Towards More Efficient Insertion Transformer with Fractional Positional Encoding

Zhisong Zhang[1]* **Yizhe Zhang**[2]† **Bill Dolan**[3]
[1]Carnegie Mellon University, [2]Apple Inc., [3]Microsoft Research
zhisongz@cs.cmu.edu,yizhe.zhang@hotmail.com,billdol@microsoft.com

Abstract

Auto-regressive neural sequence models have been shown to be effective across text generation tasks. However, their left-to-right decoding order prevents generation from being parallelized. Insertion Transformer (Stern et al., 2019) is an attractive alternative that allows outputting multiple tokens in a single generation step. Nevertheless, due to the incompatibility between absolute positional encoding and insertion-based generation schemes, it needs to refresh the encoding of every token in the generated partial hypothesis at each step, which could be costly. We design a novel reusable positional encoding scheme for Insertion Transformers called Fractional Positional Encoding (FPE), which allows reusing representations calculated in previous steps. Empirical studies on various text generation tasks demonstrate the effectiveness of FPE, which leads to floating-point operation reduction and latency improvements on batched decoding.

1 Introduction

Transformer-based models (Vaswani et al., 2017) have been successfully applied to various text generation tasks (Gong et al., 2019; Wang et al., 2019; Ahmad et al., 2020; Zhang et al., 2020a; Lewis et al., 2020; Brown et al., 2020). Most of these models utilize a fixed left-to-right auto-regressive generation strategy, where the strict factorization means that the model can only generate one token per step. This makes it difficult to parallelize the decoding process, while parallel generation may help to improve decoding efficiency.

Recently, insertion-based sequence-generation models (Stern et al., 2019; Gu et al., 2019a; Welleck et al., 2019) have been developed as attractive alternatives to the auto-regressive ones by allowing flexible generation order. In particular, the Insertion Transformer (Stern et al., 2019), which combines the Transformer architecture and the insertion-based strategy, can match the performance of an auto-regressive model while requiring many fewer decoding steps with parallel generation.

The original Insertion Transformer utilizes absolute positional encoding as in the vanilla auto-regressive transformer. In the vanilla transformer, due to its left-to-right generation scheme, tokens' absolute positions do not change; thus, previous computation can be reused. However, this property no longer holds if insertion is allowed, and the Insertion Transformer re-encodes all previously generated tokens at each decoding step, which brings additional computational overheads.

In this work, we propose a reusable positional encoding scheme called Fractional Positional Encoding (FPE) to accelerate the Insertion Transformer. This scheme dynamically calculates each token's positional representations according to its left and right neighbors at insertion time. In this way, each token's positional representations will not change during the decoding process so that the computation can be reusable in the same way as the vanilla transformer, leading to a reduction of computation for the Insertion Transformer.

We evaluate FPE with a range of text generation tasks, including machine translation, word reordering, summarization as well as an open-ended text completion task. We show that the proposed scheme can reduce floating point operations of the insertion-based model while maintaining comparable performance to the vanilla transformer.

Our implementation is available at https://github.com/zzsfornlp/zgen1/.

2 Insertion Transformer

Insertion Transformer (Stern et al., 2019) generates the target sequences via a series of insertion operations. This provides a flexible scheme that can enable different generation orders as well as parallel generation. We focus on the parallel-generation

---

*Work done during an internship at Microsoft Research.
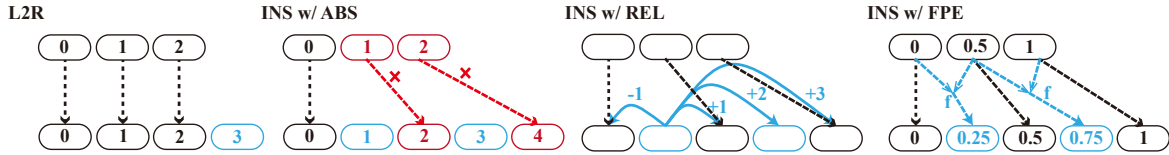†Work done at Microsoft Research.

