PIANO: An End-to-End Chinese Input Method

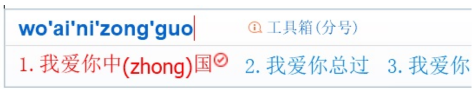
Anonymous ACL submission

Abstract

A Chinese Input Method Engine (IME) 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. We propose **PIANO** (**P**inyin b**I**directional non-**A**uto-regressive n**O**ise-robust Transformers) to solve the error propagation problem effectively and improve the IME engine performance significantly in experiments. Moreover, we model the user real input behaviors and design a method to generate the massive training corpus with typos for the K2C task. It further improves the robustness of PIANO. Finally, we design a non-autoregressive (NAR) decoder for PIANO and obtain 9x+ acceleration with limited performance degradation, which makes it possible to deploy 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


wo'ai'ni'zong'guo

1. 我爱你中(zhong)国 2. 我爱你总过 3. 我爱你

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

user to type in pinyin through the keyboard. And the input software converts the pinyin into the character sequence. As 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 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 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

[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/

[5] The screenshot is from Sogou Input Method software

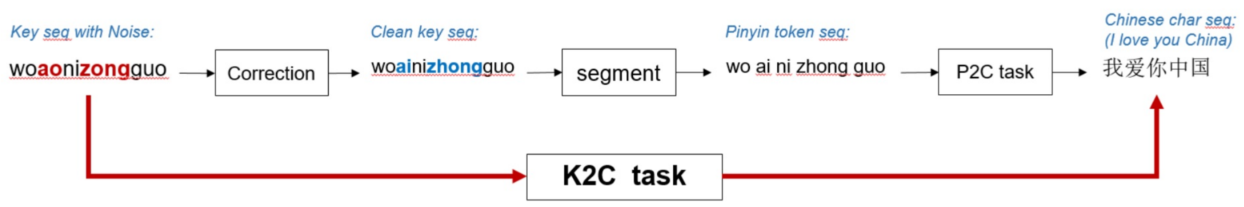


Figure 2: The Key to Character Conversion Task

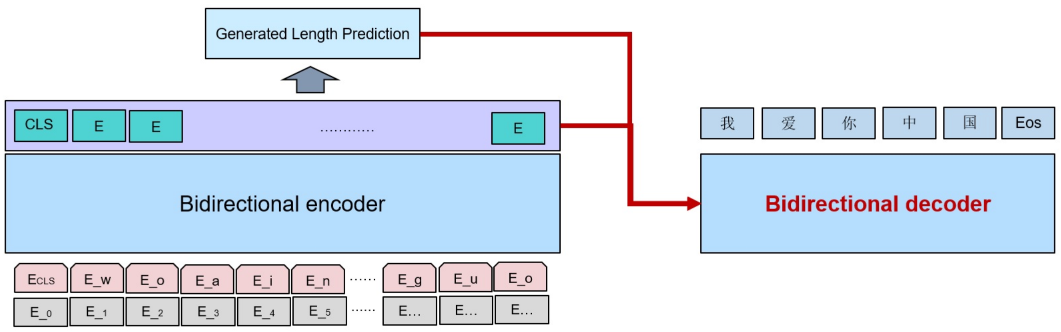


Figure 3: PIANO Model Architecture. In the input layer, E_0 and E_1 are the position embeddings; E_w and E_o are the input token embeddings. The decoder of PIANO adopts the bidirectional attentions. An additional length predictor is added on the top of encoder to guide the generation process.

end-to-end process named the Key-to-Character (K2C) conversion task and proposes PIANO to solve it. As 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 PIANO 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 improves the robustness of PIANO.
- We adopt the NAR decoder for PIANO and boost the inference speed significantly with only little performance degradation.

2 Method

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

2.1 The K2C Conversion Task

As illustrated in 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 to 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 PIANO in this paper.

2.2 PIANO

We build PIANO based on the standard encoder-decoder Transformer architecture (Vaswani et al., 2017) like MASS (Song et al., 2019), T5 (Raffel et al., 2019) and BART (Lewis et al., 2020). To fit for the K2C task, we make some customizations in several aspects, including the training paradigm as described in Section 2.2.1, the embedding layer as described in Section 2.2.2 and the NAR decoder described in Section 2.2.3.

