AN ABSTRACT OF THE THESIS OF

Renjie Zheng for the degree of Doctor of Philosophy in Computer Science presented on March 30, 2020.

Title: Advances in Simultaneous Translation

Abstract approved: _____

Liang Huang

Simultaneous translation, which translates concurrently with the source language speech, is widely used in many scenarios including multilateral organizations. However, it is well known to be one of the most challenging tasks for humans due to the simultaneous perception and production in two languages. On the other hand, simultaneous translation is also notoriously difficult for machines and has remained one of the holy grails of AI. The key challenge is the word order difference between the source and target languages. There have been efforts towards genuine simultaneous translation, but all these efforts have the following major limitations: (a) none of them can achieve any arbitrary given latency; (b) their base translation model is still trained on full sentences; and (c) their systems are complicated, involving many components and are difficult to train. In this thesis, we start by introducing several simultaneous translation approaches with two orthogonal categories: fixed or adaptive latency policies; trained on full sentences or not. Then, we investigate how to improve simultaneous translation with beam search which

is universally used in full-sentence translation but non-trivial to be applied in simultaneous translation. Finally, we explore speech-to-speech simultaneous interpretation by incorporating streaming ASR and incremental TTS.

Advances in Simultaneous Translation

by

Renjie Zheng

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented March 30, 2020
Commencement June 2020

Doctor of Philosophy thesis of Renjie Zheng presented on March 30, 2020.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Renjie Zheng, Author

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Liang Huang for the continuous support and guidance of my Ph.D study and related research. I could not imagine having a better advisor and mentor for my Ph.D study. Some of my work in this thesis was supported in part by NSF grants IIS-1817231 and IIS-1656051. Additionally, I was also supported by DARPA grant N66001-17-2-4030 for another project [Zheng et al., 2018b] but it is not part of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Prasad Tadepalli, Prof. Xiaoli Fern, Prof. Lizhong Chen, Prof. Brett Tyler and Prof. Alan Fern, for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives.

I thank my fellow labmates for the stimulating discussions, and for all the fun we have had in the past years. They are Mingbo Ma, Baigong Zheng, Kaibo Liu, Junkun Chen, Juneki Hong, Göksu Öztürk Miraç, Liang Zhang, He Zhang and Sizhen Li.

Last but not the least, I would like to thank my family: my wife and my parents for supporting me spiritually throughout writing this thesis and my life in general.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF ALGORITHMS

Chapter 1: Introduction

Simultaneous translation, which translates sentences before they are finished, is useful in many scenarios including multilateral organizations (UN/EU), and international summits (APEC/G-20). However, it is widely considered one of the most challenging tasks in NLP, and one of the holy grails of AI [Grissom II et al., 2014]. There have been efforts towards genuine simultaneous translation, but unfortunately none of them can achieve any arbitrary given latency and their base translation models are still trained on full sentences. In this thesis, we first present several simultaneous translation approaches based on the prefix-to-prefix framework which overcomes the limitations of previous work (Chapt. 2). Then we investigate a beam search algorithm which is able to further improve the translation quality of different simultaneous translation models (Chapt. 3). In the end, we introduce an incremental Text-to-speech model which is a critical component of our final speech-to-speech simultaneous translation system (Chapt. 4).

1.1 Background: Simultaneous Interpretation

There are two modes of human interpretation, consecutive and simultaneous. While consecutive interpretation waits until the speaker pauses (usually at sentence boundaries) to initiate translation, simultaneous interpretation translates concurrently with the source-language speech, with a delay of only a few seconds. This *additive* latency is much more

desirable than the *multiplicative* 2× slowdown in consecutive interpretation.

With this appealing property, simultaneous interpretation has been widely used in many scenarios. However, due to the concurrent comprehension and production in two languages, it is extremely challenging and exhaustive for humans: the number of qualified simultaneous interpreters worldwide is very limited, and each can only last for about 15-30 minutes in one turn, whose error rates grow exponentially after just minutes of interpreting [Moser-Mercer et al., 1998]. Moreover, limited memory forces human interpreters to routinely omit source content [He et al., 2016]. Therefore, there is a critical need to develop simultaneous machine translation techniques to reduce the burden of human interpreters and make it more accessible and affordable.

