Pinyin-BART: An End-to-End Chinese Input Method

Anonymous ACL submission

Abstract

A Chinese Input Method Engine helps user convert a keystroke sequence into the desired Chinese character sequence. It is usually a cascaded process in which the original input sequence is firstly corrected to remove typos, then segmented into the pinyin token sequence, and finally converted into a Chinese character sequence. Errors are prone to accumulate and propagate in that pipeline. This paper summarizes that process as a Key-to-Character (K2C) conversion task and solve it in a unified end-to-end way. Pinyin-bart is proposed which can effectively solve the error propagation problem and improve the IME engine performance significantly in experiments. Moreover, we model the user real input behaviors and design a method to generate the training corpus with typos for the K2C task. It further improves the robustness of Pinyin-bart. Finally, we design a non-autoregressive (NAR) decoder for Pinyin-bart and obtain 9x+ acceleration with limited performance degradation, which makes the deployment possible on the commercial input software.

1 Introduction

Some of languages, such as Chinese, Japanese and Thai language, can not be input directly through the standard keyboard. Users type in these languages via some commercial input software, such as Microsoft Input Method (Gao et al., 2002), Google Chinese Input Method[1], Sogou Input Method[2], Baidu Input method[3], Huawei Celia Keyboard[4], and so on. Pinyin is the official romanization representation for Chinese language. It's natural for a user to type in pinyin through the keyboard. And

[1] https://www.google.com/inputtools/
[2] https://pinyin.sogou.com/
[3] https://shurufa.baidu.com/
[4] https://consumer.huawei.com/uk/community/details/App-Gallery-Celia-Keyboard-is-now-available/topicId_48409/

Figure 1: A user Types in Chinese via Pinyin in IME. [5]

the input software converts the pinyin into the character sequence. As the figure 1 shows, a user inputs a keystroke sequence of "woainizongguo", and the software segments it into the pinyin sequence "wo′ai′ni′zong′guo" then converts it into the Chinese character sequence that user desires "我爱你中国 (I love you China)".

Specifically, as the figure 2 shows, the IME engine takes it as a cascaded process. Firstly, the correction module corrects the typos in the original keystroke sequence. In the example of the figure 1, the blade-alveolar sound of 'zong' is corrected into the cacuminal sound of 'zhong'. It is usually implemented by some rule system for efficiency. Secondly, the modified keystroke sequence is segmented into the pinyin token sequence. For example, "woainizhongguo" is segmented into "wo′ai′ni′zhong′guo". The tokenizer is usually implemented by some Chinese word segmentation algorithm, i.e. the Maximum Matching (MM) algorithm. Lastly, the pinyin sequence is converted into the character sequence, which is called the Pinyin to Character (P2C) conversion task (Zhang et al., 2019a; Yao et al., 2018; Xiao et al., 2007). It is usually resolved as a sequence labeling task by the Ngram language model (Goodman, 2001) together with the Viterbi search algorithm (Viterbi, 2006).

In the above process, the error in the previous step is prone to accumulate and propagate to the later step, which hurts the IME engine performance badly as presented in the later experiments. In this paper, we summarizes those steps into a unified end-to-end process named the Key-to-Character (K2C) task and proposes Pinyin-bart to solve it. As