2.2.1 The Training Paradigm

Currently, most of the Transformer models adopt the pretrain-then-finetune paradigm to solve the NLP tasks (Song et al., 2019; Raffel et al., 2019; Lewis et al., 2020). It firstly pre-trains the model on the massive unlabeled corpus by some self-supervised learning tasks, for example, reconstructing text from it noisy version by token masking, token deleting, text infilling, sentence permutation and so on. Then the model is fine-tuned 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 pre-train process leverages the general knowledge contained in the unlabeled corpus which boosts the performance significantly on the target tasks. As described in Section 2.3 later, we design the method to create the massive labelled corpus for the K2C task automatically. Therefore, we train PIANO directly on the target K2C task instead of the pretrain-then-finetune paradigm.

2.2.2 The Embedding Layer

Some pre-trained language models (Devlin et al., 2019; Cui et al., 2019; Sun et al., 2020) adopt segment embedding in its input layer so as to pre-train on the sentence-level tasks. However, as Figure 3 shows, there is no segment embedding in PIANO because there is no pretrain process in PIANO. Besides, PIANO takes the keystroke sequence as input rather than the subword sequence. There are only 26 individual letters which is three order of magnitude smaller than the number of subword (usually more than 50,000) used by pre-trained language model. Thus the size of embedding layer of PIANO is much smaller. In summary, the parameter number of PIANO is usually smaller than the pre-trained language models of the same scale.

2.2.3 The NAR Decoder

PIANO adopts the NAR decoder instead of the autoregressive (AR) decoder as Transformer does. The IME software is usually deployed on edge device whose resources are very limited. The AR decoder makes the assumption on the token dependence which costs much computation and causes big latency. It hinders its deployment. However, the NAR decoder breaks that assumption and is proposed in the machine translation domain(Gu et al., 2018). It can fully leverage the parallel computation so as to accelerate the inference speed greatly. Thus, we adopt the NAR decoder so as to make PIANO deployable.

Moreover, the length information of target sequence is proven to be helpful to the performance of the NAR decoder in the machine translation task (Lee et al., 2018; Gu and Kong, 2021). Inspired by that, we add a length predictor to guide the generation process of PIANO, as described in Figure 3. Specifically, we add a mean pooling layer stacked with a regression layer on the top of the encoder. Then we co-train the PIANO model with two tasks: 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 Formula 1.

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

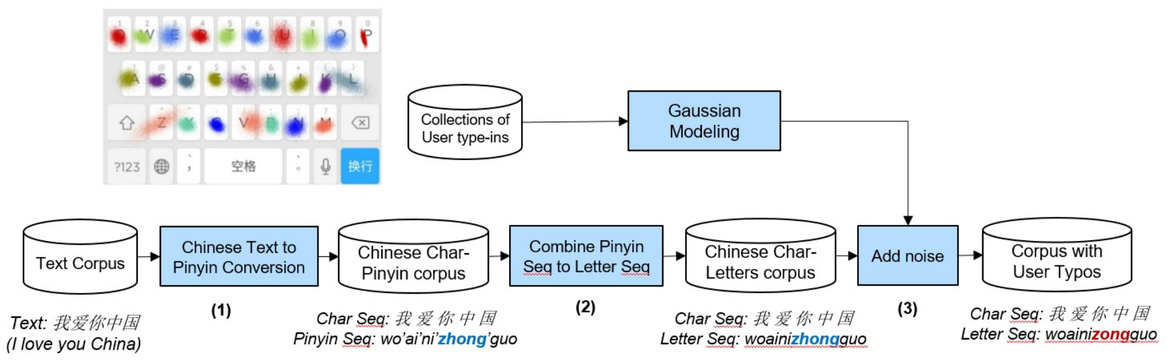
During the inference, the tokens in the target sequence are generated parallel, and the target length is predicted as well. The length is rounded off from float to the integer value. Then the target sequence is simply truncated by that length.

2.3 Generating Massive Labelled Corpus

We generate the massive labelled corpus for the K2C task. It is described in 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′zhong′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′zhong′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



Collections of User type-ins
Gaussian Modeling
Text Corpus
Chinese Text to Pinyin Conversion
Chinese Char-Pinyin corpus
Combine Pinyin Seq to Letter Seq
Chinese Char-Letters corpus
Add noise
Corpus with User Typos
Text: 我爱你中国
(I love you China)
(1)
Char Seq: 我 爱 你 中 国
Pinyin Seq: wo'ai'ni'zhong'guo
(2)
Char Seq: 我 爱 你 中 国
Letter Seq: woainizhongguo
(3)
Char Seq: 我 爱 你 中 国
Letter Seq: woainizongguo

Figure 4: Prepare the Massive Labelled Corpus for the K2C Task. From the point cloud of top left, it shows the scope and dense of user clicks for each button. Different color helps distinguish from each other.