1.2 Existing Methods in Simultaneous Translation

Unfortunately, simultaneous translation is also notoriously difficult for machines, due in large part to the diverging word order between the source and target languages. For example, think about simultaneously translating an SOV language such as Japanese or German to an SVO language such as English or Chinese:[1] you have to wait until the source language verb. As a result, existing so-called "real-time" translation systems resort to conventional full-sentence translation, causing an undesirable latency of at least one sentence. Some researchers, on the other hand, have noticed the importance of verbs in SOV→SVO translation [Grissom II et al., 2016], and have attempted to reduce latency by explicitly predicting the sentence-final German [Grissom II et al., 2014] or English verbs

[1] Technically, German is SOV+V2 in main clauses, and SOV in embedded clauses; Mandarin is a mix of SVO+SOV.

[Matsubarayx et al., 2000], which is limited to this particular case, or unseen syntactic constituents [Oda et al., 2015, He et al., 2015], which requires incremental parsing on the source sentence. Some researchers propose to translate on an optimized sentence segment level to get better translation accuracy [Oda et al., 2014, Fujita et al., 2013, Bangalore et al., 2012]. More recently, Gu et al. [2017] propose a two-stage model whose base model is a full-sentence model, On top of that, they use a READ/WRITE (R/W) model to decide, at every step, whether to wait for another source word (READ) or to emit a target word using the pretrained base model (WRITE), and this R/W model is trained by reinforcement learning to prefer (rather than enforce) a specific latency, without updating the base model. All these efforts have the following major limitations: (a) none of them can achieve any arbitrary given latency such as "3-word delay"; (b) their base translation model is still trained on full sentences; and (c) their systems are complicated, involving many components (such as pretrained model, prediction, and RL) and are difficult to train.

1.3 Our Proposed Methods

In this thesis, we will introduce our work in simultaneous translation to tackle the previous problems. We first present a very simple yet effective solution, designing a novel prefix-to-prefix framework that predicts target words using only prefixes of the source sentence. Within this framework, we study a special case, the "wait-k" policy [Ma et al., 2019a, 2020], whose translation is always k words behind the input. Consider the Chinese-to-English example in Figs. 1.1–1.2, where the translation of the sentence-final

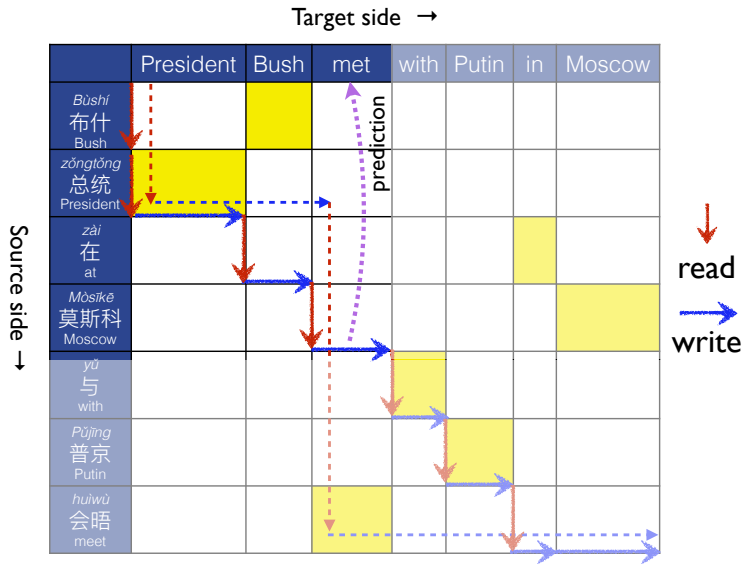


Figure 1.1: Our wait-k model emits target word y_t given source-side prefix $x_1 \ldots x_{t+k-1}$, often before seeing the corresponding source word (here k=2, outputing y_3="met" before x_7="*huìwù*"). Without anticipation, a 5-word wait is needed (dashed arrows). See also Fig. 1.2.