Figure 1: Illustrations of different positional encoding schemes. Black (or red) nodes denote the tokens that already exist in the previous step, while the newly generated ones are in blue. In ABS, the positional embeddings in previous steps may be non-reusable since absolute positions may change (marked as red). Representations can be made reusable by using REL or our proposed FPE. (Note that the fractional numbers in the FPE figure are only for better illustration, in practice, we adopt embeddings calculated by a learnable linear function.)

variant that inserts multiple tokens at each step. While the left-to-right scheme can only append one token at each step, the insertion-based scheme can add multiple tokens at different slots, thus potentially enabling more efficient generation.

3 Positional Encoding

Figure 1 provides an overview of different positional encoding schemes that we explore. The vanilla left-to-right (**L2R**) Transformer model (Vaswani et al., 2017) adopts a simple absolute positional encoding scheme by assigning left-to-right increasing indexes to each token. This naturally fits the left-to-right generation procedure and allows the previously calculated hidden representations to be reusable. However, in the insertion-based generation, since tokens can be inserted before previously generated tokens, the absolute position of a token may change. Therefore, if still using the absolute positional encoding (**ABS**), the previously calculated hidden layers cannot be reused and the Insertion Transformer needs to re-encode all the existing tokens at each step. This yields computation overhead, which may offset the computation gain from parallelization.

To solve this problem, alternative positional encoding schemes are required. Relative positional encoding (**REL**; Shaw et al., 2018) is an example, which has been adopted for insertion-based models (Lu et al., 2022). Here, each token records its relative positional information at its insertion time. Though this scheme alleviates encoding absolute positions and allows reusing, it requires complex modifications in the attention calculations.

In this work, we design fractional positional encoding (**FPE**), which is a simpler alternative scheme that only modifies the input embeddings. We still give each token a positional embedding $\mathbf{p}$, which is dynamically calculated along the gener-

ation process. Whenever a new token w_{new} is inserted between two existing tokens w_{left} and w_{right}, its positional representations will be calculated with a function f applying to its current left and right neighbors: $\mathbf{p}_{\text{new}} = f(\mathbf{p}_{\text{left}}, \mathbf{p}_{\text{right}})$. In this way, we will have the "fractional"-styled positions. The positional representations of all the tokens will not change throughout the decoding process, and re-encoding is no longer needed.

We specify the FPE representations $\mathbf{p}$ to have the same dimension as the model size and add them to the input embeddings as in the vanilla transformer. We further specify two randomly-initialized embeddings $\mathbf{p}_B$ and $\mathbf{p}_E$ for the beginning- and ending-of sequence tokens, respectively. The function f is modeled by a linear layer[1] which takes the concatenation of the two neighbors' positional embeddings and outputs a new vector of the model size. At training time, all these FPE-related parameters are tuned along with other parameters in the model. The linear layer is lightweight compared to the transformer layers, thus introducing negligible cost.

4 Experiments

4.1 Settings

We explore a variety of generation tasks, including machine translation, word reordering, summarization, as well as an open-ended text completion task. We use WMT14 En-De (Bojar et al., 2014) for machine translation, sentences in WikiText-103 (Wiki103; Merity et al., 2016) for word reordering, XSUM (Narayan et al., 2018) for summarization and paragraphs in Wiki103 for completion. In the text completion task, the model is required to complete each paragraph according to the existing con-

---

[1]We start with the simple linear layer and find it works reasonably well. We also tried some other methods such as adding non-linearity activation but did not find obvious benefits. Therefore, we adopt this simple method.