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 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 Formula 2 shows below:

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

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 Chinese news websites. We firstly segment them into sentences by the punctuation list including comma, period, and so on. Then we filter the character which can not be mapped into any pinyin token, such as punctuation and English word. 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 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 Table 1.

[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

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

- **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 Figure 2 works perfectly. It looks like "我爱你中国 (wo′ai′ni′zhong′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 in 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′a′in′i′zong′guo)". We can get it by re-segmenting the sequence containing noises after the third (last) step of of 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 **C**haracter **E**rror **R**ate (CER) to evaluate the performance of the IME engine. It is calculated by comparing the hypothesis sentence with the reference sentence as described in Formula 3.

$$CER = \frac{S + D + I}{N} \tag{3}$$

S, D and I are the character numbers of Substitution, Deletion and Insertion operations respectively. N is the total character number in the reference sentence. The lower the CER is, the better the IME engine performs[7].

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 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* [8] 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 our 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.

- **Transformer**. We use the standard Transformer 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 PIANO model, the keystroke sequence is taken as input. It is trained directly on the K2C task as described in Section 2.2.1, both on the clean corpus and on the noisy corpus generated in Section 2.3. The experimental settings are exactly the same as the standard Transformer baseline. In 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 Table 2. Two ratios of typo noises (1% and 5%) are added into the test 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 10.67% CER and the LSTM model gets a better result of 7.92% (2.75% ↓) which is consistent to the conclusion in the previous articles (Zhang et al., 2019b). The standard Transformer model achieves 2.13% which outperforms the above two models (8.54% ↓ and 5.79% ↓) significantly. It proves that language model plays a crucial role in the cascaded IME engine and its capacity can improve the performance greatly. Besides, we also present the performance of the end-to-end approach. $PIANO_{vanilla}$ (3.72%) also outperforms Bigram and LSTM significantly as Transformer does. However, it performs worse than Transformer. It is because the K2C task contains the additional process of keystroke sequence segmentation implicitly, and it's harder than the P2C task which the Transformer model does.

Secondly, the performance of the cascaded IME engine decreases badly in the noisy environment. Taking the Bigram model as an example, the CER increases from 10.67% to 12.56% (1.89% ↑) under the segment errors, and further to 16.80%

[7] We use fastwer (https://github.com/kahne/fastwer) to calculate CER in experiments

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

Model	Typo Error	Segment Error	CER	Error Reduction
Bigram	no	no	10.67%	NA
Bigram	no	yes	12.56%	1.89%↑
Bigram	1%	yes	16.80%	6.13%↑
Bigram	5%	yes	32.13%	21.46%↑
LSTM	no	no	7.92%	2.75%↓
LSTM	no	yes	9.72%	1.8%↑
LSTM	1%	yes	13.18%	5.26%↑
LSTM	5%	yes	25.31%	17.39%↑
Transformer	no	no	2.13%	8.54%↓
Transformer	no	yes	4.15%	2.02%↑
Transformer	1%	yes	7.89%	5.76%↑
Transformer	5%	yes	21.31%	19.18%↑
$PIANO_{vanilla}$	no	no	3.72%	6.95%↓
$PIANO_{vanilla}$	1%	yes	6.62%	1.27%↓
$PIANO_{vanilla}$	5%	yes	18.11%	3.20%↓
$PIANO_{uni}$	1%	yes	5.13%	2.76%↓
$PIANO_{uni}$	5%	yes	7.80%	13.51%↓
$PIANO$	1%	yes	3.87%	4.02%↓
$PIANO$	5%	yes	5.18%	16.13%↓

Table 2: The Experimental Results on the K2C Task. $PIANO_{vanilla}$ is the PIANO model trained on the clean corpus without any noise. $PIANO_{uni}$ is trained on the corpus with uniform noise. $PIANO$ is trained on the corpus with the noise generated by user model as described in Section 2.3.

(6.13% ↑) under the typo errors as well, and lastly to 32.13% (21.46% ↑) as the typo ratio increases. The similar results can be observed in the LSTM model and even in the powerful Transformer model. It indicates that errors are accumulated and propagated in the cascaded IME system and degrade its performance badly.

Thirdly, the performance of $PIANO_{vanilla}$ also decreases in the noisy environment. However, its declining degree is smaller than the above models, especially smaller than Transformer. For example, the error rate of Transformer is 7.89% under the condition of 1% typos and segment errors, whereas $PIANO_{vanilla}$ performs 6.62% which is much smaller (1.27% ↓). Considering Transformer performs better under the clean environment (2.13%) than $PIANO_{vanilla}$ (3.72%), the performance declining degree of $PIANO_{vanilla}$ in the noisy environment is smaller further. It proves that the end-to-end process makes PIANO perform more robust than the cascaded models do.

