Sequence-to-Lattice Models for Fast Translation

Yuntian Deng
Harvard University
dengyuntian@seas.harvard.edu

Alexander M. Rush
Cornell University
arush@cornell.edu

Abstract

Non-autoregressive machine translation (NAT) approaches enable fast generation by utilizing parallelizable generative processes. The remaining bottleneck in these models is their decoder layers; unfortunately unlike in autoregressive models (Kasai et al., 2020), removing decoder layers from NAT models significantly degrades accuracy. This work proposes a sequence-to-lattice model that replaces the decoder with a search lattice. Our approach first constructs a candidate lattice using efficient lookup operations, generates lattice scores from a deep encoder, and finally finds the best path using dynamic programming. Experiments on three machine translation datasets show that our method is faster than past non-autoregressive generation approaches, and more accurate than naively reducing the number of decoder layers.

1 Introduction

Non-autoregressive (NAT) machine translation (Gu et al., 2017) provides multi-fold speedups compared to sequential generation by parallelizing computation across positions. NAT models are often compared to autoregressive models with deep architectures. However, autoregressive models themselves also admit structural changes that can give large speedups, for instance using shallow decoders (Kasai et al., 2020) or pruning the per sentence vocabulary (Jean et al., 2014). Unfortunately porting these structural changes to NAT models significantly hurts accuracy, reducing their benefits.

This work proposes a sequence-to-lattice formulation for translation that yields the benefits of non-autoregressive translation while reducing the practical costs of deep decoder models. The method first constructs a candidate target lattice based on the source sentence using a variant of IBM Model 2 (Brown et al., 1993). Then each lattice edge is scored based on an encoder head. Finally, exact inference is performed with the Viterbi algorithm (Forney, 1973).

Experiments on machine translation benchmarks show that this simple approach achieves fast translation without requiring an ensemble of different approaches. Our code, models, and logs are available at https://github.com/harvardnlp/cascaded-generation.

2 Related Work

Gu et al. (2017) proposed the task of non-autoregressive machine translation, and since then there have been many followup works (Lee et al., 2018, inter alia). Among these works, a line of research using structured prediction models is particularly relevant to our approach: Sun et al. (2019) proposed to use a first-order conditional random field (CRF) (Lafferty et al., 2001) to model the dependencies among adjacent target tokens; Deng and Rush (2020) used a cascaded decoding procedure to extend to higher-order CRFs; Su et al. (2021) used BERT to produce the transition scores of a first-order CRF to leverage pretraining. Our approach builds on this line of research but differs in two aspects: first, we use an efficient count-based model to construct the candidate lattice; second, we show that the sequence-to-lattice formulation allows using many fewer lattice scorer layers.

Many prior works have considered the problem of reducing the vocabulary for efficient machine translation (Jean et al., 2014; Mi et al., 2016; Shi and Knight, 2017; L'Hostis et al., 2016). The most common approach is based on the intuition that each source word can only be translated into a small set of target words, such that taking their union gives us a reduced vocabulary. In order to find which words each source word translates into, simple statistical models (Dyer et al., 2013) are usually used to get alignments. In this work, we also use a variant of IBM Model 2 to reduce the size of target-side vocabularies, but instead of finding a

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 3765–3772
November 7–11, 2021.

reduced target vocabulary per sentence, our method finds a reduced target vocabulary per position.

3 Approach

Our method formulates translation as a first-order conditional Markov model. Given a source sentence $x = x_1, \cdots, x_S$, the goal of translation is to produce the best target sentence $y = y_1, \cdots, y_T$ under:

$$P(y|x, T) = \prod_{t=1}^{T} P_\theta(y_t|y_{t-1}, t, x).$$

The full set of translations can be compactly represented as a lattice where each edge score corresponds to $P_\theta(y_t|y_{t-1}, t, x)$.

Once a lattice is constructed and scored, the best translation can be computed using the Viterbi algorithm (Forney, 1973). However, the lattice is quadratic in the vocabulary size. In order to make this approach efficient, we need to specify how to: a) generate a tractable lattice, b) score the edges, and c) parallelize the process.

