Tree-Structured Non-Autoregressive Decoding for Sequence-to-Sequence Text Generation

Pengyu Ji[‡§ *] **Yufei Liu**[‡§ *] **Xiang Hu**[¶] **Kewei Tu**[‡§ †]
[‡]School of Information Science and Technology, ShanghaiTech University
[§]Shanghai Engineering Research Center of Intelligent Vision and Imaging
{jipy2023, liuyf22022, tukw}@shanghaitech.edu.cn
[¶]Ant Group
aaron.hx@antgroup.com

Abstract

Autoregressive Transformer (AT) dominates sequence-to-sequence generation tasks but suffers from high inference latency due to sequential token generation. Non-Autoregressive Transformer (NAT) improves inference efficiency by parallelizing token prediction, yet degrades generation quality. To address these limitations, we propose Tree-structured Non-Autoregressive Decoding (TNAD), a novel paradigm that bridges autoregressive and non-autoregressive decoding. TNAD generates a sentence through a top-down, layer-wise expansion of its constituency parse tree, enabling parallel generation within each layer while preserving contextual dependencies across layers. Experimental results on machine translation and paraphrase generation demonstrate that TNAD outperforms AT in efficiency and NAT in generation quality, thus offering a new alternative to AT and NAT in the trade-off between efficiency and quality. Our code is publicly available at https://github.com/jipy0222/TNAD.

1 Introduction

Autoregressive Transformer (AT) has shown strong performance in both language modeling (Radford et al., 2019; Achiam et al., 2023) and sequence-to-sequence (seq2seq) tasks (Vaswani et al., 2017; Lewis et al., 2019) thanks to its autoregressive decoding paradigm. However, its reliance on sequential token generation introduces significant inference latency bottlenecks. To tackle the inefficiency problem, Non-Autoregressive Transformer (NAT) (Gu et al., 2017; Kasai et al., 2020) has been proposed as a counterpart that generates all tokens simultaneously. Though achieving substantial inference speedup, the non-autoregressive decoding paradigm neglects the contextual dependencies among generated tokens, resulting in the *multi-modality problem* (i.e., mixing parts from different valid output sequences, thus lacking grammatical coherence) (Gu et al., 2017) and suffering from notable degradation in generation quality. A balanced generation paradigm that bridges autoregressive and non-autoregressive decoding and reconciles the trade-offs in generation quality and efficiency remains underexplored.

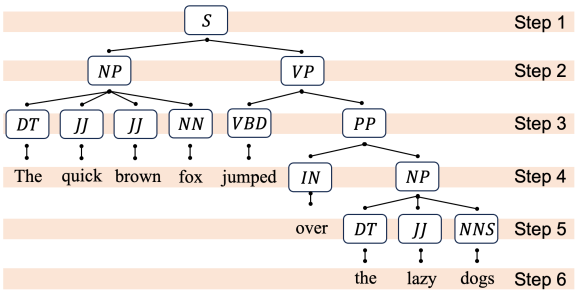

Figure 1: An illustration of the generation process. In each step, TNAD expands one layer of the constituency parse tree in a non-autoregressive way.

In this paper, we propose Tree-structured Non-Autoregressive Decoding (TNAD) as a new text generation paradigm, which generates a sentence by top-down layer-wise expansion of its constituency parse tree, with all elements within a layer generated in parallel, as illustrated in Fig. 1. It theoretically relieves the inference latency burden of AT by reducing inference steps from $\mathcal{O}(n)$ (n for sentence length) to $\mathcal{O}(\log n)$ ($\log n$ for average tree height). Moreover, this paradigm alleviates the *multi-modality problem* encountered by NAT because preceding tree layers can be observed for later generations, creating conditional dependencies absent in NAT. The experimental results on machine translation and paraphrase generation show that our proposed paradigm surpasses NAT in generation quality and AT in generation efficiency. In contrast to conventional paradigms that prioritize either quality or efficiency, TNAD strikes a new balance between the two and offers a novel alternative to AT and NAT.