[5] The screenshot is from Sogou Input Method software

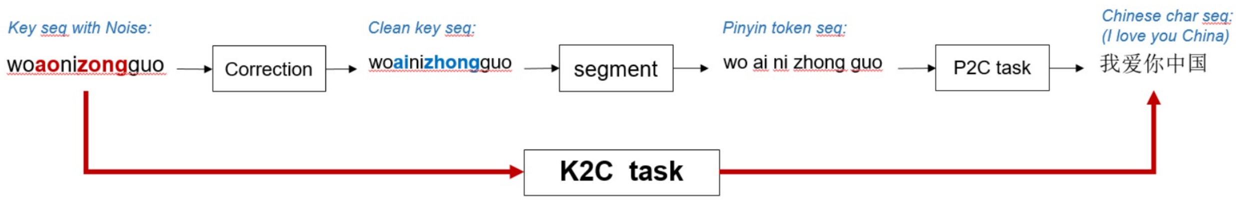

Figure 2: The Key to Character Conversion Task

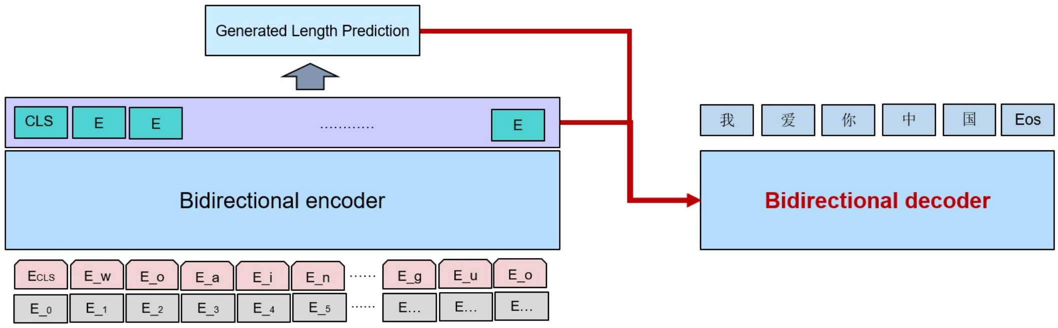

Figure 3: Pinyin-Bart Model Architecture. In the input layer, E_0 and E_1 are the position embeddings; E_w and E_o are the input token embeddings. The decoder of Pinyin-Bart adopts the bidirectional attentions. An additional length predictor is added on the top of encoder to guide the generation process.

far as we know, it's the first work to build the IME engine in an end-to-end way. We summarize the main contributions of this paper as follows:

- We propose Pinyin-bart to solve the K2C task and build the IME engine in an end-to-end way, which effectively resolves the error propagation problem in the cascaded IME engine. As far as we know, it's the first end-to-end IME engine.

- We model the user input behavior and design a method to generate the massive corpus with typos automatically for the K2C task, which further improve the robustness of Pinyin-bart.

- We design the NAR decoder for Pinyin-bart and boost the inference speed significantly with only little performance degradation.

2 Method

In this section, we describe the details about Pinyin-bart. Firstly, we introduce the K2C task formally in the section 2.1. Then we present how the Pinyin-bart is implemented in the section 2.2. Lastly, we describe the method that models user input behavior and generates the massive corpus with typos in the section 2.3.

2.1 The K2C Conversion Task

As illustrated in the figure 2, the K2C conversion task is to convert the user keystroke sequence from keyboard directly into the Chinese sentence. Formally, $k_1, k_2, ..., k_n$ is the keystroke sequence. They are converted into the character sequence of $c_1, c_2, ...c_m$ in the K2C conversion task. Usually the value of m is smaller than n since one Chinese character corresponds one pinyin token which is composed of multiple letters. The task can be resolved in a cascaded way as most of the commercial input software does, or in an end-to-end way by Pinyin-bart in this paper.

2.2 Pinyin-bart

We build Pinyin-bart based on the standard bart model (Lewis et al., 2020). To fit for the K2C task, we do some modifications in several aspects, including the training paradigm as described in the section 2.2.1, the embedding layer as described in the section 2.2.2 and the NAR decoder described in the section 2.2.3.

2.2.1 The Training Paradigm

The standard bart adopts the pretrain-then-finetune paradigm, like most of the transformer model does. It firstly pretrains the model on the massive unlabeled corpus by some self-supervised learning tasks, such as Masked Language Model (MLM), Text Infilling, Sentence Permutation and so on. Then the model is finetuned on the labelled corpus on the target task, such as SQuAD (Rajpurkar et al., 2016), MNLI (Williams et al., 2018), XSum (Narayan et al., 2018), and so on. The pretrain process leverages the general knowledge contained in the unlabeled corpus, which boosts the performance significantly on the target tasks. As described in the section 2.3 later, we design the method to create the massive labelled corpus for the K2C task automatically. Therefore, we train the Pinyin-bart directly on the target K2C task instead of the pretrain-then-finetune paradigm.