Candidate Lattice Construction To generate a candidate lattice, we propose a simple statistical model. We introduce latent alignments a from target to source ($a_t \in \{1, \cdots, S\}$) and factorize the joint distribution of alignments and target, conditioned on source as,

$$P(a, y|x) = \prod_{t=1}^{T} P(y_t|x, a_t)P(a_t|x).$$

We make the simplifying assumption to use relative positions $P(y_t|x, a_t) \approx P(y_t|x_{a_t}, a_t - t)$, and to ignore the source words in the alignment prior $P(a_t|x) \approx P(a_t|S, t)$, yielding,

$$P(a, y|x) = \prod_{t=1}^{T} P(y_t|x_{a_t}, a_t - t)P(a_t|S, t).$$

Marginalizing over latent variable a, we have

$$P(y|x) = \prod_{t=1}^{T} \sum_{a_t} P(y_t|x_{a_t}, a_t - t)P(a_t|S, t),$$

which implies,

$$P(y_t|x, t) = \sum_{a_t} P(y_t|x_{a_t}, a_t - t)P(a_t|S, t).$$

Using this equation we can very efficiently generate a position-aware candidate lattice where at each position we keep the top K words with the highest $P(y_t|x, t)$ values. This produces a lattice of size K^2. To train $P(y_t|x_{a_t}, a_t - t)$ and $P(a_t|S, t)$, we use count-based MLE (without smoothing) with supervised alignments a estimated using FastAlign (Dyer et al., 2013).

Lattice Scoring To combine the best of probabilistic modeling and neural networks, we use a transformer (Vaswani et al., 2017) to parameterize $P_\theta(y_t|y_{t-1}, t, x)$. We first encode x into a memory bank using a normal transformer encoder, then we use a single-layer head to produce $P_\theta(y_t|y_{t-1}, t, x)$ for all K values of y_{t-1} while attending to the memory bank and the target position. Transformers are very suitable for this purpose because at training, we only need to modify the standard autoregressive decoder self-attention masks to learn $P_\theta(y_t|y_{t-1}, t, x)$.

Parallelization A major benefit of NAT is the ability to parallelize the model. In our approach, each neural computation of $P_\theta(y_t|y_{t-1}, t, x)$ can be done in parallel using shared encoder representations. The only sequential part in our approach is the Viterbi algorithm, which is not a bottleneck in practice.[1] For long T, this approach can be further parallelized to be of time complexity $O(\log T)$ (Särkkä and García-Fernández, 2019; Rush, 2020).

4 Experiments

Datasets We use three translation benchmarks: IWSLT14 De $\rightarrow$ En (Cettolo et al., 2014) (160k pairs), WMT14 En $\leftrightarrow$ De[2] (Macháček and Bojar, 2014) (4M pairs), and WMT16 En $\leftrightarrow$ Ro[3] (Bojar et al., 2016) ($\sim$610k pairs). We sample validation datasets to be at most 3k following Deng and Rush (2020). We also consider a knowledge distillation setup (Kim and Rush, 2016; Gu et al., 2017), where the teachers are fully autoregressive transformer baselines described below. More preprocessing details can be found in Appendix A.3.

Baselines We use the default transformer architectures in FAIRSEQ (Ott et al., 2019) for each dataset: for IWSLT14 De-En we use $N = 6, h = 4,$

[1] While our approach is technically an autoregressive model, in practice it is as fast as NAT since the neural network dominates runtime. We consider these autoregressive dependencies a positive aspect of the model.