	Bùshí	***zǒngtǒng***	***zài***	Mòsīkē	*yǔ*	*Pǔjīng*	*huìwù*	
	布什	总统	在	莫斯科	与	普京	会晤	
	Bush	**president**	**in**	**Moscow**	with/and	Putin	meet	
(a) simultaneous: our wait-2	*...wait 2 words...*		pres.	bush	**met**	with	putin	in moscow
(b) non-simultaneous baseline	*..... wait whole sentence .....*							pres. bush met with putin in moscow
(c) simultaneous: test-time wait-2	*...wait 2 words...*		pres.	bush	in	moscow	and	pol- ite meeting
	布什	总统			在	莫斯科	与	普京 会晤
(d) simultaneous: non-predictive	*...wait 2 words...*		pres.	bush	*..... wait 5 words .....*			met with putin in moscow

Figure 1.2: Another view of Fig. 1.1, highlighting the prediction of English "met" corresponding to the sentence-final Chinese verb *huìwù*. (a) Our wait-k policy (here $k = 2$) translates concurrently with the source sentence, but always k words behind. It correclty predicts the English verb given just the first 4 Chinese words (in bold), lit. "Bush president in Moscow", because it is trained in a prefix-to-prefix fashion (Sec. 2.1), and the training data contains many prefix-pairs in the form of (X *zài* Y ..., X met ...). (c) The test-time wait-k decoding (Sec. 2.1.0.2) using the full-sentence model in (b) can not anticipate and produces nonsense translation. (d) A simultaneous translator without anticipation has to wait 5 words.

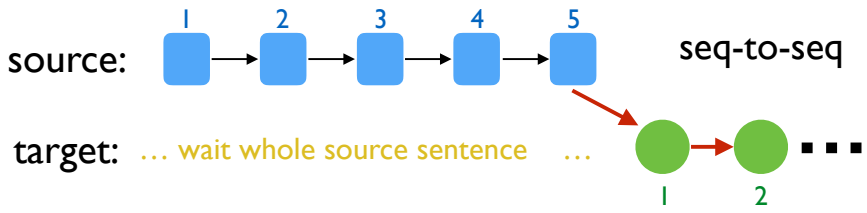


Figure 1.3: Sequence-to-sequence framework.

Chinese verb *huìwù* (“meet”) needs to be emitted earlier to avoid a long delay. Our wait-2 model correctly anticipates the English verb given only the first 4 Chinese words (which provide enough clue for this prediction given many similar prefixes in the training data).

Besides this wait-k fixed policy, we also proposed a flexible policy learned on pre-trained model [Zheng et al., 2019b] and a flexible policy learned from scratch [Zheng et al., 2019a, 2020a]. To further improves simultaneous translation quality, we propose the speculative beam search algorithm [Zheng et al., 2019d, Ma et al., 2019c, Zheng et al., 2020b] which can also be used to do timely correction. In the end, we will introduce an incremental Text-to-speech model which can be used in speech-to-speech simultaneous interpretation [Ma et al., 2019b].

1.4 Preliminaries

Neural machine translation has attracted much attention in recent years thanks to its impressive generation accuracy and wide applicability. Sequence-to-sequence (seq2seq) models based on RNNs [Sutskever et al., 2014, Bahdanau et al., 2014a] , CNNs [Gehring et al., 2017] and self-attention [Vaswani et al., 2017] have achieved great successes in Neural Machine Translation (NMT). The above family of models encode the source

sentence and predict the next word in an autoregressive fashion at each decoding time step. The classical "cross-entropy" training objective of seq2seq models is to maximize the likelihood of each word in the translation reference given the source sentence and all previous words in that reference. This word-level loss ensures efficient and scalable training of seq2seq models.

Regardless of the particular design of different seq-to-seq models, the encoder always takes the input sequence $\boldsymbol{x} = (x_1, ..., x_n)$ where each $x_i \in \mathbb{R}^{d_x}$ is a word embedding of d_x dimensions, and produces a new sequence of hidden states $\mathbf{h} = f(\boldsymbol{x}) = (h_1, ..., h_n)$. The encoding function f can be implemented by RNN or Transformer.

On the other hand, a (greedy) decoder predicts the next output word y_t given the source sequence (actually its representation $\mathbf{h}$) and previously generated words, denoted $\boldsymbol{y}_{<t} = (y_1, ..., y_{t-1})$. The decoder stops when it emits <eos>, and the final hypothesis $\boldsymbol{y} = (y_1, ..., \texttt{<eos>})$ has probability

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{y}|} p(y_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}) \tag{1.1}$$

At training time, we maximize the conditional probability of each ground-truth target sentence $\boldsymbol{y}^\star$ given input $\boldsymbol{x}$ over the whole training data D, or equivalently minimizing the following loss:

$$\ell(D) = -\sum_{(\boldsymbol{x}, \boldsymbol{y}^\star) \in D} \log p(\boldsymbol{y}^\star \mid \boldsymbol{x}) \tag{1.2}$$

