Instantaneous Grammatical Error Correction with Shallow Aggressive Decoding

Xin Sun[1*†] **Tao Ge**[2†] **Furu Wei**[2] **Houfeng Wang**[1]
[1] MOE Key Lab of Computational Linguistics, School of EECS, Peking University;
[2] Microsoft Research Asia
{sunx5,wanghf}@pku.edu.cn;
{tage,fuwei}@microsoft.com

Abstract

In this paper, we propose Shallow Aggressive Decoding (SAD) to improve the online inference efficiency of the Transformer for instantaneous Grammatical Error Correction (GEC). SAD optimizes the online inference efficiency for GEC by two innovations: 1) it aggressively decodes as many tokens as possible in parallel instead of always decoding only one token in each step to improve computational parallelism; 2) it uses a shallow decoder instead of the conventional Transformer architecture with balanced encoder-decoder depth to reduce the computational cost during inference. Experiments in both English and Chinese GEC benchmarks show that aggressive decoding could yield the same predictions as greedy decoding but with a significant speedup for online inference. Its combination with the shallow decoder could offer an even higher online inference speedup over the powerful Transformer baseline without quality loss. Not only does our approach allow a single model to achieve the state-of-the-art results in English GEC benchmarks: 66.4 $F_{0.5}$ in the CoNLL-14 and 72.9 $F_{0.5}$ in the BEA-19 test set with an almost $10\times$ online inference speedup over the Transformer-big model, but also it is easily adapted to other languages. Our code is available at https://github.com/AutoTemp/Shallow-Aggressive-Decoding.

1 Introduction

The Transformer (Vaswani et al., 2017) has become the most popular model for Grammatical Error Correction (GEC). In practice, however, the sequence-to-sequence (seq2seq) approach has been blamed recently (Chen et al., 2020; Stahlberg and Kumar, 2020; Omelianchuk et al., 2020) for its poor inference efficiency in modern writing assistance applications (e.g., Microsoft Office Word[1], Google Docs[2] and Grammarly[3]) where a GEC model usually performs online inference, instead of batch inference, for proactively and incrementally checking a user's latest completed sentence to offer instantaneous feedback.

To better exploit the Transformer for instantaneous GEC in practice, we propose a novel approach – Shallow Aggressive Decoding (SAD) to improve the model's online inference efficiency. The core innovation of SAD is aggressive decoding: instead of sequentially decoding only one token at each step, aggressive decoding tries to decode as many tokens as possible in parallel with the assumption that the output sequence should be almost the same with the input. As shown in Figure 1, if the output prediction at each step perfectly matches its counterpart in the input sentence, the inference will finish, meaning that the model will keep the input untouched without editing; if the output token at a step does not match its corresponding token in the input, we will discard all the predictions after the bifurcation position and re-decode them in the original autoregressive decoding manner until we find a new opportunity for aggressive decoding. In this way, we can decode the most text in parallel in the same prediction quality as autoregressive greedy decoding, but largely improve the inference efficiency.

In addition to aggressive decoding, SAD proposes to use a shallow decoder, instead of the conventional Transformer with balanced encoder-decoder depth, to reduce the computational cost for further accelerating inference. The experimental

[*] This work was done during the author's internship at MSR Asia. Contact person: Tao Ge (tage@microsoft.com)
[†] Co-first authors with equal contributions

[1] https://www.microsoft.com/en-us/microsoft-365/word
[2] https://www.google.com/docs/about
[3] https://www.grammarly.com

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 5937–5947
August 1–6, 2021.