[*] Equal contribution.
[†] Corresponding authors.

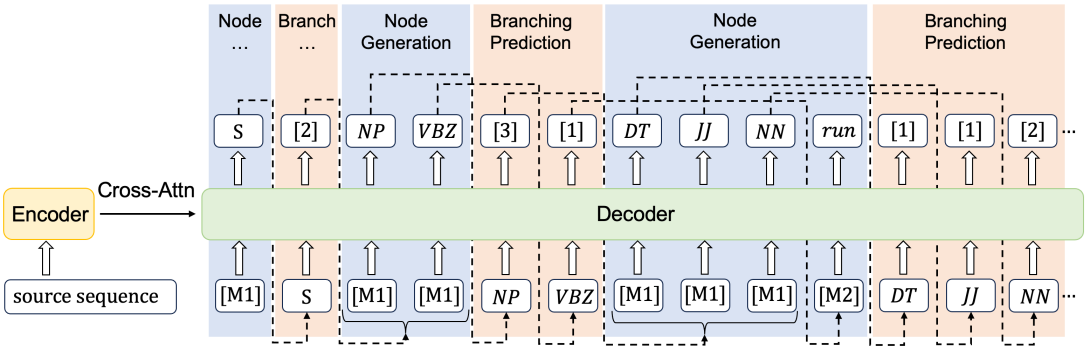

Figure 2: Model architecture. In this example, *NP, VBZ, DT, JJ, NN* are non-terminals, *run* is a token, and [M1] and [M2] stand for the non-terminal placeholder and the token placeholder.

2 Method

In this section, we first introduce the top-down layer-wise generation process and then how to implement the process with the Transformer model.

2.1 Generation Process

TNAD generates a sentence by iteratively expanding layers of its constituency parse tree from the top down, as shown in Fig. 1. Each layer is expanded in one decoding step consisting of two consecutive operations:

- Branching Prediction. Given the previous layer of the tree, we simultaneously predict the numbers of child nodes (i.e., branching factors) for all non-terminals within it. These child nodes constitute the current layer. If a non-terminal is predicted to have only one child node, we identify the non-terminal as a pre-terminal and its child node as a token; otherwise, we identify the child nodes as non-terminals.
- Node Generation. Given the exact number of nodes and their identified types in the current layer, we simultaneously predict the labels of all the nodes (i.e., specific non-terminals or tokens) according to their types.

Since the first layer contains only a single root node, we skip the Branching Prediction of the first layer and only perform Node Generation. After that, we iteratively execute decoding steps by performing Branching Prediction and Node Generation until no non-terminal is left to be expanded, obtaining a complete constituency parse tree whose leaf nodes constitute the generated sentence.

2.2 Model

We employ the architecture of `Bart` (Lewis et al., 2019), a representative encoder-decoder Transformer architecture (Vaswani et al., 2017), for sequence-to-sequence tasks, with a shared decoder for both Branching Prediction and Node Generation, as shown in Fig. 2. Note that we do not reuse the pretrained `Bart` parameters. Instead, we train the whole model from scratch.

Recall that NAT first predicts the number of target tokens k, and then inputs k MASK tokens to a transformer to predict output tokens in parallel. Here, we do Branching Prediction and Node Generation in a similar non-autoregressive way. When performing Branching Prediction of a layer, the decoder takes all the non-terminal labels from the previous layer as input and predicts their branching factors in parallel. We cast branching factor prediction as multi-class classification by setting a maximum branching factor. When performing Node Generation of a layer, the decoder takes node placeholders as input, whose number and types are determined by the preceding Branching Prediction. Specifically, if a predicted branching factor is 1, we input a token placeholder [M2]; otherwise, a corresponding number of non-terminal placeholders [M1] are input. The decoder then predicts non-terminal labels for [M1] placeholders and tokens for [M2] placeholders in parallel.