2.2.2 The Embedding Layer

Different from the standard bart, there is no segment embedding in the embedding layer of Pinyin-bart as illustrated in the figure 3. It is because there is no pretrain process in Pinyin-bart and it no longer needs the self-supervised learning task, especially the sentence-level pretrain task. Besides, Pinyin-bart takes the keystroke sequence as input rather than the subword sequence as the standard bart dose. There are only 26 individual letters which is 3x order of magnitude smaller than the number of subword (more than 50,000) in the standard bart. Thus the size of embedding layer of Pinyin-bart is much smaller than the standard bart.

2.2.3 The NAR Decoder

The standard bart adopts the autoregressive decoder like GPT which predicts the current token based on the previous one. The advantage is to leverage the dependency between tokens in sequence, whereas it's pretty slow during the inference, which hinders its deployment in the commercial input software. The NAR decoder is proposed in the machine translation domain (Gu et al., 2018). It makes the independent assumption on the tokens in sequence, which makes the inference process parallel so that accelerates the inference significantly. In this paper, we design the NAR decoder for Pinyin-bart.

As described in the figure 3, we firstly replace the GPT-like decoder of single direction attention in the standard bart with the Bert-like decoder of bidirectional attention in Pinyin-bart, which can leverage the parallel computation in GPU. Secondly, we add a predictor to predict the length of target sequence. Specifically, we add a mean pooling layer stacked with a regression layer on the top of the encoder. Thirdly, to train the Pinyin-bart model, the cross-entropy (ce) loss is adopted for the target sequence prediction task, and the mean square error (mse) loss is adopted for the length prediction task. They are weighted combined together, as shown in the formula 1.

$$loss_{total} = \lambda_1 * loss_{ce} + \lambda_2 * loss_{mse} \quad (1)$$

During the inference, the tokens in the target sequence are generated parallel, and the target length is predicted as well. The length is rounded off from float to the integer value. Then the target sequence is simply truncated by that length.

2.3 Generating Massive Labelled Corpus

We generate the massive labelled corpus for the K2C task. The whole process is described in the figure 4.

Firstly, the text in Chinese corpus, i.e. the sentence of "我爱你中国 (I love you China)", is converted into the pinyin token sequence, i.e. "wo′ai′ni′zong′guo". This task is called Text-to-Pinyin conversion which can achieve more than 99.9% accuracy (Zhang and Laprie, 2003). In this way, we can get the massive pinyin corpus automatically. Secondly, user does not type in any separator to split the pinyin token explicitly during its input process in reality, so we combine the pinyin tokens in a sequence together into the keystroke sequence. The "wo′ai′ni′zong′guo" is then combined into 'woainizhongguo'. Thirdly, some kind of noise is added into the keystroke sequence so as to simulate user's typos. Finally we get the parallel corpus with the Chinese character sequence as well as the keystroke sequence with typos.

To add noise to the keystroke sequence, we select some positions randomly from the original sequence. Then three operators are applied on the letters of these positions with equal probability, including 'Delete', 'Insert' and 'Replace'. Some probability distribution is required to guide the 'Insert' and 'Replace' operator, i.e. to insert which letter before the current position. The uniform distribution is the most straightforward choice. However, it's sub-optimal because it does not take the consideration of the keyboard layout and the user's behavior in reality. For example, when user types in the letter of 'z' in 'zong', it is prone to mistype it as 'x' instead of 'p' because the position of 'x' is much closer to 'z' on the keyboard layout than 'p' dose. Besides, the typos of one user are also usually different from another user due to their different input habits. In this paper, we collect the user type-in