[2] http://www.statmt.org/wmt14/translation-task.html

[3] http://www.statmt.org/wmt16/translation-task.html

Approach		Latency (ms)	Speedup (Rep.)	WMT14		WMT16		IWSLT14
Model	Settings			En-De	De-En	En-Ro	Ro-En	De-En
AR Transformer	(beam 5)	257.97	×1.00	27.43	31.50	33.91	33.86	34.46
Shallow Dec. (Kasai et al., 2020)	(6-1)	-	×2.7	27.4	30.8	33.2	34.3	-
With Distillation								
Ours	(K=64)	13.15	×19.62	24.08	28.22	30.19	31.04	31.83
Fully NAT (Gu and Kong, 2020)	(CTC+GLAT)	17.0	×16.8	27.20	31.39	33.71	34.16	-
NAT-REG (Wang et al., 2019)		22	×27.6	20.65	24.77	-	-	23.89
Hint-NAT (Li et al., 2019)		26	×30.2	21.11	25.24	-	-	25.55
imitate-NAT (Wei et al., 2019)		-	×18.6	22.44	25.67	28.61	28.90	-
BERT CRF (Su et al., 2021)	(α=1)	-	×11.31*	-	-	-	-	30.45
Coverage-NAT (Shan et al., 2021)		27.58	×10.04	21.35	25.04	30.05	30.33	-
LAT (Kong et al., 2020)	(i=1)	31	×15.68	25.20	29.91	30.74	31.24	31.92
NART-DCRF (Sun et al., 2019)		37	×10.4	23.44	27.22	27.44	-	-
Cascaded (Deng and Rush, 2020)	(K=16,i=2)	50.28	×6.34	26.34	30.69	32.70	32.66	33.90
Levenshtein (Gu et al., 2019b)		92	×4.01	27.27	-	-	33.26	-
FlowSeq-large (Ma et al., 2019)		-	-	23.72	28.39	29.73	30.72	-
CMLM (Ghazvininejad et al., 2019)	(i=10)	-	-	27.03	30.53	33.08	33.31	-
SMART (Ghazvininejad et al., 2020)	(i=10)	-	-	27.65	31.27	-	-	-
Imputer (Saharia et al., 2020)	(i=1)	-	-	25.8	28.4	-	-	-
Without Distillation								
Ours	(K=64)	13.19	×19.56	20.98	25.04	29.59	30.33	31.15
Cascaded (Deng and Rush, 2020)	(K=16,i=2)	47.05	×6.78	21.34	26.91	32.11	32.53	32.95
Levenshtein (Gu et al., 2019b)		126	×2.93	25.20	-	-	33.02	-
FlowSeq-base (Ma et al., 2019)		-	-	18.55	23.36	29.34	30.44	24.75
FlowSeq-large (Ma et al., 2019)		-	-	20.85	25.40	29.73	30.72	-

Table 1: Main results. Latency/speedup are measured on WMT14 En-De test set with batch size 1. Latency/reported speedup numbers from reference papers are not directly comparable due to implementation and hardware differences. $*$: speedup measured on IWSLT14 De-En.

$d_{\text{model}} = 512$, $d_{\text{ff}} = 1024$; for all WMT datasets we use $N = 6$, $h = 8$, $d_{\text{model}} = 512$, $d_{\text{ff}} = 2048$.

Model Settings We use the same architecture as the baselines, except that we use a single-layer lattice scorer. For candidate lattice construction we only consider the top 40 candidates from $P(y_t|x_{a_t}, a_t - t)$ per each $(x_{a_t}, a_t - t)$. For lattice decoding, we use linear regression to predict approximate length L from S. We introduce a padding symbol to allow for variable length generation and consider lengths T from $L-\Delta L$ to $L+\Delta L$ where L is the predicted length and $\Delta L = 3$.

Results Table 1 shows the main results. Latency is measured on WMT14 En-De (full results can be found in Appendix A.2). Our approach is the fastest in terms of raw speed yet still reaches a decent accuracy. It is hard to compare reported speedups across works. While there are higher speedups (Wang et al., 2019; Li et al., 2019), we have a lower baseline latency which might make further speedups harder.

Notably, only Fully NAT (Gu and Kong, 2020) outperforms our approach in terms of accuracy (27.20 v.s. 24.08 on WMT14 En-De w/ distillation) while also giving a comparable speedup (×16.8 v.s. ×19.62). However, we note that Fully NAT is an ensemble of best-of-class techniques including glancing target (Qian et al., 2020), CTC (Graves et al., 2006), and VAE (Kingma and Welling, 2013).