Task	Model	Evaluation ↑	#Step	#Len	Latency ↓
Translation (WMT14 EN-DE)	L2R	**27.72**	28.4	22.1	230.1
	ABS	27.45	5.7	21.5	100.3
	REL	27.40	5.5	21.5	105.0
	FPE	27.47	5.6	21.4	**97.2**
Text Reordering (Wiki-103)	L2R	**52.82**	27.9	24.8	224.7
	ABS	50.69	6.8	24.2	113.9
	REL	52.63	6.0	24.6	120.7
	FPE	52.52	6.0	24.8	**105.9**
Summarization (XSUM)	L2R	31.33/11.65/25.32	21.3	19.7	206.9
	ABS	**32.09**/11.39/**25.68**	6.7	24.3	114.9
	REL	31.90/**11.66**/25.80	6.2	22.9	125.2
	FPE	31.78/11.57/25.67	6.2	22.7	**114.4**
Text Completion (Wiki-103)	L2R	**3.87/8.48/14.54**	55.6	48.9	468.0
	ABS	1.19/7.66/12.90	9.5	49.4	141.9
	REL	1.69/8.41/13.23	8.1	54.7	161.0
	FPE	1.61/8.26/13.47	8.0	52.3	**129.9**

Table 1: Main results of comparing an auto-regressive left-to-right (L2R) model and three insertion models with absolute (ABS), relative (REL), and fractional (FPE) positional encoding. "Evaluation" denotes automatic evaluation metrics: BLEU for MT and reordering, R-1/R-2/R-L for summarization, and BLEU/METEOR/R-L for completion. "#Step" and "#Len" indicate the average decoding steps and output lengths, respectively. "Latency" denotes the actual average decoding time (ms) per instance with single-instance decoding.
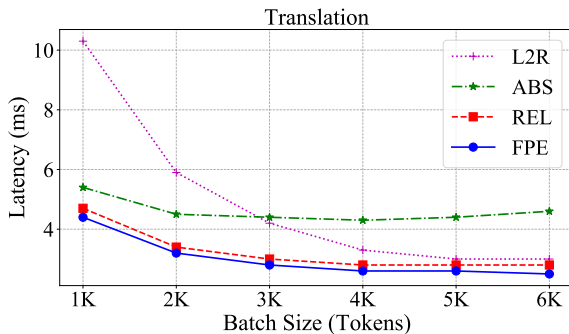

Figure 2: Latency of MT models with different decoding batch sizes (source tokens). Results with single-instance decoding are not shown here since since its latency is much higher.

Batch-size	1K	2K	3K	4K	5K	6K
L2R	10.3	5.9	4.2	3.3	3.0	3.0
ABS	5.4	4.5	4.4	4.3	4.4	4.6
REL	4.7	3.4	3.0	2.8	2.8	2.8
FPE	4.4	3.2	2.8	2.6	2.6	2.5

Table 2: Latency of MT models (milliseconds per instance) with different decoding batch sizes (source tokens). This table shows the detailed numbers corresponding to those in Figure 2.

text. All the decoding experiments are performed with one V100 GPU. Please refer to Appendix A and B for more dataset and experimental details.

4.2 Results

We compare our method (FPE) with the vanilla transformer (L2R), and two other insertion-based models with absolute (ABS) and relative (REL) positional encoding. The main results are shown in Table 1. For automatic performance evaluations, the three insertion-based transformer models (ABS, REL, and FPE) achieve similar results. Compared with L2R, the insertion models' performance is competitive on MT, reordering, and summarization tasks, while being behind on the open-ended text

completion task. This is presumably due to the conditional independence assumption in the parallel generation steps. This issue is beyond the scope of this paper, so we leave it to future work.

For efficiency, insertion-based models can generate target sequences with much fewer decoding steps, leading to latency reduction where the insertion models can achieve around 2x speedups compared to L2R in single-instance mode.

