallel sentences for incomplete sentences. Consequently, 9,710 sentence pairs are created. We use these data to evaluate the performance of GEC for incomplete sentences.

	# of sentences	# of corrections
Train	1,093,633	1
Validation	806	2
Test	663	3

Table 1: Dataset statistics. The number of corrections denotes the number of reference sentences for one learner's sentence.

Tokenization We tokenize data in all models as follows. First, we segment data into morpheme units using MeCab[6] (Ver. 0.996) using the UniDic[7] (Ver. 2.2.0) as a dictionary. Next, we divide the morpheme units into subword units by applying the byte pair encoding (Sennrich et al., 2016) model for dealing with rare words. We apply character normalization (compatibility decomposition, followed by canonical composition) and share vocabulary between source and target sides.[8] The vocabulary size was set to 30,000 words. We use sentencepiece[9] for implementation.

NAR Model In this study, we apply the Levenshtein Transformer (Gu et al., 2019), which is a Transformer-based NAR neural model, to GEC. We update the model 300,000 times with a batch size of 64,000 tokens and select the model with the highest GLEU score (Napoles et al., 2016) for the validation set. Other hyperparameters are the same as in Gu et al. (2019). We use publicly available PyTorch-based code[10] for implementation. In this paper, this model is called the LevT model.

The maximum number of iterative refinements was set to nine in a previous study (Gu et al.,

[3]The kana–kanji conversion system translates the input hiragana (the Japanese cursive syllabary) into kanji (Chinese characters) when necessary.

[4]The appropriate unit may change depending on the user's language learning level and Japanese input ability level.

[5]These data are less noisy than the corrected sentences included initially in the Lang-8 learner corpus and have multiple references to all sentences, which is considered useful for evaluation.

[6]https://taku910.github.io/mecab/

[7]https://unidic.ninjal.ac.jp/

[8]As a preliminary experiment, the source side was set to the character unit, and the target side was set to the subword unit; however, the GLEU score (Napoles et al., 2016) was slightly decreased; therefore, we decided to tokenize both sides in the subword units.

[9]https://github.com/google/sentencepiece

[10]https://github.com/pytorch/fairseq/tree/master/examples/nonautoregressive_translation

2019). However, it is unclear whether it is the correct value for GEC because GEC is a local sequence conversion task in which almost all the source words remain in the target side. Therefore, we also evaluate the performance when the maximum iterative refinement number is changed.

Training data are obtained by replacing the corrected sentences with the output of an AR model. We use it to train a KD model (Zhou et al., 2020) of LevT. The hyperparameters are the same as for the LevT model, and the model described in the next paragraph is used for the AR model. In this paper, this model is called the LevT+KD model.

AR Model Because we focus on speeding up GEC, we adopt the CNN-based model (Chollampatt and Ng, 2018), which is faster than the Transformer-based model as the AR baseline. Unlike Chollampatt and Ng (2018), the output is not reranked to match the conditions with the LevT model. Other hyperparameters are the same as in Chollampatt and Ng (2018). We use publicly available PyTorch-based code[11] for implementation. In this paper, this model is called the CNN model.

Correction Evaluation We use the GLEU score (Napoles et al., 2016) and the $F_{0.5}$ score, which weighs the precision as twice the recall, as evaluation metrics for the correction accuracy of GEC. We map words automatically using the ERRANT[12] to calculate $F_{0.5}$. However, because the ERRANT is designed for English, we cannot use it directly; instead, we specify the Levenshtein distance in the distance function that calculates the edit distance without using linguistic information. Furthermore, we measure the $F_{0.5}$ score in terms of the word-wise agreement.

Inference Speed Evaluation We measure the inference speed with the following settings. We use the Intel® Xeon® processor E5-2660 without GPUs. To measure the performance in a realistic setting, we set the batch size to one sentence. We measure time using the built-in `time` module in Python. Specifically, the inference speed of one sentence is calculated as the change in the system clock from the input of the sentence before tokenization to the output of the GEC result.

Model	GLEU	Prec.	Rec.	$F_{0.5}$
CNN	73.3	0.159	**0.225**	0.169
LevT	72.1	0.102	0.185	0.112
LevT+KD	**75.2**	**0.213**	0.217	**0.214**

Table 2: Correction accuracy of each model for complete sentences from the test set. "Prec." and "Rec." represent precision and recall, respectively.