5 Analysis

Lattice Construction Various methods can construct a candidate lattice by filtering the top K tokens for each target-position t. The *shared vocab* reduction approach of L'Hostis et al. (2016) ignores the position t ($P(y_t|x,t) \approx P(y.|x)$). During training, FastAlign is used to estimate $P(y.|x_s)$. During inference, for each source word x_s, the top K words maximizing $P(y.|x_s)$ are selected. Alternatively we can use an *NAT baseline* model, $P(y|x) = \prod_t P(y_t|x,t)$, where $P(y_t|x,t)$ is parameterized with a six-layer transformer encoder-decoder. This approach has access to the position and a deep decoder, but is much slower.

Figure 1 compares different approaches for constructing a candidate lattice. Our method significantly outperforms *shared vocab* which produces a single reduced vocabulary at the target side. While

our approach underperforms *NAT baseline*, it is competitive and much more efficient.

While our statistical model is very efficient, using it alone (without lattice scoring and decoding, or equivalently, $K = 1$) gets a much lower BLEU, as shown in Table 2.

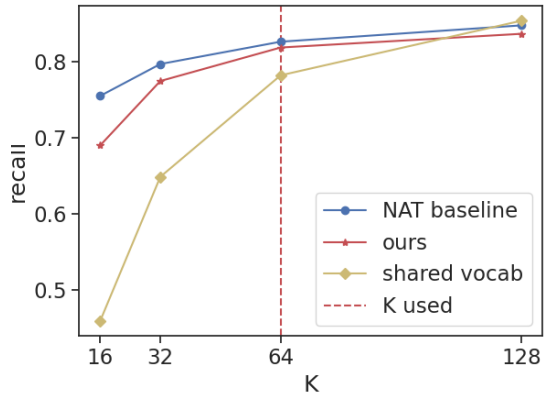

Figure 1: Recall of candidate lattice generation on IWSLT14 De-En distilled val. Measures the percentage of target tokens retained in the lattice.

Settings	BLEU	Latency
K=64	31.05	9.99ms
K=32	30.16	9.81ms
K=1	5.10	0.41ms

Table 2: The effect of lattice generation K on IWSLT14 De-En val (w/ distillation, $\Delta L = 0$).

Sequence-to-Lattice Formulation Figure 3 plots BLEU score with the number of lattice scorer layers. We can see that a sequence-to-lattice formulation significantly outperforms the baseline NAT model, and that it enables using much fewer layers whereas the baseline accuracy quickly degrades as the number of scorer layers decreases. Being able to use fewer lattice scorer layers allows faster inference, as shown in Appendix A.1. The fact that NAT accuracy degrades indicates that structural changes, like those proposed for autoregressive models by Kasai et al. (2020), can hurt NAT models.

Figure 2 demonstrates the importance of Viterbi search with respect to the final model. While almost 15% of words are already ranked highest without lattice decoding, there is a non-negligible percentage of changes due to search.

Latency Analysis Figure 4 shows latency breakdown as a function of length. Most time is spent

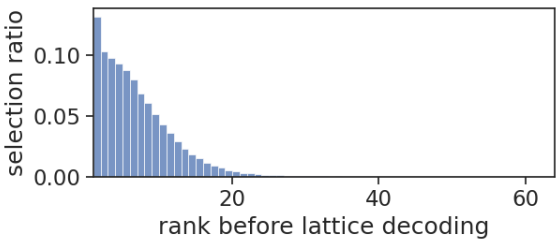

Figure 2: Reordering by lattice decoding, measured by the selection ratio given a rank before search (using edge scores only). We use IWSLT14 De-En val with $K = 64$ and $\Delta L = 0$.