4.3 Batched Decoding

We further explore batched decoding, which is usually adopted to speed up the computation via data parallelism. The latency of MT models against different batch sizes can be found in Figure 2 and Table 2, from which we observe that:

- ABS becomes less efficient when decoding in batches, probably due to the extra computations[2]

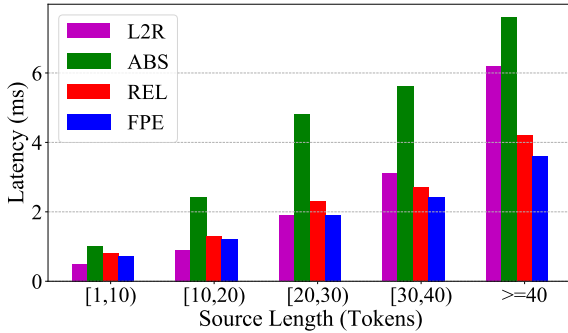[2] We further measure the floating point operations (FLOPs)

Figure 3: Latency breakdowns on source sequence lengths for the MT task with batched decoding (6K).

brought by re-encoding. Though this does not affect its efficiency in the single-instance mode where GPU's computational capacity may not be fulfilled yet, in batched decoding the extra re-encoding computations greatly dampen its efficiency improvements.

- FPE and REL are faster than L2R for relatively small batch sizes. While the batch size becomes larger, the efficiency gain becomes less obvious. Presumably, more complex indexing operations in the insertion-based schemes do not utilize GPUs as fully as L2R. We leave this optimization to future work.

- REL behaves similarly to FPE, but is consistently around 10% slower, probably due to the extra relative positional computations in attentions.

The patterns in other tasks are similar to MT and are shown in Appendix C.

Note that many previous works consider only single-instance or batched decoding mode when measuring efficiency, while we examine both to include a spectrum of real scenarios covering various device memory capacities and querying patterns. While the L2R model and the original Insertion Transformer (w/ ABS) only excel at one end, FPE could help to make the model efficient for both scenarios, potentially benefiting more use cases.

4.4 Length Breakdown

We further perform an ablated speed analysis by breaking down the instances by sequence length. Here, we investigate the task of machine translation and split the instances into different bins according

for decoding an instance following the method of Elbayad et al. (2020) and find that the ABS-based model requires much larger FLOPs than other models. For example, on the MT models, FLOPs per instance is 8.69B for ABS, while FPE only requires 4.65B (REL needs 4.68B).

to source lengths. The breakdown results with a batch size of 6K are shown in Figure 3. The trends are generally similar to the overall results. ABS is not quite efficient for batched decoding and can be even slower than L2R. Though the utilization of FPE does not provide obvious efficiency improvements on short sentences, it brings benefits for generating longer sequences. It can achieve a 1.7x speedup over L2R for sentences that are longer than 40 tokens, even with a large batch size. Similar to the overall trend, FPE is consistently more efficient than REL, yet being simpler.

5 Related work

Generation Order. Previous works have been exploring relaxing the output dependencies and allowing parallel generation. The Non-Autoregressive Transformer (NAT) (Gu et al., 2018) enables the decoder to generate target sequences in one or several decoding steps (Gu et al., 2018; Gu and Kong, 2021; Lee et al., 2018; Gu et al., 2019b; Ghazvininejad et al., 2019). Most of these models require predicting target length and generating multiple consecutive tokens at once. The generation is sometimes not fluent, as multiple tokens may compete for the same meaning. The insertion-based methods (Stern et al., 2019; Gu et al., 2019a; Welleck et al., 2019; Chan et al., 2020; Zhang et al., 2020b) also change the standard left-to-right generation by allowing dynamically inserting tokens for the generation process. This provides a good balance between generation fluency and efficiency, and does not require predicting target lengths first. In this work, we follow this insertion-based generation scheme and further improve its efficiency. In addition to efficiency, allowing flexible generation order is another motivation to study non-L2R generation schemes. A good generation order may bring performance benefits (Ford et al., 2018; Jiang and Bansal, 2021).