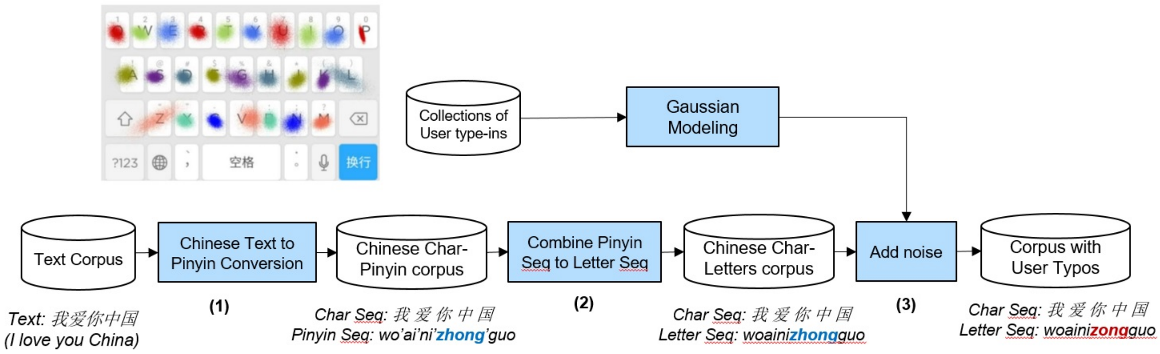

Figure 4: Prepare the Massive Labelled Corpus for the K2C Task.

behaviors in reality [6]. Some of them are visualized as the points cloud shown at the top left of figure 4. Based on these data, we build the Gaussian model for each key on the keyboard layout, as the formula 2 shows below:

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{\sigma^2}) \quad (2)$$

According to the Gaussian model, we can calculate the probability that the current key is mis-typed to any other key. And we finally generate the typo noise according to that mis-type probability matrix, as illustrated at the upper half part of figure 4.

3 Experiment

3.1 Data Set Preparation

As far as we know, there is no public benchmark for the Chinese Pinyin input method. So we build our own data set and will make it public to the community later. More than 2.6 million articles are collected from the Chinese news websites. We firstly segment them into sentence by the punctuation list including comma, period, and so on. Then we filter the characters which there are no pinyin corresponded to. Thirdly, these sentences are further segmented by a max length (i.e. 16 in our experiment) because user only types in a few Chinese characters at one time. Lastly, we make them as the labelled corpus as described in the section 2.3. Most of the corpus are taken as the training corpus, and another one thousand disjoint articles are taken as the test corpus, as described in the table 1.

Besides, to evaluate the performance of the cascade IME engine, we build several test corpus with different degree of noise:

[6] We get these data under the users' authorization

Corpus	#Articles	#Chars	#Disk
Train	2,603,869	2,432,585,138	9.7G
Test	1000	926,792	3.7M

Table 1: The Detailed Information of Corpus

- **No Typos and No Segment Errors**. In the first one, we assume that there is no typo from user's input and the pinyin tokenizer in the figure 2 works perfectly. It looks like "我爱你中国 (wo′ai′ni′zong′guo)". It is a total clean environment and the only factor that matters the IME performance is language model. It can be taken as the upper bound of the IME engine performance in reality. We get this corpus by processing only the first step of figure 4.

- **No Typos BUT Segment Errors**. In the second corpus, we assume that there is no typo but the pinyin tokenizer works probably with errors. It looks like "我爱你中国 (wo′a′in′i′zhong′guo)". It is a possible situation if the user types carefully and precisely. We can get it by re-segmenting the combined keystroke sequence automatically after the second step of figure 4 by some real tokenizer, i.e. the MM algorithm.

- **Typos and Segment Errors**. In the last one, we assume both the typo and the segmenting error, which is the situation in the real world. It might look like "我爱你中国 (wo′ao′ni′zong′guo)". We can get it by re-segmenting the sequence containing noises after the third (last) step of of the figure 4.

During evaluating, we apply language model directly on these kinds of corpus to simulate the performance of the cascaded IME engine in various noisy environment.