Model	M (242)	R (441)	U (124)
CNN	**33**	73	**32**
LevT	22	54	5
LevT+KD	**33**	**79**	20

Table 3: Number of errors, according to category, that each model modified correctly on the test set. "M," "R," and "U" represent missing, replacement, and unnecessary errors, respectively. Each number in parentheses represents the maximum error frequency[13].

4.2 Correction Accuracy

To confirm GEC's effectiveness on complete sentences, we evaluate each model using the test set.

Table 2 lists the results. Both GLEU and $F_{0.5}$ scores of LevT without KD are worse than those of CNN, but LevT+KD's score exceeds CNN's score. In terms of the recall and precision of LevT and LevT+KD, both are improved by KD, and the precision is significantly increased. Therefore, KD dramatically improves the precision in GEC.

For a more detailed analysis of the effect of KD, the number of categorical errors that the model correctly changed is presented in Table 3. Comparing LevT and LevT+KD, it can be seen that the number of corrections for all types of errors has increased owing to KD. In particular, the correction accuracy for "unnecessary" errors has increased. We believe that this is because LevT+KD can inherit the correction accuracy for the "unnecessary" errors of the CNN by KD.

LevT+KD has a correction accuracy comparable to that of the CNN. As KD's effectiveness in the NAR model for GEC is confirmed, we focus only on LevT+KD for the NAR model in the subsequent experiments.

[11] https://github.com/nusnlp/mlconvgec2018

[12] https://github.com/chrisjbryant/errant

[13] The maximum number of corrections made by each of the three annotators is shown. The smallest ones are 210, 428, and 103.

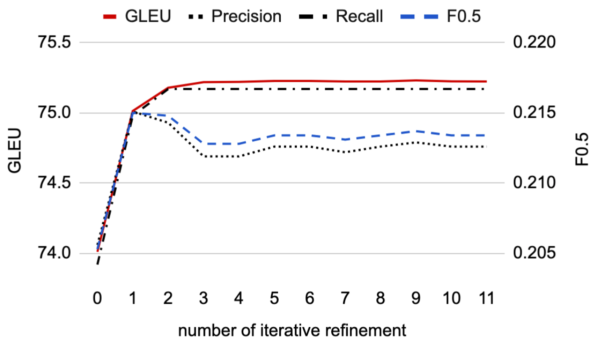


Figure 2: Accuracy of LevT+KD model for the maximum number of iterative refinements. The solid red, dotted black, dash-dotted, and broken blue-black lines represent the GLEU score, precision, recall, and $F_{0.5}$ score, respectively.

Model	GLEU	Prec.	Rec.	$F_{0.5}$
CNN	74.6	0.100	**0.199**	0.111
LevT+KD	**76.9**	**0.125**	0.184	**0.133**

Table 4: Correction accuracy of each model for incomplete sentences.

Number of Iterative Refinements Unlike in the machine translation task (which was mainly addressed in the previous studies on NAR models), it seems that the necessary number of iterative refinements is reduced in the GEC task because the input and the output are close. Therefore, we evaluate the change in performance because of the number of iterative refinements in the LevT+KD model.

Figure 2 shows the result of the GLEU and $F_{0.5}$ scores for the best epoch selected in the validation set. We can see that after the first iteration, the GLEU score does not change significantly with the maximum number of iterative refinements, and it is almost optimal when the number is three. Furthermore, similar to the GLEU score, we can see that the change in the $F_{0.5}$ score after the first iteration is small. Moreover, when the number of iterations is more than one, the score degrades with a decrease in precision. When the maximum number of iterations is one, the score becomes the maximum. Therefore, there is little need to increase the maximum number of iterative refinements of LevT+KD in GEC. One to three iterations are sufficient.

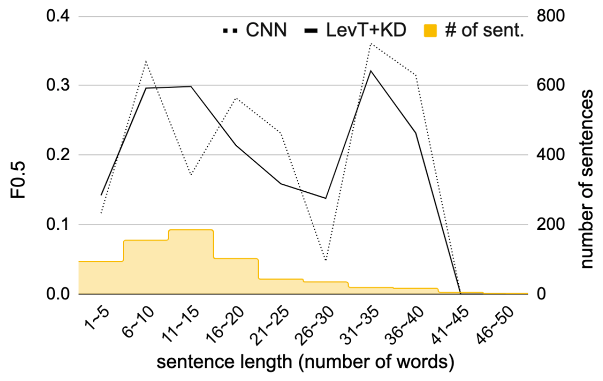
Accuracy for Incomplete Sentences Table 4 shows each model's overall correction accuracy for incomplete sentences. Compared with Table 2, it can be seen that the overall tendency is the same: LevT+KD has a high GLEU score and a high precision, whereas CNN has a high recall. Furthermore, in both models, the GLEU score improves slightly, and the $F_{0.5}$ score deteriorates for incomplete sentences. Overall, the GEC model trained only on complete sentences is useful to some extent, even for incomplete sentences.