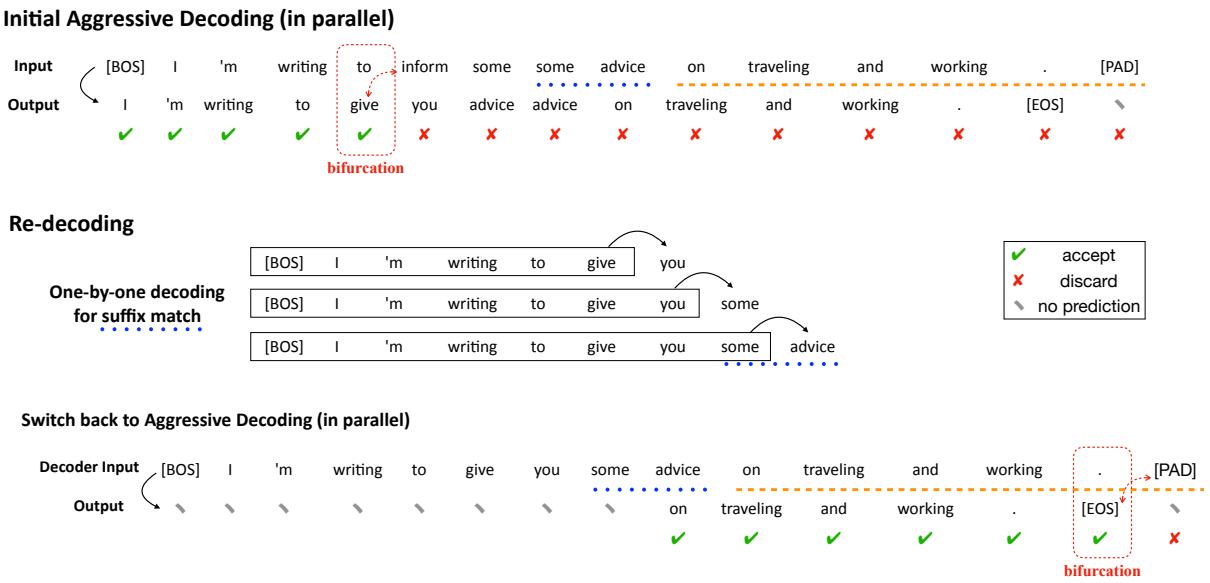

Figure 1: The overview of aggressive decoding. Aggressive decoding tries decoding as many tokens as possible in parallel with the assumption that the input and output should be almost the same in GEC. When we find a bifurcation between the input and the output of aggressive decoding, then we accept the predictions before (including) the bifurcation, and discard all the predictions after the bifurcation and re-decode them using original one-by-one autoregressive decoding. If we find a suffix match (i.e., *some advice* highlighted with the blue dot lines) between the output and the input during one-by-one re-decoding, we switch back to aggressive decoding by copying the tokens (highlighted with the orange dashed lines) following the matched tokens in the input to the decoder input by assuming they are likely to be the same.

results in both English and Chinese GEC benchmarks show that both aggressive decoding and the shallow decoder can significantly improve online inference efficiency. By combining these two techniques, our approach shows a $9\times \sim 12\times$ online inference speedup over the powerful Transformer baseline without sacrificing the quality.

The contributions of this paper are two-fold:

- We propose a novel aggressive decoding approach, allowing us to decode as many token as possible in parallel, which yields the same predictions as greedy decoding but with a substantial improvement of computational parallelism and online inference efficiency.
- We propose to combine aggressive decoding with the Transformer with a shallow decoder. Our final approach not only advances the state-of-the-art in English GEC benchmarks with an almost $10\times$ online inference speedup but also is easily adapted to other languages.

2 Background: Transformer

The Transformer is a seq2seq neural network architecture based on multi-head attention mechanism, which has become the most successful and widely used seq2seq models in various generation tasks such as machine translation, abstractive summarization as well as GEC.

The original Transformer follows the balanced encoder-decoder architecture: its encoder, consisting of a stack of identical encoder layers, maps an input sentence $\boldsymbol{x} = (x_1, \ldots, x_n)$ to a sequence of continuous representation $\boldsymbol{z} = (z_1, \ldots, z_n)$; and its decoder, which is composed of a stack of the same number of identical decoder layers as the encoder, generates an output sequence $\boldsymbol{o} = (o_1, \ldots, o_m)$ given $\boldsymbol{z}$.

In the training phase, the model learns an autoregressive scoring model $P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\Phi})$, implemented with teacher forcing:

$$\begin{aligned} \boldsymbol{\Phi}^* &= \arg\max_{\boldsymbol{\Phi}} \log P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\Phi}) \\ &= \arg\max_{\boldsymbol{\Phi}} \sum_{i=0}^{l-1} \log P(y_{i+1} \mid \boldsymbol{y}_{\leq i}, \boldsymbol{x}; \boldsymbol{\Phi}) \end{aligned} \tag{1}$$

where $\boldsymbol{y} = (y_1, \ldots, y_l)$ is the ground-truth target sequence and $\boldsymbol{y}_{\leq i} = (y_0, \ldots, y_i)$. As ground truth is available during training, Eq (1) can be efficiently obtained as the probability $P(y_{i+1} \mid \boldsymbol{y}_{\leq i}, \boldsymbol{x})$ at each step can be computed in parallel.

During inference, the output sequence $\boldsymbol{o} =$

$(o_1, \ldots, o_m)$ is derived by maximizing the following equation:

$$\begin{aligned} \boldsymbol{o}^* &= \arg\max_{\boldsymbol{o}} \log P(\boldsymbol{o} \mid \boldsymbol{x}; \boldsymbol{\Phi}) \\ &= \arg\max_{\boldsymbol{o}} \sum_{j=0}^{m-1} \log P(o_{j+1} \mid \boldsymbol{o}_{\leq j}, \boldsymbol{x}; \boldsymbol{\Phi}) \end{aligned} \tag{2}$$

Since no ground truth is available in the inference phase, the model has to decode only one token at each step conditioning on the previous decoded tokens $\boldsymbol{o}_{\leq j}$ instead of decoding in parallel as in the training phase.

3 Shallow Aggressive Decoding

3.1 Aggressive Decoding

As introduced in Section 2, the Transformer decodes only one token at each step during inference. The autoregressive decoding style is the main bottleneck of inference efficiency because it largely reduces computational parallelism.

For GEC, fortunately, the output sequence is usually very similar to the input with only a few edits if any. This special characteristic of the task makes it unnecessary to follow the original autoregressive decoding style; instead, we propose a novel decoding approach – aggressive decoding which tries to decode as many tokens as possible during inference. The overview of aggressive decoding is shown in Figure 1, and we will discuss it in detail in the following sections.