Reusable Positional Encoding. In insertion-based models, the absolute positional encoding can be non-reusable since an inserted token will change the absolute positions of its following tokens. Alternative schemes are required to enable reusable encoding. Relative positional encoding (Shaw et al., 2018) is utilized for the insertion-based generation in some recent work (Gu et al., 2019a; Lu et al., 2022) to avoid the re-encoding of the previously generated tokens. However, it requires modifications to the inner attention mech-

anism of the Transformer, while our FPE scheme is a simpler alternative that only modifies the input. In a similar spirit to our scheme, Shiv and Quirk (2019) explore a tree-based positional encoding strategy. Our scheme is different in that in the insertion-based generation, each node has two parent nodes, yielding a graph structure rather than a tree.

6 Conclusion

In this work, we investigate the re-encoding issue that sometimes hinders the Insertion Transformer from receiving its computation gain and propose a Fractional Positional Encoding scheme that is naturally compatible with the insertion-based generation scheme to solve this issue. With experiments on various tasks, we show that this simple scheme eliminates the need of re-encoding the previously generated tokens and obtains a promising balance between efficiency and performance.

Limitations

This work has several limitations. First, we mainly rely upon the architecture and decoding strategy of the Insertion Transformer, which only allows the generation of one token between two neighboring tokens at one step. It would be interesting to consider more flexible generation schemes. It would also be interesting to compare our models with other (semi) non-autoregressive models, which we leave to future work. Moreover, we follow the best-performing binary tree training objective of the Insertion Transformer, which in some way sacrifices the flexibility of the generation order. It would be interesting to explore the application of the proposed positional encoding scheme with more flexible generation orders. It would also be interesting to explore the impacts of using larger pre-trained models and investigate how it interacts with the insertion-based scheme. Finally, on the open-ended generation task, the insertion-based method still performs worse than the left-to-right one, which requires further investigation since the output dependencies would need more careful modeling in the open-ended scenarios.

Broader Impact

This work focuses on improving for the natural language processing (NLP) and general artificial intelligence (AI) research community. Our work can be leveraged to improve natural language generation (NLG) models, including but not limited to text editing, conversational agents, and question answering systems. The **broader impact** and the **risks** of this work are summarized as following:

• This work can facilitate research in the NLG tasks in a generic manner, to potentially accelerate generations in applications like machine translation, text summarization, and virtual assistants.

• This work is a fundamental research work that focuses on technical improvement, thus we have NOT imposed additional aggressive filtering techniques to the text data we used, beyond what has been performed to the original dataset from their sources. The text data we used may have offensiveness/toxicity/fairness/bias issues that we haven't been able to identify, as those are not the main focus of this work.

• Given the above potential risk, due to the nature of natural language generative models, we note that the generations or outputs of this work, though not likely, may reflect gender and other historical biases implicit in the data. Under rare circumstances, the generations may exhibit a mild extent of unethical, biased, or offensive attitudes. These are known issues with current state-of-the-art text generation models. We would hope that a faster generation system as what we present can enable more iterations of further mitigation strategies to inappropriate and hallucinated generations.

• This work aims to advance AI technology in an environmental-friendly manner. Our proposed method can potentially reduce carbon footprints produced by AI models.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4998–5007, Online. Association for Computational Linguistics.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind

Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

William Chan, Mitchell Stern, Jamie Kiros, and Jakob Uszkoreit. 2020. An empirical study of generation order for machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5764–5773, Online. Association for Computational Linguistics.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *International Conference on Learning Representations*.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George Dahl. 2018. The importance of generation order in language modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2942–2946, Brussels, Belgium. Association for Computational Linguistics.

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 6112–6121, Hong Kong, China. Association for Computational Linguistics.

Li Gong, Josep Crego, and Jean Senellart. 2019. Enhanced transformer model for data-to-text generation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 148–156, Hong Kong. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *ICLR*.

Jiatao Gu and Xiang Kong. 2021. Fully non-autoregressive neural machine translation: Tricks of the trade. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 120–133, Online. Association for Computational Linguistics.