Figure 3 shows each model's correction accuracy per sentence length. Comparing the incomplete sentences (b) with the complete sentences (a), we can see that the accuracy is considerably reduced when the sentence length is extremely short in both models. When presenting the GEC result for incomplete sentences, it is considered appropriate not to show it when the input sentence length is short. In addition, the correction accuracy for the complete sentences fluctuates substantially in the range of 31–50 words in both models. This might be attributed to the lack of test sentences.

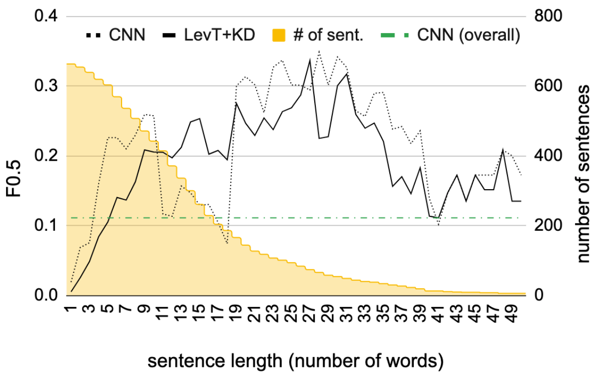
4.3 Inference Speed

Figure 4 shows the inference speed of test data containing 9,710 incomplete sentences for each model. The average inference times are 0.49, 0.24, and 0.19 seconds for CNN, LevT+KD with the maximum number of iterations set as nine, and LevT+KD with the maximum iterations set as one, respectively. According to our results, the variance of the inference time of LevT+KD is significantly suppressed compared with that of CNN, and the average time is also significantly lower than that of CNN. The variance and average can be further suppressed by reducing the maximum number of iterations. Excluding the outliers, CNN also fits in approximately one second. However, the correspondence between the inference speeds of each model does not change, and LevT+KD is faster than CNN. Here, most sentences with outliers are long sentences created from sentences whose original length is 100 words or more, and we believe that the lengthy sentences are the leading cause of the increase in inference time.

Figure 5 depicts the inference speed for each sentence length of each model. Focusing on the linear approximation, we find that CNN is faster than each LevT+KD when the sentence length is extremely short (one to four words). We assume that this is because the Levenshtein Transformer executes three types of operations: delete, insert a



(a) For complete sentences.



(b) For incomplete sentences.[14]

Figure 3: Correction accuracy per sentence length breakdown for complete and incomplete sentences. The solid, dotted, and straight dash-dotted green lines represent the $F_{0.5}$ scores of LevT+KD and CNN, and CNN's overall $F_{0.5}$ score, respectively. The step-form graph represents the number of sentences.

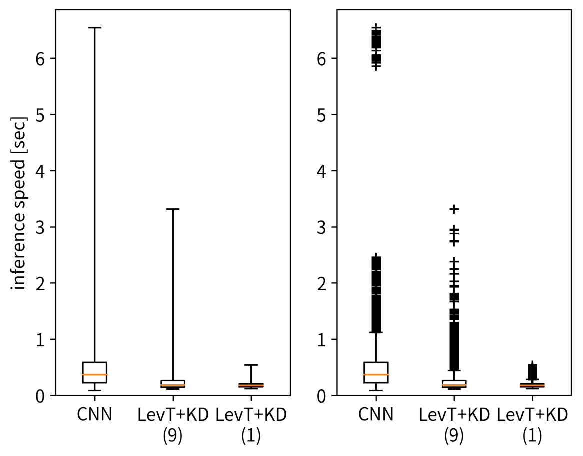


Figure 4: Inference speed of each model. The number in parentheses in the model name represents the maximum number of iterations. The graph on the left does not consider outliers, and the graph on the right shows outliers as "+" in the range where the whiskers length exceeds 1.5 times the interquartile range.

placeholder, and replace it with a token in one iterative refinement, thereby having more overhead than the CNN model does. However, the results show that each LevT+KD model is faster than CNN when the sentence length is five words or more.