3.1.1 Initial Aggressive Decoding

The core motivation of aggressive decoding is the assumption that the output sequence $\boldsymbol{o} = (o_1, \ldots, o_m)$ should be almost the same with the input sequence $\boldsymbol{x} = (x_1, \ldots, x_n)$ in GEC. At the initial step, instead of only decoding the first token o_1 conditioning on the special [*BOS*] token o_0, aggressive decoding decodes $\boldsymbol{o}_{1\ldots n}$ conditioning on the pseudo previous decoded tokens $\hat{\boldsymbol{o}}_{0\ldots n-1}$ in parallel with the assumption that $\hat{\boldsymbol{o}}_{0\ldots n-1} = \boldsymbol{x}_{0,\ldots,n-1}$. Specifically, for $j \in \{0, 1, \ldots, n-2, n-1\}$, o_{j+1} is decoded as follows:

$$\begin{aligned} o^*_{j+1} &= \arg\max_{o_{j+1}} \log P(o_{j+1} \mid \boldsymbol{o}_{\leq j}, \boldsymbol{x}; \boldsymbol{\Phi}) \\ &= \arg\max_{o_{j+1}} \log P(o_{j+1} \mid \hat{\boldsymbol{o}}_{\leq j}, \boldsymbol{x}; \boldsymbol{\Phi}) \\ &= \arg\max_{o_{j+1}} \log P(o_{j+1} \mid \boldsymbol{x}_{\leq j}, \boldsymbol{x}; \boldsymbol{\Phi}) \end{aligned} \tag{3}$$

where $\hat{\boldsymbol{o}}_{\leq j}$ is the pseudo previous decoded tokens at step $j+1$, which is assumed to be the same with $\boldsymbol{x}_{\leq j}$.

After we obtain $\boldsymbol{o}_{1\ldots n}$, we verify whether $\boldsymbol{o}_{1\ldots n}$ is actually identical to $\boldsymbol{x}_{1\ldots n}$ or not. If $\boldsymbol{o}_{1\ldots n}$ is fortunately exactly the same with $\boldsymbol{x}_{1\ldots n}$, the inference will finish, meaning that the model finds no grammatical errors in the input sequence $\boldsymbol{x}_{1\ldots n}$ and keeps the input untouched. In more cases, however, $\boldsymbol{o}_{1\ldots n}$ will not be exactly the same with $\boldsymbol{x}_{1\ldots n}$. In such a case, we have to stop aggressive decoding and find the first bifurcation position k so that $\boldsymbol{o}_{1\ldots k-1} = \boldsymbol{x}_{1\ldots k-1}$ and $o_k \neq x_k$.

Since $\boldsymbol{o}_{1\ldots k-1} = \hat{\boldsymbol{o}}_{1\ldots k-1} = \boldsymbol{x}_{1\ldots k-1}$, the predictions $\boldsymbol{o}_{1\ldots k}$ could be accepted as they will not be different even if they are decoded through the original autoregressive greedy decoding. However, for the predictions $\boldsymbol{o}_{k+1\ldots n}$, we have to discard and re-decode them because $o_k \neq \hat{o}_k$.

3.1.2 Re-decoding

As $o_k \neq \hat{o}_k = x_k$, we have to re-decode for o_{j+1} $(j \geq k)$ one by one following the original autoregressive decoding:

$$o^*_{j+1} = \arg\max_{o_{j+1}} P(o_{j+1} \mid \boldsymbol{o}_{\leq j}, \boldsymbol{x}; \boldsymbol{\Phi}) \tag{4}$$

After we obtain $\boldsymbol{o}_{\leq j}$ $(j > k)$, we try to match its suffix to the input sequence $\boldsymbol{x}$ for further aggressive decoding. If we find its suffix $\boldsymbol{o}_{j-q\ldots j}$ $(q \geq 0)$ is the unique substring of $\boldsymbol{x}$ such that $\boldsymbol{o}_{j-q\ldots j} = \boldsymbol{x}_{i-q\ldots i}$, then we can assume that $\boldsymbol{o}_{j+1\ldots}$ will be very likely to be the same with $\boldsymbol{x}_{i+1\ldots}$ because of the special characteristic of the task of GEC.

If we fortunately find such a suffix match, then we can switch back to aggressive decoding to decode in parallel with the assumption $\hat{\boldsymbol{o}}_{j+1\ldots} = \boldsymbol{x}_{i+1\ldots}$. Specifically, the token o_{j+t} $(t > 0)$ is decoded as follows:

$$o^*_{j+t} = \arg\max_{o_{j+t}} P(o_{j+t} \mid \boldsymbol{o}_{<j+t}, \boldsymbol{x}; \boldsymbol{\Phi}) \tag{5}$$

In Eq (5), $\boldsymbol{o}_{<j+t}$ is derived as follows:

$$\begin{aligned} \boldsymbol{o}_{<j+t} &= \textsc{Cat}(\boldsymbol{o}_{\leq j}, \hat{\boldsymbol{o}}_{j+1\ldots j+t-1}) \\ &= \textsc{Cat}(\boldsymbol{o}_{\leq j}, \boldsymbol{x}_{i+1\ldots i+t-1}) \end{aligned} \tag{6}$$

where $\textsc{Cat}(\boldsymbol{a}, \boldsymbol{b})$ is the operation that concatenates two sequences $\boldsymbol{a}$ and $\boldsymbol{b}$.