Jiatao Gu, Qi Liu, and Kyunghyun Cho. 2019a. Insertion-based decoding with automatically inferred generation order. *Transactions of the Association for Computational Linguistics*, 7:661–676.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019b. Levenshtein transformer. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Yichen Jiang and Mohit Bansal. 2021. Learning and analyzing generation order for undirected sequence models. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3513–3523, Punta Cana, Dominican Republic. 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, Austin, Texas. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

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, Brussels, Belgium. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. 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 3730–3740, Hong Kong, China. Association for Computational Linguistics.

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*.

Sidi Lu, Tao Meng, and Nanyun Peng. 2022. Insnet: An efficient, flexible, and performant insertion-based text generation model. In *Advances in Neural Information Processing Systems*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Vighnesh Shiv and Chris Quirk. 2019. Novel positional encodings to enable tree-based transformers. *Advances in Neural Information Processing Systems*, 32:12081–12091.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

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.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

Sean Welleck, Kianté Brantley, Hal Daumé Iii, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. In *International Conference on Machine Learning*, pages 6716–6726. PMLR.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. 2020a. DIALOGPT : Large-scale generative pre-training for conversational response generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 270–278, Online. Association for Computational Linguistics.

Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. 2020b. POINTER: Constrained progressive text generation via insertion-based generative pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8649–8670, Online. Association for Computational Linguistics.

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

A Dataset details

We provide more details of the datasets utilized in this work:

- **WMT14 (En-De).** For machine translation, we utilize the widely used WMT 2014 English-German translation dataset (Bojar et al., 2014), with newstest2013 as the development and newstest2014 as the test set. Following previous work (Stern et al., 2019; Chan et al., 2020), we apply sequence-level knowledge distillation (Hinton et al., 2015; Kim and Rush, 2016) from a left-to-right autoregressive model, which has been found helpful to reduce data complexity and improve the performance of NAT models (Zhou et al., 2020).

- **Wiki103(S).** For word reordering, we take text sequences from the WikiText-103 dataset (Merity et al., 2016). Here, we focus on the task at the Sentence level and thus perform sentence-splitting and treat each sentence as an individual sequence.

- **XSUM.** For summarization, we utilize the XSUM dataset (Narayan et al., 2018), where the targets are short, one-sentence news summaries for news articles. This task does not favor the extractive strategies and provides a good test bed for abstractive generation-based models. Following previous work (Liu and Lapata, 2019), We truncate the input documents to 512 tokens.

- **Wiki103(P).** For paragraph completion, we again utilize the WikiText-103 dataset, but at the Paragraph level this time. We take paragraphs that contain four to seven sentences. For each paragraph, we take the last two sentences as the target and the previous ones are used as the source inputs.

Table 3 summarizes the statistics of the datasets.

B Experimental settings

We mainly follow the settings of the original Insertion Transformer (Stern et al., 2019). To further encourage generations in fewer steps, we adopt the Binary Tree training loss. For the REL-based insertion model, we include the relative positional encoding by modifying attentions following (Shaw et al., 2018). For other hyper-parameter settings, we mainly follow the common practice. We adopt slightly different settings for constrained and open-ended tasks, which are described in the following.

For the source-constrained tasks (MT, reordering, and summarization), we take the Transformer-base architecture (Vaswani et al., 2017) (6 layers, 8 heads per layer, 512 model dimensions) and the full model contains around 66M parameters. The

Datasets	#Seq.(train/dev/test)	Src-Len	Trg-Len
WMT14	4.0M/3.0K/3.0K	26.1	24.8
Wiki103(S)	1.8M/3.8K/4.1K	25.7	25.7
XSUM	204K/11.3K/11.3K	328.5	23.3
Wiki103(P)	349K/0.8K/0.8K	79.9	50.6

Table 3: Statistics of the datasets utilized in this work. Here, "#Seq." denotes the number of instances (sequences) in each split, "Src-Len" indicates the average number of words in the source, and "Trg-Len" shows the average number of words in the target.