We also modify attention scoring and masking in the decoder transformer to incorporate tree structure information as follows:

- Layer-wise causal attention mask. On top of the standard causal mask, we allow full attention within the same Branching Prediction or Node Generation block. In other words, a position can attend to everything to its left and those positions to its right that fall into the same block. From the perspective of the tree structure, a node can see everything in the layer it belongs to as well as all the preceding layers.

Models	Machine Translation								
	IWSLT14 De-En			IWSLT14 En-De			WMT16 Ro-En		
	BLEU	Speedup	Iter	BLEU	Speedup	Iter	BLEU	Speedup	Iter
AT	35.64	1.0×	23.25	27.37	1.0×	23.98	33.33	1.0×	28.05
Vanilla-NAT	21.71	7.48×	1	13.07	8.50×	1	24.53	9.35×	1
TNAD	33.53	1.24×	18.31	23.37	2.19×	11.62	32.61	1.35×	20.18
For Reference (Orthogonal to TNAD)									
DAT	32.99	5.71×	1	23.14	5.68×	1	32.25	6.13×	1

Table 1: Results on machine translation. 'Iter' means the number of decoding iterations. The speedup is evaluated on the test set with a batch size of 1.

- Node-distance-based linear attention bias. We define the distance between two nodes in a tree structure as the length of the (only) path connecting them. We employ the same distance computation for non-terminal inputs and placeholder inputs. Following ALiBi (Press et al., 2021), we bias attention scores between two nodes with a penalty proportional to their distance.

We use learnable absolute position embeddings as in `Bart` for both non-terminal and placeholder inputs based on their absolute positions in the input sequence. During training, the model is optimized using the standard cross-entropy loss, with gold output labels derived from ground-truth parse trees.

3 Experiment

Following Li et al. (2023), we perform experiments on two seq2seq tasks: machine translation and paraphrase generation. We compare our model, TNAD, primarily with AT and vanilla-NAT. Note that there exist several approaches to improving the performance of NAT, including training-based (Bao et al., 2022; Zhang et al., 2022) and model-based (Huang et al., 2022; Gui et al., 2023) methods, but these approaches are orthogonal to and can be combined with TNAD. A further discussion is provided in Section 4. Therefore, here we only include the performance of DA-Transformer (DAT) (Huang et al., 2022), a representative improved NAT model, for reference.

3.1 Experiment Setup

Datasets. For machine translation, we conduct experiments on IWSLT14 German-English (De-En), IWSLT14 English-German (En-De), and WMT16 Romanian-English (Ro-En). For the IWSLT datasets, we follow the scripts provided by `fairseq` (Ott et al., 2019) to do preprocessing. For the WMT dataset, we use the same preprocessed data and train/dev/test splits as in Lee et al. (2018). We keep the raw version of these machine translation datasets instead of utilizing knowledge distillation techniques. For paraphrase generation, we use ParaNMT-Small (Chen et al., 2019). These datasets are all encoded into subword units by BPE (Sennrich et al., 2015). We use Berkeley Parser (Kitaev and Klein, 2018; Kitaev et al., 2019) to obtain silver constituency parse trees with an optional postprocessing step for all datasets. More dataset details are listed in Appendix A.

Implementation Details. We employ `Bart` as the model architecture. Following Gu et al. (2017) and Vaswani et al. (2017), we adopt different model configurations for different datasets. We follow Gui et al. (2023) and Li et al. (2023) for the training setup. When doing inference, we choose greedy decoding as the decoding strategy for both our model and baselines. We implement our code and conduct experiments on the `transformers` framework by HuggingFace[1]. More implementation details can be referred to in Appendix B.