3.2 Evaluation Metrics

We use the character-based precision to evaluate the performance of the IME engine. It is defined as the ratio that the IME engine converts to the Chinese character correctly, as described in the formula 3.

$$Precision_{char\_based} = \frac{\#correct\_converted\_char}{\#total\_converted\_char} \quad (3)$$

3.3 Baseline Models and Experiment Settings

The cascaded IME engine is taken as the baseline model, and is evaluated on the corpus with different degree of noise as described in the section 3.1. Several kinds of language models are integrated respectively into the cascaded IME engine:

- **Bigram**. Bigram is the De facto model adopted widely in the commercial IME engine. We build the bigram model on the lexicon of *the Table of General Standard Chinese Characters* [7] which contains more than 6 thousand Chinese frequent characters. No pruning strategy is adopted since the scale of training corpus is large enough.

- **LSTM**. LSTM is reported that obtains better performance than the bigram model (Zhang et al., 2019b; Yao et al., 2018; Malhotra et al., 2015). In out implement of the LSTM model, both the embedding size and the hidden size are 256, and the learning rate is $5e^{-4}$. The batch size is $2k$ and the epoch number is 10.

- **Bart**. We use the standard bart in the sequence-to-sequence way. The pinyin token sequence is taken as input, and the Chinese character sequence is taken as output. It is trained from scratch directly on the P2C task. We follow most of the specifications in the paper (Lewis et al., 2020), except that the max sequence length is set to 16 instead of 512. The epoch number is 10.

For the Pinyin-bart model, the keystroke sequence is taken as input. It is trained directly on the K2C task as described in the section 2.2.1. The experimental settings are exactly the same as the standard bart model. In the formula 1, the value of λ_1 is 1 and the value of λ_2 is 0.01.

3.4 Experimental Results on the K2C Task

The experimental results are presented in the table 2. Two ratios of typo noises (1% and 5%) are added into the corpus.

Firstly, let's take a quick look at the results under the clean environment (no typo and no segment error). The bigram model obtains 84.56% precision and the LSTM model gets a better result of 89.71% (5.15% $\uparrow$) as reported in the previous articles (Zhang et al., 2019b). The standard bart model achieves 96.97% which outperforms the above two models (12.41% $\uparrow$ and 7.26% $\uparrow$) significantly. It proves that language model plays a crucial role in the cascaded IME engine and its capacity can improve the performance greatly.

Secondly, the performance of the cascaded IME engine decreases badly in the noisy environment. Taking the bigram model as an example, the precision decreases from 84.56% to 79.30% (5.26% $\downarrow$) under the segment errors, and further to 66.87% (17.69% $\downarrow$) under the typo errors as well, and lastly to 37.75% (46.81% $\downarrow$) as the typo ratio increases. The similar results can be observed in the LSTM model and even in the powerful bart model.

Thirdly, Pinyin-bart achieves much higher precision than the standard bart in the cascade IME engine under the same noisy environment. For example, with the 1% typos and the segment error, Pinyin-bart gets 94.86% precision which is much higher (11.13% $\uparrow$) than 83.73% of the standard bart model. The improvement is further expanded to 35.10% $\uparrow$ as the typo ratio increase to 5%. These results are also significantly higher than the bigram model and the LSTM model. It proves that Pinyin-bart can effectively avoid the error propagation problem and performs much more robust than the cascaded IME engine.

3.5 Effectiveness of Modeling User Behaviors

In the section 2.3, we model user's input behavior and generate the typos for the training corpus of the K2C task. In this section, we compare it with the method that adds typos by the uniform distribution. The results are presented in the table 3.