Here, we analyze the effect of sentence length on incomplete sentences in terms of both correction accuracy and speed. As shown in Figure 3b, when CNN's overall $F_{0.5}$ score for the incomplete sentence is used as the minimum criterion, the LevT+KD model's scores with five or fewer words

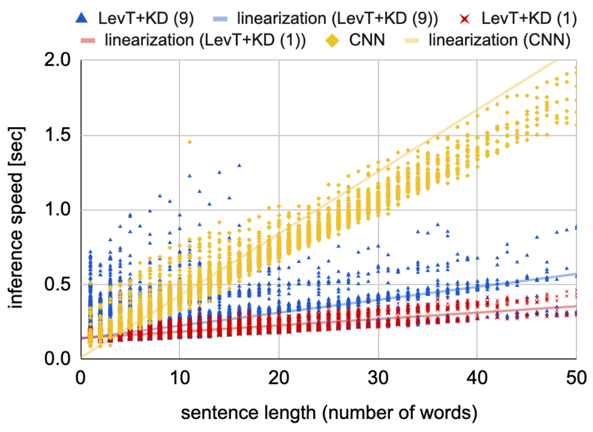


Figure 5: Inference speed of each model per sentence length breakdown. Each straight line represents a linear approximation, and the number in parentheses in the model name represents the maximum number of iterations.

are below the standard. Furthermore, Figure 5 shows that LevT+KD is consistently faster than CNN after six words or more. In other words, by using the LevT+KD model and performing GEC when six or more words are input, it is possible to present the correction results at high speed while maintaining a certain degree of correction accuracy[15].

4.4 Case Study

We show examples of system output in Table 5. In (1), a sentence in which "ですか? *desuka?*"

[14]Note that the performance is stable because more sentences are used to evaluate the GEC model than complete sentences.

[15]Sentences of five or fewer words account for approximately 32.7% of the incomplete sentences used in this experiment. Furthermore, in reality, considering that long sentences are corrected many times, it is believed that the rate of short sentences of fewer than five words is even lower.

(1)	Learner's sentence	これはほんとに大切だか?
	CNN	これはほんとに大切**なの**?
	LevT+KD	これはほんとに大切**です**か?
	Corrected sentence	これはほんとに大切だろうか? "Is this really important?"
(2)	Learner's sentence	かわいいくて、安い、素敵な生地です。
	CNN	**かわいい**て、安い、素敵な生地です。
	LevT+KD	**かわい**て、安い、素敵な生地です。
	Corrected sentence	かわいくて、安い、素敵な生地です。 "It's a cute, cheap and lovely fabric."

Table 5: Output examples for each model. Grammatical errors are underlined. Boldface represents where the model has changed the text. Double quotes represent the meaning of the sentence.

Input: learner's sentence	きの よる は たく やき パーチイ ー い ます 。												
`insert`[17](0)	**きのう**		**よる**	**は**	**たくさん**		**パーティー**	**パーティー**		**し**	**ます**	。
`delete` (1)	きのう		よる	は	たくさん		パーティー	~~パーティー~~		し	ます	。
`insert` (1)	きのう		よる	は	たくさん	**やき**	パーティー		**を**	し	ます	。
`delete` (2)	きのう		よる	は	たくさん	やき	パーティー		を	し	ます	。
`insert` (2)	きのう	**の**	よる	は	たくさん	やき	パーティー		を	し	ます	。
	kinou	*no*	*yoru*	*wa*	*takusan*	*yaki*	*paatii*	*paatii*	*wo*	*shi*	*masu*	
System output sentence	きのう の よる は たくさん やき パーティー を し ます 。 "I have a lot of bake party last night."											
Corrected sentence	きのう の よる は たこやき パーティー に い まし た 。 "I was at a Takoyaki party last night."											

Table 6: Example of iterative refinements. The number of iterations is written in parentheses. Grammatical errors are underlined. Boldface represents the inserted word, and strikethrough represents the deleted word. Italics represent the Japanese pronunciations of each word, and double quotes represent the meaning of the sentence.

is mistaken for "だか? *daka?*"[16] is input. The correction differs depending on the model, but the outputs of both models are grammatically correct. In (2), a sentence in which "かわいく *kawaiku*," which is a conjunctive form of "かわいい *kawaii*" (cute), is mistaken for "かわいいく *kawaiiku*" is input. Both models changed the error part, but both outputs are grammatically incorrect. We believe that the reason for this is that there are few similar error examples in the training set. In the training set, there are 172 errors of "だか? *daka?*," whereas only two errors of "かわいいくて *kawaiikute*." We assume that the performance of the sequence-to-sequence GEC method is limited by the number of similar errors in the training set.