Evaluation. In terms of generation quality, we adopt the BLEU score (Papineni et al., 2002) as the main evaluation metric for machine translation tasks. The ROUGE score (Lin, 2004) is additionally provided as a reference for paraphrase generation. For all the datasets, we pick the best 3 checkpoints based on the validation BLEU score and average their test set performance. For generation efficiency, we measure the number of decoding iterations (Iter) and inference speedup over AT. Iter is the sequence length for AT and is 1 for Vanilla-NAT and DAT. For TNAD, Iter is twice the height of the generated parse tree because generating each layer of the tree requires two iterations, one for Branching Prediction and the other for Node Generation. The speedup is evaluated on the test set with a batch size of 1 on a single A100 GPU.

[1] https://github.com/huggingface/transformers

Models	Paraphrase Generation			
	BLEU	ROUGE-1/2/L/avg	Speedup	Iter
AT	16.3	52.1 / 27.3 / 47.7 / 42.4	1.0×	12.1
Vanilla-NAT	10.6	47.8 / 20.7 / 43.0 / 37.2	4.2×	1
TNAD	12.3	48.0 / 22.4 / 44.5 / 38.3	1.2×	7.9
For Reference (Orthogonal to TNAD)				
DAT	12.1	47.5 / 23.2 / 43.2 / 38.0	2.6×	1

Table 2: Results on paraphrase generation.

Models	IWSLT14	
	De-En	En-De
Trivial Tree	20.35	12.25
No Label	30.72	20.79
No ALiBi	32.16	22.26
TNAD	33.53	23.37

Table 3: Ablation study results, showing BLEU scores on the IWSLT14 De-En and En-De datasets.

3.2 Results

Machine Translation. We report results on machine translation tasks in Table 1. TNAD outperforms Vanilla-NAT and even the reference baseline DAT in generation quality in all three datasets. In terms of generation efficiency, TNAD surpasses the AT baseline consistently. We also observe that both the speedup and generation quality of TNAD are correlated with decoding iterations. For the IWSLT14 En-De dataset, the parse trees are flatter, and hence TNAD achieves the maximum speedup over AT. On the other hand, for the IWSLT14 De-En and WMT16 Ro-En datasets, the decoding iterations are larger because of deeper parse trees, and TNAD can be seen to only slightly underperform AT in terms of the BLEU score.

Paraphrase. Table 2 shows the results on paraphrase generation. TNAD surpasses Vanilla-NAT and DAT in the BLEU score and additional ROUGE scores. For generation efficiency, because AT requires fewer decoding iterations than in machine translation due to shorter target sentences, the inference speedup for Vanilla-NAT, DAT, and TNAD is smaller in comparison with the machine translation results. Nonetheless, TNAD still achieves positive speedup over AT.

3.3 Ablations

We perform ablation experiments on TNAD as shown in Table 3. Trivial Tree refers to TNAD trained with balanced binary trees (with a dummy non-terminal label on all non-leaf nodes) instead of silver trees. No Label denotes replacing all the non-terminal labels with a dummy label in TNAD. No ALiBi is TNAD without node-distance-based linear attention bias mentioned in section 2.2.

It can be seen that Trivial Tree degrades the most in translation quality, even underperforming Vanilla-NAT. We believe it is because with a balanced binary tree structure, most tokens are predicted simultaneously during Node Generation in the last layer, which is similar to Vanilla-NAT. No Label suffers a clear decline in generation quality as well. The results from the above two baselines suggest that the performance gains of TNAD over Vanilla-NAT stem from its incorporation of reasonable syntactic structures and non-terminal labels, rather than from increased computational costs compared with Vanilla-NAT. Finally, the results of No ALiBi show that our proposed linear attention bias is also critical for enhancing model performance.

4 Related Work

Non-autoregressive Transformers. Gu et al. (2017) proposes non-autoregressive decoding to accelerate text generation at the expense of degraded quality. A series of work (Bao et al., 2022; Kasai et al., 2020; Huang et al., 2022; Gui et al., 2023) has been developed to address the performance discrepancy. Among these, DA-Transformer (Huang et al., 2022) (DAT) shows superior effectiveness and establishes itself as a state-of-the-art non-autoregressive method. DAT first generates a Directed Acyclic Graph (DAG) representing possible tokens and probabilistic transitions between them in a non-autoregressive way, and then finds the most probable token sequence through the DAG. Techniques, including DAT, can be considered direct and superior replacements for vanilla NAT. Recently, diffusion-based methods (Austin et al., 2021; Arriola et al., 2025) provide a new direction for parallelized decoding. In comparison, our work decomposes the generation process into a series of top-down steps aligned with the syntax tree, where each step employs non-autoregressive decoding. So it is straightforward to combine DAT and our TNAD: simply replace each step of TNAD with DAT.