As we can see, Pinyin-bart achieves better performances. As the typo ratio increases from 1% to 5%, the improvement rises from 2.29% to 5.49% accordingly. It proves that our method can generate

[7] https://en.wikipedia.org/wiki/Table_of_General_Standard_Chinese_Characters

Model	Typo Error	Segment Error	Precision	Improvement
Bigram	no	no	84.56%	NV
Bigram	no	yes	79.30%	5.26%↓
Bigram	1%	yes	66.87%	17.69%↓
Bigram	5%	yes	37.75%	46.81%↓
LSTM	no	no	89.71%	5.15%↑
LSTM	no	yes	84.96%	4.75%↓
LSTM	1%	yes	66.87%	22.84%↓
LSTM	5%	yes	51.75%	37.96%↓
Bart	no	no	96.97%	12.41%↑
Bart	no	yes	93.05%	3.92%↓
Bart	1%	yes	83.73%	13.24%↓
Bart	5%	yes	57.39%	39.58%↓
Pinyin-bart	1%	yes	94.86%	11.13%↑
Pinyin-bart	5%	yes	92.49%	35.10%↑

Table 2: The Experimental Results on the K2C Task

Model	Typo Error	Segment Error	Precision	Improvement
Pinyin-bart-uni	1%	yes	92.57%	NV
Pinyin-bart-uni	5%	yes	87.00%	NV
Pinyin-bart	1%	yes	94.86%	2.29%↑
Pinyin-bart	5%	yes	92.49%	5.49%↑

Table 3: Effectiveness of Modeling User Behaviors. *Pinyin-bart-uni* is trained with the typos generated by the uniform distribution. *Pinyin-bart* is trained with the typos generated by the Gaussian model.

the typos closer to the reality, and boost the IME engine's performance.

3.6 Effectiveness of the NAR Decoder

In this section, we compare the NAR decoder with the AR decoder in Pinyin-bart on both the performance and the inference speed. The experimental results are presented in the table 4.

Compared to the AR-model, there is 0.03% performance drop from the NAR-model under the 1% typo ratio, and further 0.91% drop under the 5% typo ratio. Considering the fact that the precision of Pinyin-bart has already exceeded 90%, that degradation is very slightly. However, the inference process is accelerated greatly. The time to infer one token drops from 15.66ms to 1.60ms under the 1% typo ratio, and drops from 16.09ms to 1.73ms under the 5% typo ratio, which is reduced by more than 9 times. It makes the deployment of Pinyin-bart possible to the commercial input method software.

4 Related Works

4.1 Language model

Language model predicts the current word probability by its previous words. It plays an essential role in the P2C task in the IME engine. The dominant model is the Ngram model (Bahl et al., 1983). However, its simplicity and low capacity limits its performance. In recent years, RNN is proposed to improve the performance by modeling longer history information (Kalchbrenner and Blunsom, 2013). Variant network architectures are proposed to solve the vanishing gradient problem and the exploding gradient problem, such as LSTM (Malhotra et al., 2015; Graves et al., 2013), GRU (Cho et al., 2014), and so on. Yao et al. (2018) replaces the Ngram model with the LSTM model in the IME engine and get performance improvement both in the candidate prompt task and in the P2C task. It further proposes an incremental selective softmax method to solve the LSTM efficiency problem in the Viterbi algorithm. Zhang et al. (2019b) applies LSTM in a sequence-to-sequence way in the P2C task, and verify it in a smart sliding input method. Zhang et al. (2019a) designs a novel online learn-

Model	Typo Error	Segment Error	Precision	Improvement	ms/token	Speedup Times
AR-model	1%	yes	94.86%	NV	15.66	NV
AR-model	5%	yes	92.49%	NV	15.66	NV
NAR-model	1%	yes	94.83%	0.03%↓	1.60	9.78↑
NAR-model	5%	yes	91.58%	0.91%↓	1.73	9.30↑

Table 4: Comparison between autoregressive Pinyin-bart and non-autoregressive Pinyin-bart. *AR-model* is the Pinyin-bart with the autoregressive decoder as the standard bart does. *NAR-model* is the Pinyin-bart with the non-autoregressive decoder as described in the section 2.2.3

ing method that adapts the vocabulary to the P2C task. Huang et al. (2018) takes the P2C task as a language translation problem. The neural machine translation model is adopted in which the RNN model is used as encoder and a global attention model is used as decoder.