Table 6 presents an example of iterative refinements in LevT+KD. In the first insertion phase, a missing token error and repeated token errors have occurred. The repeated token, "パーティー *paatii*" (party), is deleted in the next deletion, and in the next insertion phase, the missing tokens, "やき *yaki*" (bake) and "を *wo*" (accusative case marker), are inserted to the left and right of "パーティー *paatii*," recovering from the multimodality problem. However, erroneous parts remain: "たくやき *takuyaki*," which is a misspelling of "たこやき *takoyaki*" (octopus dumplings), is mistakenly corrected as "たくさん やき *takusan yaki*." Moreover, "ます *masu*" (politeness marker), which should be corrected to the past form corresponding to "きのう *kinou*" (yesterday), is not corrected.

5 Conclusion

In this study, we investigated the applicability of the NAR model, which has a constant inference speed, to Japanese GEC toward constructing a writing support system. The experiments showed that the NAR model can obtain a correction accuracy that is equal to or better than that of the AR multilayer convolutional neural model. Furthermore, we demonstrated that the GEC model trained on complete sentences can also be applied to incomplete sentences. However, we found that when the number of input words is small, the correction accuracy is significantly lower than that of the complete sentence. Therefore, the system should defer presenting correction results for short

[16]The Japanese question marker particle, "か *ka*," cannot be added at the end of a sentence in the plain-style sentence.

[17]`insert` shows the result of both inserting the placeholder and replacing it with the actual token. In addition, because it starts with an empty string, there is no `delete` in the first iteration.

sentences. We also showed that the worst inference time could be reduced by approximately 6.0 seconds, and the average inference time could be reduced by approximately 0.3 seconds in the NAR model that performs one-time iterative refinement compared with the AR model.

Future work includes an extrinsic evaluation of the GEC system integrated into a writing support system. Moreover, we plan to investigate a large-scale pretrained model to improve GEC's performance.

Acknowledgments

We want to thank Yangyang Xi for consenting to use text from Lang-8.

References

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4260–4270. Association for Computational Linguistics.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75. Association for Computational Linguistics.

Yo Joong Choe, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. 2019. A neural grammatical error correction system built on better pre-training and sequential transfer learning. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 213–227. Association for Computational Linguistics.

Shamil Chollampatt and Hwee Tou Ng. 2018. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5755–5762. Association for the Advancement of Artificial Intelligence.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11181–11191. Curran Associates, Inc.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709. Association for Computational Linguistics.

Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki, and Kentaro Inui. 2020. Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4248–4254. Association for Computational Linguistics.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327. Association for Computational Linguistics.

Shun Kiyono, Jun Suzuki, Masato Mita, Tomoya Mizumoto, and Kentaro Inui. 2019. An empirical study of incorporating pseudo data into grammatical error correction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1236–1242. Association for Computational Linguistics.

Aomi Koyama, Tomoshige Kiyuna, Kenji Kobayashi, Mio Arai, and Mamoru Komachi. 2020. Construction of an evaluation corpus for grammatical error correction for learners of Japanese as a second language. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 204–211. European Language Resources Association.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182. Association for Computational Linguistics.

Ruobing Li, Chuan Wang, Yefei Zha, Yonghong Yu, Shiman Guo, Qiang Wang, Yang Liu, and Hui Lin. 2019. The LAIX systems in the BEA-2019 GEC shared task. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 159–167. Association for Computational Linguistics.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048. International Symposium on Computer Architecture.

Tomoya Mizumoto, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2011. Mining revision log of language learning SNS for automated Japanese error correction of second language learners. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 147–155. Asian Federation of Natural Language Processing.

Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2016. GLEU without tuning. *eprint arXiv:1605.02592 [cs.CL]*.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170. Association for Computational Linguistics.

Qiu Ran, Yankai Lin, Peng Li, and Jie Zhou. 2020. Learning to recover from multi-modality errors for non-autoregressive neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3059–3069. Association for Computational Linguistics.

Yi Ren, Jinglin Liu, Xu Tan, Zhou Zhao, Sheng Zhao, and Tie-Yan Liu. 2020. A study of non-autoregressive model for sequence generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 149–159. 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. Association for Computational Linguistics.

Ben Shneiderman. 1979. Human factors experiments in designing interactive systems. *Computer*, 12(12):9–19.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Chunting Zhou, Jiatao Gu, and Graham Neubig. 2020. Understanding knowledge distillation in non-autoregressive machine translation. In *International Conference on Learning Representations*.