Otherwise (i.e., we cannot find a suffix match at the step), we continue decoding using the original

Algorithm 1 Aggressive Decoding

Input: $\mathbf{\Phi}$, $\boldsymbol{x} = ([BOS], x_1, \ldots, x_n, [PAD])$, $\boldsymbol{o} = (o_0) = ([BOS])$;
Output: $\boldsymbol{o}_{1\ldots j} = (o_1, \ldots, o_j)$;

1: Initialize $j \leftarrow 0$;
2: **while** $o_j \neq [EOS]$ and $j <$ MAX_LEN **do**
3: **if** $\boldsymbol{o}_{j-q\ldots j}$ $(q \geq 0)$ is a unique substring of $\boldsymbol{x}$ such that $\exists\,!\, i : \boldsymbol{o}_{j-q\ldots j} = \boldsymbol{x}_{i-q\ldots i}$ **then**
4: Aggressive Decode $\widetilde{\boldsymbol{o}}_{j+1\ldots}$ according to Eq (5) and Eq (6);
5: Find bifurcation $j+k$ $(k > 0)$ such that $\widetilde{\boldsymbol{o}}_{j+1\ldots j+k-1} = \boldsymbol{x}_{i+1\ldots i+k-1}$ and $\widetilde{o}_{j+k} \neq x_{i+k}$;
6: $\boldsymbol{o} \leftarrow \textsc{Cat}(\boldsymbol{o}, \widetilde{\boldsymbol{o}}_{j+1\ldots j+k})$;
7: $j \leftarrow j + k$;
8: **else**
9: Decode $o^*_{j+1} = \arg\max_{o_{j+1}} P(o_{j+1} \mid \boldsymbol{o}_{\leq j}, \boldsymbol{x}; \mathbf{\Phi})$;
10: $\boldsymbol{o} \leftarrow \textsc{Cat}(\boldsymbol{o}, o^*_{j+1})$;
11: $j \leftarrow j + 1$;
12: **end if**
13: **end while**

autoregressive greedy decoding approach until we find a suffix match.

We summarize the process of aggressive decoding in Algorithm 1. For simplifying implementation, we make minor changes in Algorithm 1: 1) we set $o_0 = x_0 = [BOS]$ in Algorithm 1, which enables us to regard the initial aggressive decoding as the result of suffix match of $o_0 = x_0$; 2) we append a special token $[PAD]$ to the end of $\boldsymbol{x}$ so that the bifurcation (in the 5^{th} line in Algorithm 1) must exist (see the bottom example in Figure 1). Since we discard all the computations and predictions after the bifurcation for re-decoding, aggressive decoding guarantees that generation results are exactly the same as greedy decoding (i.e., beam=1). However, as aggressive decoding decodes many tokens in parallel, it largely improves the computational parallelism during inference, greatly benefiting the inference efficiency.

3.2 Shallow Decoder

Even though aggressive decoding can significantly improve the computational parallelism during inference, it inevitably leads to intensive computation and even possibly introduces additional computation caused by re-decoding for the discarded predictions.

To reduce the computational cost for decoding, we propose to use a shallow decoder, which has proven to be an effective strategy (Kasai et al., 2020; Li et al., 2021) in neural machine translation (NMT), instead of using the Transformer with balanced encoder-decoder depth as the previous state-of-the-art Transformer models in GEC. By combining aggressive decoding with the shallow decoder, we are able to further improve the inference efficiency.

4 Experiments

4.1 Data and Model Configuration

We follow recent work in English GEC to conduct experiments in the restricted training setting of BEA-2019 GEC shared task (Bryant et al., 2019): We use Lang-8 Corpus of Learner English (Mizumoto et al., 2011), NUCLE (Dahlmeier et al., 2013), FCE (Yannakoudakis et al., 2011) and W&I+LOCNESS (Granger; Bryant et al., 2019) as our GEC training data. For facilitating fair comparison in the efficiency evaluation, we follow the previous studies (Omelianchuk et al., 2020; Chen et al., 2020) which conduct GEC efficiency evaluation to use CoNLL-2014 (Ng et al., 2014) dataset that contains 1,312 sentences as our main test set, and evaluate the speedup as well as Max-Match (Dahlmeier and Ng, 2012) precision, recall and $F_{0.5}$ using their official evaluation scripts[4]. For validation, we use CoNLL-2013 (Ng et al., 2013) that contains 1,381 sentences as our validation set. We also test our approach on NLPCC-18 Chinese GEC shared task (Zhao et al., 2018), following their training[5] and evaluation setting, to verify the effectiveness of our approach in other languages. To compare with the state-of-the-art approaches in English GEC that pretrain with synthetic data,

[4] https://github.com/nusnlp/m2scorer

[5] Following Chen et al. (2020), we sample 5,000 training instances as the validation set.

Model	Synthetic Data	Total Latency (s)	Speedup	CoNLL-13		
				P	R	$F_{0.5}$
Transformer-big (beam=5)	No	440	1.0×	**53.84**	18.00	**38.50**
Transformer-big (greedy)	No	328	1.3×	52.75	**18.34**	38.36
Transformer-big (aggressive)	No	**54**	**8.1×**	52.75	**18.34**	38.36
Transformer-big (beam=5)	Yes	437	1.0×	**57.06**	23.62	44.47
Transformer-big (greedy)	Yes	320	1.4×	56.45	**24.70**	**44.91**
Transformer-big (aggressive)	Yes	**60**	**7.3×**	56.45	**24.70**	**44.91**

Table 1: The performance and online inference efficiency of the Transformer-big with aggressive decoding in our validation set (CoNLL-13) that contains 1,381 sentences. We use Transformer-big (beam=5) as the baseline to compare the performance and efficiency of aggressive decoding.

we also synthesize 300M error-corrected sentence pairs for pretraining the English GEC model following the approaches of Grundkiewicz et al. (2019) and Zhang et al. (2019). Note that in the following evaluation sections, the models evaluated are by default trained without the synthetic data unless they are explicitly mentioned.