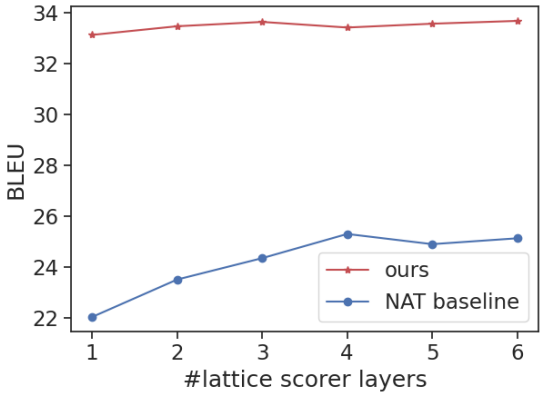

Figure 3: BLEU v.s. number of lattice scorer layers on the validation set of IWSLT14 De-En (w/ distillation).

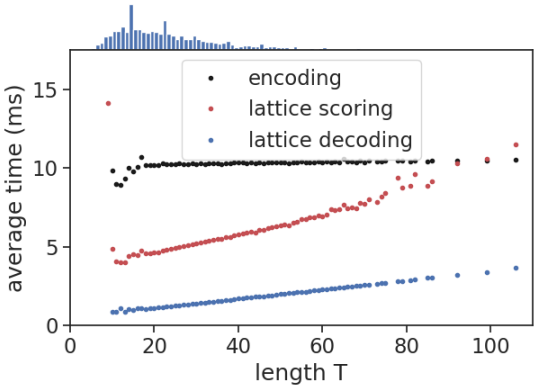

Figure 4: Latency breakdown as a function of target length. Time is measured on WMT14 En-De test with $K = 64$ and asynchronous execution disabled. The histogram on top shows the distribution of target length.

on the encoder since there are six encoder layers. Given the rarity of long sentences, in most cases the other two times are dominated by the encoder.

In practice, we use serial Viterbi decoding, which does grow linearly with length, but remains faster. Lattice scoring time is parallelizable but also grows with length. Practically hardware has lim-

ited parallel capacity, creating a bottleneck when sequences become very long. Future work will need to better explore parallel approaches beyond the regime of sentence-level generation.

6 Conclusion

In this work, we find that using a sequence-to-lattice formulation enables using much smaller model architectures for fast machine translation. Our approach first generates a candidate lattice using a statistical model, then uses a transformer with a position-wise head layer to score the lattice, and finally uses the Viterbi algorithm to find the best hypothesis. Experiments on three machine translation benchmarks show that our simple approach is very fast yet achieves a decent accuracy.

Acknowledgements

We would like to thank Justin Chiu and Jiawei Zhou. YD is sponsored by NSF 1704834 and a Baidu AI Fellowship. AMR is sponsored by NSF CAREER 2037519.

References

Ondřej Bojar, Yvette Graham, Amir Kamran, and Miloš Stanojević. 2016. Results of the wmt16 metrics shared task. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 199–231.

Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, volume 57.

Yuntian Deng and Alexander Rush. 2020. Cascaded text generation with markov transformers. *Advances in Neural Information Processing Systems*, 33.

Chris Dyer, Victor Chahuneau, and Noah A Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648.

G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Constant-time machine translation with conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Marjan Ghazvininejad, Omer Levy, and Luke Zettlemoyer. 2020. Semi-autoregressive training improves mask-predict decoding. *arXiv preprint arXiv:2001.08785*.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376.

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

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

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*, pages 11179–11189.

Junliang Guo, Xu Tan, Di He, Tao Qin, Linli Xu, and Tie-Yan Liu. 2019. Non-autoregressive neural machine translation with enhanced decoder input. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3723–3730.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

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*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

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

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Xiang Kong, Zhisong Zhang, and Eduard Hovy. 2020. Incorporating a local translation mechanism into non-autoregressive translation. *arXiv preprint arXiv:2011.06132*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.

Gurvan L'Hostis, David Grangier, and Michael Auli. 2016. Vocabulary selection strategies for neural machine translation. *arXiv preprint arXiv:1610.00072*.

Zhuohan Li, Zi Lin, Di He, Fei Tian, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Hint-based training for non-autoregressive machine translation. *arXiv preprint arXiv:1909.06708*.