4.2 Non-autoregressive Machine Translation

Usually the decoder in the neural machine translation system is the autoregressive one. Recently, the non-autoregressive decoder is proposed to accelerate the inference speed. Especially, there are two kinds of non-autoregressive models. The first one is **fully non-autoregressive model** which generates the target sequence simultaneously with single forward of network, such as the vanilla NAT model (Gu et al., 2018). The NAT-CRF model (Sun et al., 2019) adds a CRF layer on the top of the NAT decoder so as to build the token dependency in the target sequence. Gu and Kong (2021) makes a detailed investigation on the aspects that take effective on the NAT model. The second one is **the iterative refinement non-autoregressive models** (Lee et al., 2018) in which an additional decoder is adopted to refine the generated target sequence in an iterative way. CMLM (Ghazvininejad et al., 2019) makes use of the Masked Language Model (MLM) task to refine the generated result. A bert-like decoder with bidirectional attentions is adopted, and at each iteration it selects some tokens to mask and predict them again. In this way, the un-masked tokens can be taken as the contexts to improve the prediction of the masked token.

5 Conclusions

In this paper, we propose the Key to Character conversion task and design Pinyin-bart to build the IME engine in an end-to-end way. Compared with the cascaded IME engine, Pinyin-bart can solve the error propagation problem effectively and shows much more robustness in the noisy input environment. Moreover, our method of modeling user input behavior can improve its robustness further. Lastly, the non-autoregressive decoder adopted in Pinyin-bart can accelerate the inference speed greatly with little performance degradation.

References

Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5(2):179–190.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL.

Jianfeng Gao, Joshua Goodman, Mingjing Li, and Kai-Fu Lee. 2002. Toward a unified approach to statistical language modeling for chinese. *ACM Trans. Asian Lang. Inf. Process.*, 1(1):3–33.

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 2019, Hong Kong, China, November 3-7, 2019*, pages 6111–6120. Association for Computational Linguistics.

Joshua T. Goodman. 2001. A bit of progress in language modeling. *Comput. Speech Lang.*, 15(4):403–434.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

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, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 120–133. Association for Computational Linguistics.

Yafang Huang, Zuchao Li, Zhuosheng Zhang, and Hai Zhao. 2018. Moon IME: neural-based chinese pinyin aided input method with customizable association. In *Proceedings of ACL 2018, Melbourne, Australia, July 15-20, 2018, System Demonstrations*, pages 140–145. Association for Computational Linguistics.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1700–1709. ACL.

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, Brussels, Belgium, October 31 - November 4, 2018*, pages 1173–1182. 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 pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *23rd European Symposium on Artificial Neural Networks, ESANN 2015, Bruges, Belgium, April 22-24, 2015*.

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, Brussels, Belgium, October 31 - November 4, 2018*, pages 1797–1807. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhi-Hong Deng. 2019. Fast structured decoding for sequence models. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3011–3020.

Andrew J. Viterbi. 2006. A personal history of the viterbi algorithm. *IEEE Signal Process. Mag.*, 23(4):120–142.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Jinghui Xiao, Bingquan Liu, and Xiaolong Wang. 2007. An empirical study of non-stationary ngram model and its smoothing techniques. *Int. J. Comput. Linguistics Chin. Lang. Process.*, 12(2).

Jiali Yao, Raphael Shu, Xinjian Li, Katsutoshi Ohtsuki, and Hideki Nakayama. 2018. Real-time neural-based input method. *CoRR*, abs/1810.09309.

Sen Zhang and Yves Laprie. 2003. Text-to-pinyin conversion based on contextual knowledge and d-tree for mandarin. In *IEEE International Conference on Natural Language Processing and Knowledge Engineering, NLP-KE 2003, Beijing, China, 2003*.

Zhuosheng Zhang, Yafang Huang, and Hai Zhao. 2019a. Open vocabulary learning for neural chinese pinyin IME. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 1584–1594. Association for Computational Linguistics.

Zhuosheng Zhang, Zhen Meng, and Hai Zhao. 2019b. A smart sliding chinese pinyin input method editor on touchscreen.