We use the most popular GEC model architecture – Transformer (big) model (Vaswani et al., 2017) as our baseline model which has a 6-layer encoder and 6-layer decoder with 1,024 hidden units. We train the English GEC model using an encoder-decoder shared vocabulary of 32K Byte Pair Encoding (Sennrich et al., 2016) tokens and train the Chinese GEC model with 8.4K Chinese characters. We include more training details in the supplementary notes. For inference, we use greedy decoding[6] by default.

All the efficiency evaluations are conducted in the online inference setting (i.e., batch size=1) as we focus on instantaneous GEC. We perform model inference with fairseq[7] implementation using Pytorch 1.5.1 with 1 Nvidia Tesla V100-PCIe of 16GB GPU memory under CUDA 10.2.

4.2 Evaluation for Aggressive Decoding

We evaluate aggressive decoding in our validation set (CoNLL-13) which contains 1,381 validation examples. As shown in Table 1, aggressive decoding achieves a $7\times \sim 8\times$ speedup over the original autoregressive beam search (beam=5), and generates exactly the same predictions as greedy decoding, as discussed in Section 3.1.2. Since greedy decoding can achieve comparable overall performance (i.e., $F_{0.5}$) with beam search while it tends to make more edits resulting in higher recall but lower precision, the advantage of aggressive decoding in practical GEC applications is obvious given its strong performance and superior efficiency.

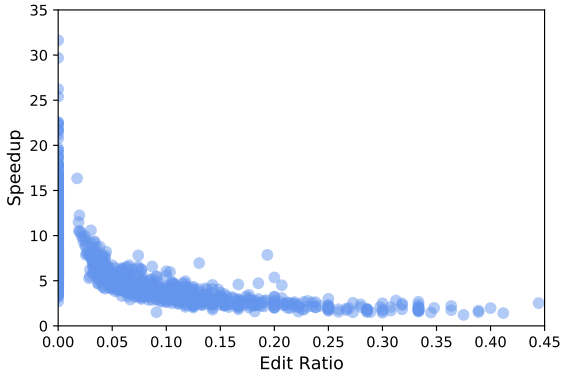

Figure 2: The speedup (over greedy decoding) distribution of all the 1,381 validation examples with respect to their edit ratio in CoNLL-13.

We further look into the efficiency improvement by aggressive decoding. Figure 2 shows the speedup distribution of the 1,381 examples in CoNLL-13 with respect to their edit ratio which is defined as the normalized (by the input length) edit distance between the input and output. It is obvious that the sentences with fewer edits tend to achieve higher speedup, which is consistent with our intuition that most tokens in such sentences can be decoded in parallel through aggressive decoding; on the other hand, for the sentences that are heavily edited, their speedup is limited because of frequent re-decoding. To give a more intuitive analysis, we also present concrete examples with various speedup in our validation set to understand how aggressive decoding improves the inference efficiency in Table 2.

Moreover, we conduct an ablation study to in-

[6]Our implementation of greedy decoding is simplified for higher efficiency ($1.3\times \sim 1.4\times$ speedup over beam=5) than the implementation of beam=1 decoding in fairseq (around $1.1\times$ speedup over beam=5).

[7]https://github.com/pytorch/fairseq

Speedup	Edit Ratio	Input	Output
16.7×	0	Personally , I think surveillance technology such as RFID (radio-frequency identification) should not be used to track people , for the benefit it brings to me can not match the concerns it causes .	[Personally , I think surveillance technology such as RFID (radio-frequency identification) should not be used to track people , for the benefit it brings to me can not match the concerns it causes .]$_0$
5.8×	0	Nowadays , people use the all-purpose smart phone for communicating .	[Nowadays , people use the all-purpose smart phone for communicating .]$_0$
6.8×	0.03	Because that the birth rate is reduced while the death rate is also reduced , the percentage of the elderly is increased while that of the youth is decreased .	[Because the]$_0$ [birth]$_1$ [rate is reduced while the death rate is also reduced , the percentage of the elderly is increased while that of the youth is decreased .]$_2$
5.1×	0.06	More importantly , they can share their ideas of how to keep healthy through Internet , to make more interested people get involve and find ways to make life longer and more wonderful .	[More importantly , they can share their ideas of how to keep healthy through the]$_0$ [Internet]$_1$ [, to make more interested people get involved]$_2$ [and]$_3$ [find]$_4$ [ways to make life longer and more wonderful .]$_5$
3.5×	0.13	As a result , people have more time to enjoy advantage of modern life .	[As a result , people have more time to enjoy the]$_0$ [advantages]$_1$ [of]$_2$ [modern life .]$_3$
1.5×	0.27	Nowadays , technology is more advance than the past time .	[Nowadays , technology is more advanced]$_0$ [than]$_1$ [in]$_2$ [the]$_3$ [past .]$_4$
1.4×	0.41	People are able to predicate some disasters like the earth quake and do the prevention beforehand .	[People are able to predict]$_0$ [disasters]$_1$ [like the earthquake]$_2$ [and]$_3$ [prevent]$_4$ [them]$_5$ [beforehand]$_6$ [.]$_7$

Table 2: Examples of various speedup ratios by aggressive decoding over greedy decoding in CoNLL-13. We show how the examples are decoded in the column of **Output**, where the tokens within a blue block are decoded in parallel through aggressive decoding while the tokens in red blocks are decoded through the original autoregressive greedy decoding.

L_{max}	Total Latency (s)	Speedup
1 (Baseline)	328	1.0×
2	208	1.6×
3	148	2.2×
5	109	3.0×
10	75	4.4×
20	64	5.1×
40	**54**	**6.1×**
Unlimited	**54**	**6.1×**

Table 3: The ablation study of the effect of constraining the maximal aggressive decoding length L_{max} on the online inference efficiency in CoNLL-13. Note that in CoNLL-13, the average length of an example is 21 and 96% examples are shorter than 40 tokens.