Jindřich Libovickỳ and Jindřich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. *arXiv preprint arXiv:1811.04719*.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. Flowseq: Non-autoregressive conditional sequence generation with generative flow. *arXiv preprint arXiv:1909.02480*.

Matouš Macháček and Ondřej Bojar. 2014. Results of the wmt14 metrics shared task. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 293–301.

Elman Mansimov, Alex Wang, and Kyunghyun Cho. 2019. A generalized framework of sequence generation with application to undirected sequence models. *arXiv preprint arXiv:1905.12790*.

Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. 2016. Vocabulary manipulation for neural machine translation. *arXiv preprint arXiv:1605.03209*.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help? In *Advances in Neural Information Processing Systems*, pages 4696–4705.

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. *arXiv preprint arXiv:1904.01038*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. 2020. Glancing transformer for non-autoregressive neural machine translation. *arXiv preprint arXiv:2008.07905*.

Alexander M Rush. 2020. Torch-struct: Deep structured prediction library. *arXiv preprint arXiv:2002.00876*.

Chitwan Saharia, William Chan, Saurabh Saxena, and Mohammad Norouzi. 2020. Non-autoregressive machine translation with latent alignments. *arXiv preprint arXiv:2004.07437*.

Simo Särkkä and Ángel F García-Fernández. 2019. Temporal parallelization of bayesian filters and smoothers. *arXiv preprint arXiv:1905.13002*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Yong Shan, Yang Feng, and Chenze Shao. 2021. Modeling coverage for non-autoregressive neural machine translation. *arXiv preprint arXiv:2104.11897*.

Xing Shi and Kevin Knight. 2017. Speeding up neural machine translation decoding by shrinking run-time vocabulary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 574–579.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. *arXiv preprint arXiv:1902.03249*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, pages 10086–10095.

Yixuan Su, Deng Cai, Yan Wang, David Vandyke, Simon Baker, Piji Li, and Nigel Collier. 2021. Non-autoregressive text generation with pre-trained language models. *arXiv preprint arXiv:2102.08220*.

Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhihong Deng. 2019. Fast structured decoding for sequence models. In *Advances in Neural Information Processing Systems*, pages 3011–3020.

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.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5377–5384.

Bingzhen Wei, Mingxuan Wang, Hao Zhou, Junyang Lin, and Xu Sun. 2019. Imitation learning for non-autoregressive neural machine translation. *arXiv preprint arXiv:1906.02041*.

Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. 2020. Pointer: Constrained text generation via insertion-based generative pre-training. *arXiv preprint arXiv:2005.00558*.

Jiawei Zhou and Phillip Keung. 2020. Improving non-autoregressive neural machine translation with monolingual data. *arXiv preprint arXiv:2005.00932*.

A Appendix

A.1 Speedup and BLEU Versus Layers

Table 3 shows the speedup and BLEU as we vary the number of lattice scorer layers. We can see that reducing the number of lattice score layers makes inference much faster without hurting BLEU score much.

#Layers	Speedup	BLEU
1	×16.78	30.90
2	×14.47	31.83
3	×12.74	32.02
4	×9.62	30.00
5	×10.28	31.15
6	×9.41	31.33

Table 3: Speedup and BLEU score as we vary the number of lattice score layers (IWSLT14 De-En, K=64).

A.2 Full Results

We used $K = 64$ in the main paper, and only reported latency/speedup on WMT14 En-De. Full results can be found at Table 4, Table 5, Table 6, Table 7, and Table 9.

A.3 Data Preprocessing

To process the data, we use Byte Pair Encoding (BPE) (Sennrich et al., 2015; Kudo and Richardson, 2018) learned on the training set with a shared vocabulary between source and target. For IWSLT14 the vocabulary size is 10k; for WMT14 the vocabulary size 40k. For WMT16 we use the processed data provided by Lee et al. (2018).