Fourthly, $PIANO_{uni}$ and $PIANO$ perform much better than Transformer as well as $PIANO_{vanilla}$ in the noisy environment. For example, under the condition of 1% typos and segment errors, $PIANO_{uni}$ gets 5.13% error rate which is much lower than Transformer (7.89%, 2.76% ↓) and $PIANO_{vanilla}$ (6.62%, 1.49% ↓). The error reduction becomes larger significantly as the ratio of typos increases. It proves that the method generating massive corpus with noise described in Section 2.3 can make our model robust further.

Lastly, $PIANO$ gets the lowest error rate. For example, under the condition of 1% typos and segment errors, $PIANO$ gets 3.87% error rate which is lower than Transformer (7.89%, 4.02% ↓), $PIANO_{vanilla}$ (6.62%, 2.75% ↓) and especially lower than $PIANO_{uni}$ (5.13%, 1.26% ↓). It becomes more obviously as the ratio of typos increases. It proves that the way we models users' input behavior described in Section 2.3 can help to generate high quality typos and improve the robustness of $PIANO$ further.

3.5 Effectiveness of the NAR Decoder

In this section, we compare the performances of PIANOs with the AR decoder and with the NAR decoder. The error rate and the inference speed are reported in Table 3.

Compared to $PIANO_{AR}$, the error rate of $PIANO_{NAR}$ increases by 0.03% under the 1%

Model	Typo Error	Segment Error	CER	Reduction	ms/token	Speedup
$PIANO_{AR}$	1%	yes	3.87%	NA	15.66	NA
$PIANO_{AR}$	5%	yes	5.18%	NA	15.66	NA
$PIANO_{NAR}$	1%	yes	3.90%	-0.03%↑	1.60	9.78x↑
$PIANO_{NAR}$	5%	yes	5.95%	-0.77%↑	1.73	9.30x↑

Table 3: Comparison between Autoregressive PIANO and Non-autoregressive PIANO. $PIANO_{AR}$ is the PIANO with the autoregressive decoder as the standard Transformer does. $PIANO_{NAR}$ is the PIANO with the non-autoregressive decoder as described in Section 2.2.3

Model	Typo Error	Segment Error	CER	Error Reduction
$PIANO_{-LP}$	1%	yes	4.21%	NA
$PIANO_{-LP}$	5%	yes	7.69%	NA
$PIANO_{+LP}$	1%	yes	3.90%	0.31%↓
$PIANO_{+LP}$	5%	yes	5.95%	1.74%↓

Table 4: Effectiveness of Length Predictor. $PIANO_{-LP}$ is the PIANO model without length predictor. $PIANO_{+LP}$ is the PIANO model with length predictor.

typo ratio, and further by 0.77% under the 5% typo ratio. Considering the fact that the performance of PIANO is already good enough, that performance degradation is very slightly. Here our conclusion on the K2C task is somehow contrary to those in machine translation in which the performances of NAR models are severely degraded (Gu et al., 2018; Lee et al., 2018; Gu and Kong, 2021). It's because that the order of tokens in the target sequence roughly corresponds to the source sequence, which makes the task simpler. Whereas, in machine translation, the token order correspondence can not be guaranteed, and the correct translation heavily relies on the dependence between tokens in the target sequence. The NAR decoder breaks that dependence which makes it a much harder task.

However, the inference process is accelerated greatly by $PIANO_{NAR}$. The time to infer per token drops from $15.66ms$ to $1.60ms$ which is accelerated by 9.78 times under the 1% typo ratio, and drops from $16.09ms$ to $1.73ms$ which is accelerated by 9.30 times under the 5% typo ratio. It makes the deployment possible to the commercial input method software.[9]

[9]The industry usually requires that the inference latency less than one millisecond per token. As reported by the papers of related techniques, such as distillation (Jiao et al., 2020), quantization (Zhao et al., 2021; Zhang et al., 2020; Qin et al., 2022) and pruning (Zafrir et al., 2021), PIANO can easily meet that requirement after applying those techniques. Moreover, there is open tools such as (https://github.com/huawei-noah/bolt) to help. It's our future work to deploy it in the real product.

3.6 Ablation Study on the Length Predictor

In this section, we compare the model performance with or without the length predictor. The experimental results are presented in Table 4.