Model (Enc+Dec)	CoNLL-13 $F_{0.5}$	Total Latency	Speedup
6+6	38.36	328	1.0×
3+6	36.26	314	1.0×
9+6	38.82	345	1.0×
6+3	37.95	175	1.9×
6+9	38.02	457	0.7×
7+5	38.49	271	1.2×
8+4	38.63	240	1.4×
9+3	**38.88**	181	1.8×
10+2	38.21	137	2.4×
11+1	38.15	**86**	**3.8×**

Table 4: The performance and efficiency of the Transformer with different encoder and decoder depths in CoNLL-13, where 6+6 is the original Transformer-big model that has a 6-layer encoder and a 6-layer decoder.

vestigate whether it is necessary to constrain the maximal aggressive decoding length[8], because it might become highly risky to waste large amounts of computation because of potential re-decoding for a number of steps after the bifurcation if we aggressively decode a very long sequence in parallel. Table 3 shows the online inference efficiency with different maximal aggressive decoding lengths. It appears that constraining the maximal aggressive decoding length does not help improve the efficiency; instead, it slows down the inference if the maximal aggressive decoding length is set to a small number. We think the reason is that sentences in GEC datasets are rarely too long. For example, the average length of the sentences in CoNLL-13 is 21 and 96% of them are shorter than 40 tokens. Therefore, it is unnecessary to constrain the maximal aggressive decoding length in GEC.

[8]Constraining the maximal aggressive decoding length to L_{max} means that the model can only aggressively decode at most L_{max} tokens in parallel.

Model	Synthetic Data	Multi-stage Fine-tuning	CoNLL-14			
			P	R	$F_{0.5}$	Speedup
Transformer-big (beam=5)	No	No	60.2	32.1	**51.2**	1.0×
Levenshtein Transformer* (Gu et al., 2019)	No	No	53.1	23.6	42.5	2.9×
LaserTagger* (Malmi et al., 2019)	No	No	50.9	26.9	43.2	**29.6×**
Span Correction* (Chen et al., 2020)	No	No	**66.0**	24.7	49.5	2.6×
Our approach (9+3)	No	No	58.8	**33.1**	50.9	**10.5×**
Transformer-big (beam=5)	Yes	No	73.0	38.1	61.6	1.0×
PIE* (Awasthi et al., 2019)	Yes	No	66.1	**43.0**	59.7	**10.3×**
Span Correction* (Chen et al., 2020)	Yes	No	72.6	37.2	61.0	2.6×
Our approach (9+3)	Yes	No	**73.3**	41.3	**63.5**	**10.3×**
Seq2Edits (Stahlberg and Kumar, 2020)	Yes	Yes	63.0	45.6	58.6	-
GECToR(*RoBERTa*) (Omelianchuk et al., 2020)	Yes	Yes	73.9	41.5	64.0	**12.4×**
GECToR(*XLNet*) (Omelianchuk et al., 2020)	Yes	Yes	**77.5**	40.1	65.3	-
Our approach (12+2 BART-Init)	Yes	Yes	71.0	**52.8**	**66.4**	9.6×

Table 5: The performance and online inference efficiency evaluation of efficient GEC models in CoNLL-14. For the models with $\star$, their performance and speedup numbers are from Chen et al. (2020) who evaluate the online efficiency in the same runtime setting (e.g., GPU and runtime libraries) with ours. The underlines indicate the speedup numbers of the models are evaluated with Tensorflow based on their released codes, which are not strictly comparable here. Note that for *GECToR*, we re-implement its inference process of *GECToR* (RoBERTa) using fairseq for testing its speedup in our setting. - means the speedup cannot be tested in our runtime environment because the model has not been released or not implemented in fairseq.

4.3 Evaluation for Shallow Decoder

We study the effects of changing the number of encoder and decoder layers in the Transformer-big on both the performance and the online inference efficiency. By comparing 6+6 with 3+6 and 9+6 in Table 4, we observe the performance improves as the encoder becomes deeper, demonstrating the importance of the encoder in GEC. In contrast, by comparing the 6+6 with 6+3 and 6+9, we do not see a substantial fluctuation in the performance, indicating no necessity of a deep decoder. Moreover, it is observed that a deeper encoder does not significantly slow down the inference but a shallow decoder can greatly improve the inference efficiency. This is because Transformer encoders can be parallelized efficiently on GPUs, whereas Transformer decoders are auto-regressive and hence the number of layers greatly affects decoding speed, as discussed in Section 3.2. These observations motivate us to make the encoder deeper and the decoder shallower.

As shown in the bottom group of Table 4, we try different combinations of the number of encoder and decoder layers given approximately the same parameterization budget as the Transformer-big. It is interesting to observe that 7+5, 8+4 and 9+3 achieve the comparable and even better performance than the Transformer-big baseline with much less computational cost. When we further increase the encoder layer and decrease the decoder layer, we see a drop in the performance of 10+2 and 11+1 despite the improved efficiency because it becomes difficult to train the Transformer with extremely imbalanced encoder and decoder well, as indicated[9] by the previous work (Kasai et al., 2020; Li et al., 2021; Gu and Kong, 2020).

Since the 9+3 model achieves the best result with an around 2× speedup in the validation set with almost the same parameterization budget, we choose it as the model architecture to combine with aggressive decoding for final evaluation.