Syntax-Based Generation. One thread of research (Sartran et al., 2022; Murty et al., 2023; Hu et al., 2024; Zhao et al., 2024) focuses on jointly modeling syntactic trees and sentences. They se-

quentially generate not only tokens, but also actions building up a syntactic parse tree. Another thread of research (Welleck et al., 2019; Li et al., 2023) performs hierarchical generation under the guidance of syntax. These studies primarily focus on generation quality instead of efficiency improvements in the generation process. Different from their work, our work exploits the inherent parallelism of hierarchical tree structures to achieve efficiency gains in generation.

5 Conclusion

We propose TNAD, a new text generation paradigm that generates a sentence by iteratively expanding layers of its constituency parse tree from the top down, with all nodes within a layer predicted in parallel by performing Branching Prediction and Node Generation. Experimental results on seq2seq tasks show that TNAD outperforms NAT in generation quality and AT in generation efficiency, which demonstrates TNAD as a balanced option between AT and NAT.

6 Limitations

There are several limitations of this work: (1) TNAD relies on constituency parse trees for training, which are predicted by an external parser in this study. For languages with limited access to high-quality constituency parsers, the advantages of TNAD could diminish. (2) We adopt a shared decoder transformer for Branching Prediction and Node Generation. Intuitively, Branching Prediction is an easier task than Node Generation. We may apply techniques such as layer skipping to Branching Prediction for further efficiency gains. (3) Since TNAD bridges NAT and AT, we conduct experiments on seq2seq tasks, which serve as a common ground where both paradigms are widely applied. TNAD is also suitable for other architectures and tasks, such as the decoder-only transformer architecture for language modeling. We leave it as future work.

Acknowledgements

This work was supported by the robotic AI-Scientist platform of Chinese Academy of Science, and the Core Facility Platform of Computer Science and Communication, SIST, Shanghaitech University.

References

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, and 260 others. 2023. Gpt-4 technical report.

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. *ArXiv*, abs/2503.09573.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. *ArXiv*, abs/2107.03006.

Yu Bao, Hao Zhou, Shujian Huang, Dongqi Wang, Lihua Qian, Xinyu Dai, Jiajun Chen, and Lei Li. 2022. Glat: Glancing at latent variables for parallel text generation. In *Annual Meeting of the Association for Computational Linguistics*.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019. Controllable paraphrase generation with a syntactic exemplar. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5972–5984, Florence, Italy. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Shangtong Gui, Chenze Shao, Zhengrui Ma, Xishan Zhang, Yunji Chen, and Yang Feng. 2023. Non-autoregressive machine translation with probabilistic context-free grammar. *ArXiv*, abs/2311.07941.

Xiang Hu, Pengyu Ji, Qingyang Zhu, Wei Wu, and Kewei Tu. 2024. Generative pretrained structured transformers: Unsupervised syntactic language models at scale. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2640–2657, Bangkok, Thailand. Association for Computational Linguistics.

Fei Huang, Hao Zhou, Yang Liu, Hanguang Li, and Minlie Huang. 2022. Directed acyclic transformer for non-autoregressive machine translation. *ArXiv*, abs/2205.07459.

Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. 2020. Non-autoregressive machine translation with disentangled context transformer. In *International Conference on Machine Learning*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *ArXiv*, abs/1802.06901.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdel rahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*.