models are trained by the Adam optimizer (Kingma and Ba, 2015), with the same learning rate scheduling scheme of (Vaswani et al., 2017). We train the models for a maximum of 300K steps for machine translation and 100K steps for reordering and summarization. The models are validated on the development set every 1K steps and we average the five checkpoints that obtain the best results to obtain the final model. We take standard evaluation metrics for the corresponding tasks: BLEU[3] (Papineni et al., 2002) for machine translation and word reordering, ROUGE[4] (Lin, 2004) for summarization. Unless otherwise specified, we utilize a beam size of 4 in decoding. Following Stern et al. (2019) and Chan et al. (2020), we select an EOS penalty $\in \{0, 0.5, 1, ..., 5\}$ according to the results on the development set.

For the open-ended paragraph completion task, we adopt similar schemes, but with a difference of employing pre-trained models, which we find helpful in preliminary experiments. Due to limitation of computational resources, we adopt a relatively small model: the distilled version[5] (Sanh et al., 2019) of RoBERTa (Liu et al., 2019) (6 layers, 12 heads per layer, 768 model dimensions). The full model contains around 140M parameters. We adopt similar training schemes to the constrained cases and the models are trained for 300K steps. Since there are no reliable automatic evaluation metrics for this task, we simply average the final five checkpoints as the final model. For the open-ended task, we find that using greedy or beam search sometimes leads to outputs with severe repetition problems, we thus apply sampling, specif-

---

[3] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl
[4] https://github.com/google-research/google-research/tree/master/rouge
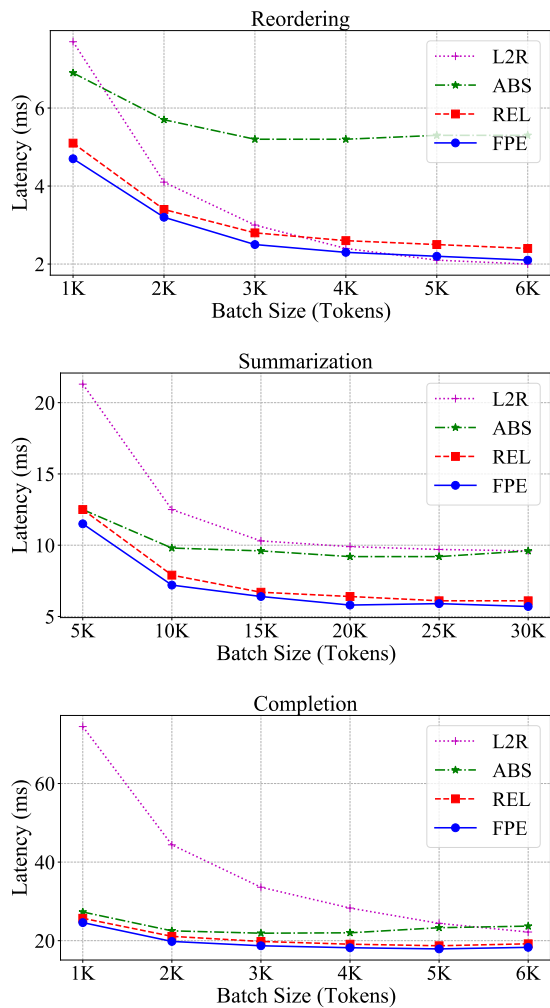[5] https://huggingface.co/distilroberta-base

Figure 4: Latency of the models with different decoding batch sizes (source tokens) for more tasks.

ically, nucleus sampling with p=0.95 (Holtzman et al., 2020) in decoding.

All the models are trained with four V100 GPUs and tested with one V100 GPU. The training takes one to three days depending on the tasks.

C Additional Results

In Figure 4, we further show the batched-decoding latency of different models on more tasks. The patterns are generally similar to those in Figure 2 of the MT task.