4.4 Results

We evaluate our final approach – shallow aggressive decoding which combines aggressive decoding with the shallow decoder. Table 5 shows the performance and efficiency of our approach and recently proposed efficient GEC models that are all faster than the Transformer-big baseline in CoNLL-14 test set. Our approach (the 9+3 model with aggressive decoding) that is pretrained with synthetic data achieves 63.5 $F_{0.5}$ with 10.3× speedup over the Transformer-big baseline, which outperforms the majority[10] of the efficient GEC models in terms of either quality or speed. The only model that shows advantages over our 9+3 model is *GECToR* which is developed based on the powerful pretrained mod-

[9]They show that sequence-level knowledge distillation (KD) may benefit training the extremely imbalanced Transformer in NMT. However, we do not conduct KD for fair comparison to other GEC models in previous work.

[10]It is notable that *PIE* is not strictly comparable here because their training data is different from ours: *PIE* does not use the W&I+LOCNESS corpus.

Model	NLPCC-18			
	P	R	$F_{0.5}$	Speedup
Transformer-big (beam=5)	36.0	17.2	**29.6**	1.0×
Levenshtein Transformer⋆	24.9	15.0	22.0	3.1×
LaserTagger⋆	25.6	10.5	19.9	**38.0×**
Span Correction⋆	**37.3**	14.5	28.4	2.7×
Our approach (9+3)	33.0	**20.5**	29.4	**12.0×**

Table 6: The performance and online inference efficiency evaluation for the language-independent efficient GEC models in the NLPCC-18 Chinese GEC benchmark.

els (e.g., RoBERTa (Liu et al., 2019) and XLNet (Yang et al., 2019)) with its multi-stage training strategy. Following *GECToR*'s recipe, we leverage the pretrained model BART (Lewis et al., 2019) to initialize a 12+2 model which proves to work well in NMT (Li et al., 2021) despite more parameters, and apply the multi-stage fine-tuning strategy used in Stahlberg and Kumar (2020). The final single model[11] with aggressive decoding achieves the state-of-the-art result – 66.4 $F_{0.5}$ in the CoNLL-14 test set with a 9.6× speedup over the Transformer-big baseline.

Unlike *GECToR* and *PIE* that are difficult to adapt to other languages despite their competitive speed because they are specially designed for English GEC with many manually designed language-specific operations like the transformation of verb forms (e.g., VBD→VBZ) and prepositions (e.g., in→at), our approach is data-driven without depending on language-specific features, and thus can be easily adapted to other languages (e.g., Chinese). As shown in Table 6, our approach consistently performs well in Chinese GEC, showing an around 12.0× online inference speedup over the Transformer-big baseline with comparable performance.

5 Related Work

The state-of-the-art of GEC has been significantly advanced owing to the tremendous success of seq2seq learning (Sutskever et al., 2014) and the Transformer (Vaswani et al., 2017). Most recent work on GEC focuses on improving the performance of the Transformer-based GEC models. However, except for the approaches that add synthetic erroneous data for pretraining (Ge et al., 2018a; Grundkiewicz et al., 2019; Zhang et al., 2019; Lichtarge et al., 2019; Zhou et al., 2020; Wan et al., 2020), most methods that improve performance (Ge et al., 2018b; Kaneko et al., 2020) introduce additional computational cost and thus slow down inference despite the performance improvement.

To make the Transformer-based GEC model more efficient during inference for practical application scenarios, some recent studies have started exploring the approaches based on edit operations. Among them, PIE (Awasthi et al., 2019) and GECToR (Omelianchuk et al., 2020) propose to accelerate the inference by simplifying GEC from sequence generation to iterative edit operation tagging. However, as they rely on many language-dependent edit operations such as the conversion of singular nouns to plurals, it is difficult for them to adapt to other languages. LaserTagger (Malmi et al., 2019) uses the similar method but it is data-driven and language-independent by learning operations from training data. However, its performance is not so desirable as its seq2seq counterpart despite its high efficiency. The only two previous efficient approaches that are both language-independent and good-performing are Stahlberg and Kumar (2020) which uses span-based edit operations to correct sentences to save the time for copying unchanged tokens, and Chen et al. (2020) which first identifies incorrect spans with a tagging model then only corrects these spans with a generator. However, all the approaches have to extract edit operations and even conduct token alignment in advance from the error-corrected sentence pairs for training the model. In contrast, our proposed shallow aggressive decoding tries to accelerate the model inference through parallel autoregressive decoding which is related to some previous work (Ghazvininejad et al., 2019; Stern et al., 2018) in neural machine translation (NMT), and the imbalanced encoder-decoder architecture which

[11] The same model checkpoint also achieves the state-of-the-art result – 72.9 $F_{0.5}$ with a 9.3× speedup in the BEA-19 test set.

is recently explored by Kasai et al. (2020) and Li et al. (2021) for NMT. Not only is our approach language-independent, efficient and guarantees that its predictions are exactly the same with greedy decoding, but also does not need to change the way of training, making it much easier to train without so complicated data preparation as in the edit operation based approaches.

6 Conclusion and Future Work

In this paper, we propose Shallow Aggressive Decoding (SAD) to accelerate online inference efficiency of the Transformer for instantaneous GEC. Aggressive decoding can yield the same prediction quality as autoregressive greedy decoding but with much less latency. Its combination with the Transformer with a shallow decoder can achieve state-of-the-art performance with a $9\times \sim 12\times$ online inference speedup over the Transformer-big baseline for GEC.