Yafu Li, Leyang Cui, Jianhao Yan, Yongjing Yin, Wei Bi, Shuming Shi, and Yue Zhang. 2023. Explicit syntactic guidance for neural text generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14095–14112, Toronto, Canada. Association for Computational Linguistics.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Annual Meeting of the Association for Computational Linguistics*.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D. Manning. 2023. Pushdown layers: Encoding recursive structure in transformer language models. *ArXiv*, abs/2310.19089.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *North American Chapter of the 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 *Annual Meeting of the Association for Computational Linguistics*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *ArXiv*, abs/2108.12409.

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

Laurent Sartran, Samuel Barrett, Adhiguna Kuncoro, Milovs Stanojevi'c, Phil Blunsom, and Chris Dyer. 2022. Transformer grammars: Augmenting transformer language models with syntactic inductive biases at scale. *Transactions of the Association for Computational Linguistics*, 10:1423–1439.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *ArXiv*, abs/1508.07909.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

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

Kexun Zhang, Rui Wang, Xu Tan, Junliang Guo, Yi Ren, Tao Qin, and Tie-Yan Liu. 2022. A study of syntactic multi-modality in non-autoregressive machine translation. In *North American Chapter of the Association for Computational Linguistics*.

Yida Zhao, Chao Lou, and Kewei Tu. 2024. Dependency transformer grammars: Integrating dependency structures into transformer language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1543–1556, Bangkok, Thailand. Association for Computational Linguistics.

A Dataset Details

We provide statistics of the preprocessed datasets in Table 4, including: (1) the number of samples for each dataset split; (2) BPE: the merge operations set for learning a BPE model to do tokenization; (3) Length: the average length of the target sequence in the train set; (4) Height: the average height of the silver parse trees of the target sequences; (5) Max BF: maximum branching factor of the silver parse trees; (6) NT Labels: the number of types of non-terminal labels of the silver parse trees.

For the ParaNMT-Small dataset, we perform a postprocessing step after we obtain the silver parse trees. We first remove a few extreme samples to reduce the maximum branching factor from 325 to 80. Then, we modify the parse tree by restricting its height. Specifically, we apply a depth-first traversal over the tree. Once the number of leaf nodes covered by this visited node is less than some threshold, we directly connect the covered leaf nodes, along with their pre-terminals, to this visited node and ignore the underlying subtree. The threshold we set for ParaNMT-Small is 8.

Dataset	Train/Dev/Test	BPE	Length	Height	Max BF	NT Labels
IWSLT14 De-En	160K/7K/6K	10K	23.65	9.75	53	71
IWSLT14 En-De	160K/7K/6K	10K	24.18	6.36	72	74
WMT16 Ro-En	608K/2K/2K	40K	26.48	11.17	68	72
ParaNMT-Small	493K/0.5K/0.8K	6K	12.22	4.53	80	73

Table 4: Details of datasets used in our experiments. BPE: the merge operations set for learning a BPE model to do tokenization; Length: the average length of target sequence in the train set; Height: the average height of the silver parse trees of the target sequences; Max BF: maximum branching factor of the silver parse trees; NT Labels: the number of types of non-terminal labels of the silver parse trees;

B Implementation Details

Following Gu et al. (2017) and Vaswani et al. (2017), we adopt a small model setting ($d_{model} = 256, d_{hidden} = 1024, n_{layer} = 5, n_{head} = 4$) for the IWSLT14 datasets and a base setting ($d_{model} = 512, d_{hidden} = 2048, n_{layer} = 6, n_{head} = 8$) for the other two datasets. For Vanilla-NAT, the length prediction loss factor is set to 0.1. For DAT, we set $\lambda = 4$ for the graph size.

We train models on 8 A100 GPUs. All models are optimized with Adam (Kingma and Ba, 2014) with $\beta = (0.9, 0.999)$ and are trained for 200K steps, with each batch containing 1024 samples. The learning rate increases to $7 \cdot 10^{-4}$ in the first 10K steps and then anneals exponentially. We set the weight decay as 0.01 and the dropout as 0.3.