Model	Settings	Latency (Speedup)	BLEU
Transformer	(beam 5)	257.97ms (×1.00)	27.43
W/ Distillation			
Ours	(K=32)	12.11ms (×21.30)	21.57
Ours	(K=64)	13.15ms (×19.62)	24.08
Ours	(K=128)	14.75ms (×17.49)	25.37
W/O Distillation			
Ours	(K=32)	12.34ms (×20.91)	18.89
Ours	(K=64)	13.19ms (×19.56)	20.98
Ours	(K=128)	14.98ms (×17.22)	21.73

Table 4: Results on WMT14 En-De.

Model	Settings	Latency (Speedup)	BLEU
Transformer	(beam 5)	238.52ms (×1.00)	31.50
W/ Distillation			
Ours	(K=32)	12.42ms (×19.20)	26.15
Ours	(K=64)	13.00ms (×18.35)	28.22
Ours	(K=128)	14.80ms (×16.11)	29.20
W/O Distillation			
Ours	(K=32)	12.29ms (×19.41)	23.62
Ours	(K=64)	13.26ms (×17.99)	25.04
Ours	(K=128)	14.78ms (×16.14)	25.64

Table 5: Results on WMT14 De-En.

Model	Settings	Latency (Speedup)	BLEU
Transformer	(beam 5)	261.43ms (×1.00)	33.91
W/ Distillation			
Ours	(K=32)	12.59ms (×20.76)	27.72
Ours	(K=64)	13.10ms (×19.96)	30.19
Ours	(K=128)	14.72ms (×17.76)	30.80
W/O Distillation			
Ours	(K=32)	12.60ms (×20.75)	27.14
Ours	(K=64)	13.35ms (×19.58)	29.59
Ours	(K=128)	14.85ms (×17.60)	30.16

Table 6: Results on WMT16 En-Ro.

Model	Settings	Latency (Speedup)	BLEU
Transformer	(beam 5)	235.63ms (×1.00)	33.86
W/ Distillation			
Ours	(K=32)	12.48ms (×18.88)	30.16
Ours	(K=64)	13.13ms (×17.95)	31.04
Ours	(K=128)	14.52ms (×16.23)	31.15
W/O Distillation			
Ours	(K=32)	12.71ms (×18.54)	29.53
Ours	(K=64)	13.21ms (×17.84)	30.33
Ours	(K=128)	14.45ms (×16.31)	30.45

Table 7: Results on WMT16 Ro-En.

Dataset	dropout	fp16	GPUs	batch	accum	warmup steps	max steps	max lr	weight decay
WMT14 En-De/De-En	0.1	Y	3	4096	3	4k	240k	7e-4	0
WMT16 En-Ro/Ro-En	0.3	Y	3	5461	1	10k	240k	7e-4	1e-2
IWSLT14 De-En	0.3	N	1	4096	1	4k	120k	5e-4	1e-4

Table 8: Optimization settings. We used the same settings for knowledge distillation experiments.

Model	Settings	Latency (Speedup)	BLEU
Transformer	(beam 5)	171.20ms ($\times$1.00)	34.46
W/ Distillation			
Ours	(K=32)	10.00ms ($\times$17.12)	30.90
Ours	(K=64)	10.20ms ($\times$16.78)	31.83
Ours	(K=128)	10.61ms ($\times$16.13)	32.02
W/O Distillation			
Ours	(K=32)	10.00ms ($\times$17.12)	30.00
Ours	(K=64)	10.11ms ($\times$16.93)	31.15
Ours	(K=128)	10.63ms ($\times$16.11)	31.33

Table 9: Results on IWSLT14 De-En.

A.4 Optimization Settings

We train our model as a Markov transformer (Deng and Rush, 2020) with bigrams to trigrams. We used Adam optimizer (Kingma and Ba, 2014), with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and inverse square root learning rate decay after linear warmup (Ott et al., 2019). We train with label smoothing strength 0.1 (Müller et al., 2019). For model selection, we used BLEU score on validation set, with $K = 64$ and $\Delta L = 3$. Other hyperparameters can be found at Table 8.

A.5 Implementation Details

Our implementation is based on FAIRSEQ (Ott et al., 2019) and PyTorch (Paszke et al., 2019), and we use an Nvidia A100 GPU with CUDA version 11.1 to perform inference.