Based on the preliminary study of SAD in GEC, we plan to further explore the technique for accelerating the Transformer for other sentence rewriting tasks, where the input is similar to the output, such as style transfer and text simplification. We believe SAD is promising to become a general acceleration methodology for writing intelligence models in modern writing assistant applications that require fast online inference.

Acknowledgments

We thank all the reviewers for their valuable comments to improve our paper. We thank Xingxing Zhang, Xun Wang and Si-Qing Chen for their insightful discussions and suggestions. The work is supported by National Natural Science Foundation of China under Grant No.62036001. The corresponding author of this paper is Houfeng Wang.

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 4251–4261.

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.

Mengyun Chen, Tao Ge, Xingxing Zhang, Furu Wei, and Ming Zhou. 2020. Improving the efficiency of grammatical error correction with erroneous span detection and correction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7162–7169.

Daniel Dahlmeier and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. Building a large annotated corpus of learner english: The nus corpus of learner english. In *Proceedings of the eighth workshop on innovative use of NLP for building educational applications*, pages 22–31.

Tao Ge, Furu Wei, and Ming Zhou. 2018a. Fluency boost learning and inference for neural grammatical error correction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1055–1065, Melbourne, Australia. Association for Computational Linguistics.

Tao Ge, Furu Wei, and Ming Zhou. 2018b. Reaching human-level performance in automatic grammatical error correction: An empirical study. *arXiv preprint arXiv:1807.01270*.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. 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 6114–6123.

Sylviane Granger. *The computer learner corpus: a versatile new source of data for SLA research*.

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.

Jiatao Gu and Xiang Kong. 2020. Fully non-autoregressive neural machine translation: Tricks of the trade. *arXiv preprint arXiv:2012.15833*.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, pages 11181–11191.

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.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *arXiv preprint arXiv:2006.10369*.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021. An efficient transformer decoder with compressed sub-layers. *arXiv preprint arXiv:2101.00542*.

Jared Lichtarge, Chris Alberti, Shankar Kumar, Noam Shazeer, Niki Parmar, and Simon Tong. 2019. Corpora generation for grammatical error correction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3291–3301.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. 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 5057–5068.

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.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The conll-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The CoNLL-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–12.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. Gector–grammatical error correction: Tag, not rewrite. *arXiv preprint arXiv:2005.12592*.

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.

Felix Stahlberg and Shankar Kumar. 2020. Seq2edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5147–5159.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *NeurIPS*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Zhaohong Wan, Xiaojun Wan, and Wenguang Wang. 2020. Improving grammatical error correction with data augmentation by editing latent representation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2202–2212.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading esol texts. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 180–189.

Yi Zhang, Tao Ge, Furu Wei, Ming Zhou, and Xu Sun. 2019. Sequence-to-sequence pre-training with data augmentation for sentence rewriting. *arXiv preprint arXiv:1909.06002*.

Yuanyuan Zhao, Nan Jiang, Weiwei Sun, and Xiaojun Wan. 2018. Overview of the nlpcc 2018 shared task: Grammatical error correction. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 439–445. Springer.

Wangchunshu Zhou, Tao Ge, Chang Mu, Ke Xu, Furu Wei, and Ming Zhou. 2020. Improving grammatical error correction with machine translation pairs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 318–328.

A Hyper-parameters

Hyper-parameters of training the Transformer for English GEC are listed in table 7. The hyper-parameters for Chinese GEC are the same with those of training from scratch.

Configurations	Values
Train From Scratch	
Model Architecture	Transformer (big) (Vaswani et al., 2017)
Number of epochs	60
Devices	4 Nvidia V100 GPU
Max tokens per GPU	5120
Update Frequency	4
Optimizer	Adam (β_1=0.9, β_2=0.98, ϵ=1×10^{-8}) (Kingma and Ba, 2014)
Learning rate	[3×10^{-4} , 5×10^{-4}]
Learning rate scheduler	inverse sqrt
Warmup	4000
Weight decay	0.0
Loss Function	label smoothed cross entropy (label-smoothing=0.1) (Szegedy et al., 2016)
Dropout	[0.3, 0.4, 0.5]
Pretrain	
Number of epochs	10
Devices	8 Nvidia V100 GPU
Update Frequency	8
Learning rate	3×10^{-4}
Warmup	8000
Dropout	0.3
Fine-tune	
Number of epochs	60
Devices	4 Nvidia V100 GPU
Update Frequency	4
Learning rate	3×10^{-4}
Warmup	4000
Dropout	0.3

Table 7: Hyper-parameters values of training from scratch, pretraining and fine-tuning.

Model (Enc+Dec)	**Thread**	**Beam=5** Speedup	**Greedy** Speedup	**Aggressive** Speedup
6+6	8	1×	1.6×	6.5×
9+3	8	1.5×	2.5×	8.0×
6+6	2	1×	2.1×	6.1×
9+3	2	1.5×	3.1×	7.6×

Table 8: The efficiency of the Transformer with different encoder and decoder depths in CoNLL-13 on CPU with 8 and 2 threads.

B CPU Efficiency

Table 8 shows total latency and speedup of the Transformer with different encoder-decoder depth on an Intel® Xeon® E5-2690 v4 Processor(2.60GHz) with 8 and 2 threads[12], respectively. Our approach achieves a 7× $\sim$ 8× online inference speedup over the Transformer-big baseline on CPU.

[12]We set OMP_NUM_THREADS to 8 or 2.