$PIANO_{+LP}$ achieves much lower error rates than $PIANO_{-LP}$. It becomes more obvious as the ratio of typos increases. It proves that the length predictor module can effectively improve the performance and is necessary to $PIANO$.

4 Related Works

4.1 Input Method Engine

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 Ngram with LSTM 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 efficiency problem of LSTM 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 learning 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 RNN 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.

4.3 Error Propagation Problem in Cascaded Systems

There are a lot of articles discussing the performance in the noisy environment and how to handle the error propagation problem in the cascaded system, including the QA system (Ravichander et al., 2021), speech translation (Cheng et al., 2019; Belinkov and Bisk, 2018), spoken language understanding (Chen et al., 2021), machine translation (Li et al., 2018) and so on. Usually, the noises degrade the performances badly. The most natural way to solve the problem is to train the model based on the corpus with noises. It firstly analysis the source of noise, and then synthesize the similar noise to inject into the training corpus. For example, it exchanges words randomly in sequence to synthesize the random noise; it uses a TTS system pipelined with an ASR system to generate the ASR noise; it introduces typos based on the proximity of the keys in a QWERTY keyboard layout (Ravichander et al., 2021). PIANO takes the similar way. However, different from the above works, we model users' real behaviors on keyboard and generate high quality typos which improves the performance further as presented in Section 3.4.

Besides data augmentation, some works designs specific network architecture and training process. Belinkov and Bisk (2018) designs the structure-invariant word representation to increase model robustness. Cheng et al. (2019) adopts the adversarial learning to address encoder and decoder simultaneously in its training process of speech translation system. Chen et al. (2021) utilizes the phonetic information and designs a joint text-phonetic pre-training tasks to improve the robustness of the end-to-end spoken language understanding system. Similar to the above works, PIANO combines each component of pipeline system into a unified K2C task and train the model in an end-to-end way so as to improve its robustness.

5 Conclusions

In this paper, we propose the K2C conversion task and design PIANO to build the IME engine in an end-to-end way. Compared with the cascaded IME engine, PIANO 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 NAR decoder adopted in PIANO can accelerate the inference speed greatly with little performance degradation.

6 Limitations and Future Works

In the future, we are going to deploy PIANO into the commercial input software and improve the user experiences. There are totally 1.6 billion Chinese people who has to type in their words by the IME software. According to the statistics from iFLYTEK input method, the total number of Chinese input characters from its customers in one year exceeds 10.5 trillion [10]. Thus our technique can make a huge impact on the daily life of people. Not to mention the people of other Asian countries, i.e. Japanese, Thai, and so on.

[10]https://m.mydrivers.com/newsview/665433.html

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.

Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break 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.

Qian Chen, Wen Wang, and Qinglin Zhang. 2021. Pre-training for spoken language understanding with joint textual and phonetic representation learning. In *Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association, Brno, Czechia, 30 August - 3 September 2021*, pages 1244–1248. ISCA.

Qiao Cheng, Meiyuan Fang, Yaqian Han, Jin Huang, and Yitao Duan. 2019. Breaking the data barrier: Towards robust speech translation via adversarial stability training. In *Proceedings of the 16th International Conference on Spoken Language Translation, IWSLT 2019, Hong Kong, November 2-3, 2019*. Association for Computational Linguistics.

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.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. 2019. Pre-training with whole word masking for chinese BERT. *CoRR*, abs/1906.08101.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

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.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4163–4174. 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.

Xiang Li, Haiyang Xue, Wei Chen, Yang Liu, Yang Feng, and Qun Liu. 2018. Improving the robustness of speech translation. *CoRR*, abs/1811.00728.

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.

Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. 2022. Bibert: Accurate fully binarized BERT. *CoRR*, abs/2203.06390.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

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.

Abhilasha Ravichander, Siddharth Dalmia, Maria Ryskina, Florian Metze, Eduard H. Hovy, and Alan W. Black. 2021. Noiseqa: Challenge set evaluation for user-centric question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 2976–2992. Association for Computational Linguistics.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: masked sequence to sequence pre-training for language generation. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936. PMLR.

Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. ERNIE 2.0: A continual pre-training framework for language understanding. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8968–8975. AAAI Press.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

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.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021. Prune once for all: Sparse pre-trained language models. *CoRR*, abs/2111.05754.

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*.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 509–521. Association for Computational Linguistics.

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.

Changsheng Zhao, Ting Hua, Yilin Shen, Qian Lou, and Hongxia Jin. 2021. Automatic mixed-precision quantization search of BERT. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 3427–3433. ijcai.org.