Chapter 2: Policies and Models for Simultaneous Translation

2.1 Prefix-to-Prefix and Wait-k Policy

To overcome the drawbacks in previous work, we instead present a very simple yet effective solution, designing a novel prefix-to-prefix framework that predicts target words using only prefixes of the source sentence. Within this framework, we study a special case, the "wait-k" policy, whose translation is always k words behind the input. Consider the Chinese-to-English example in Figs. 1.1–1.2, where the translation of the sentence-final Chinese verb *huìwù* ("meet") needs to be emitted earlier to avoid a long delay. Our wait-2 model correctly anticipates the English verb given only the first 4 Chinese words (which provide enough clue for this prediction given many similar prefixes in the training data).

In full-sentence translation ,each y_i is predicted using the entire source sentence $\boldsymbol{x}$.

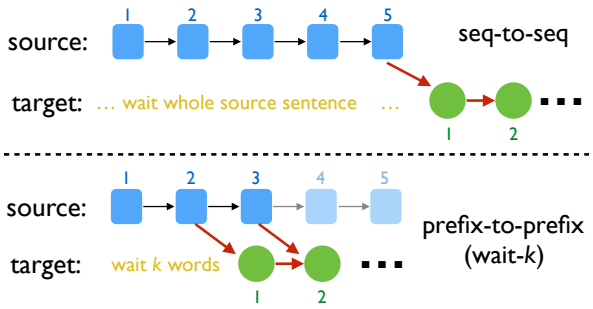


Figure 2.1: Seq-to-seq vs. our prefix-to-prefix frameworks (showing wait-2 as an example).

But in simultaneous translation, we need to translate concurrently with the (growing) source sentence, so we design a new prefix-to-prefix architecture to (be trained to) predict using a source prefix.

2.1.0.1 Prefix-to-Prefix Architecture

Definition 1. Let $g(t)$ be a **monotonic non-decreasing** function of t that denotes the number of source words processed by the encoder when deciding the target word y_t.

For example, in Figs. 1.1–1.2, $g(3) = 4$, i.e., a 4-word Chinese prefix is used to predict y_3="met". We use the source prefix $(x_1, ..., x_{g(t)})$ rather than the whole $\boldsymbol{x}$ to predict y_t: $p(y_t \mid \boldsymbol{x}_{\leq g(t)}, \boldsymbol{y}_{<t})$. Therefore the decoding probability is:

$$p_g(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{y}|} p(y_t \mid \boldsymbol{x}_{\leq g(t)}, \boldsymbol{y}_{<t}) \tag{2.1}$$

and given training D, the training objective is:

$$\ell_g(D) = -\sum_{(\boldsymbol{x}, \boldsymbol{y}^\star) \in D} \log p_g(\boldsymbol{y}^\star \mid \boldsymbol{x}) \tag{2.2}$$

Generally speaking, $g(t)$ can be used to represent any arbitrary policy, and we give two special cases where $g(t)$ is constant: (a) $g(t) = |\boldsymbol{x}|$: baseline full-sentence translation; (b) $g(t) = 0$: an "oracle" that does not rely on any source information. Note that in any case, $0 \leq g(t) \leq |\boldsymbol{x}|$ for all t.

Definition 2. We define the **"cut-off" step**, $\tau_g(|\boldsymbol{x}|)$, to be the decoding step when source

sentence finishes:

$$\tau_g(|\boldsymbol{x}|) = \min\{t \mid g(t) = |\boldsymbol{x}|\} \tag{2.3}$$

For example, in Figs. 1.1–1.2, the cut-off step is 6, i.e., the Chinese sentence finishes right before y_6="in".

Training vs. Test-Time Prefix-to-Prefix. While most previous work in simultaneous translation, in particular Bangalore et al. [2012] and Gu et al. [2017], might be seen as special cases in this framework, we note that only their *decoders* are prefix-to-prefix, while their training is still full-sentence-based. In other words, they use a full-sentence translation model to do simultaneous decoding, which is a mismatch between training and testing. The essence of our idea, however, is to *train* the model to predict using source prefixes. Most importantly, this new training implicitly learns anticipation as a by-product, overcoming word-order differences such as SOV→SVO. Using the example in Figs. 1.1–1.2, the anticipation of the English verb is possible because the training data contains many prefix-pairs in the form of (X *zài* Y ..., X met ...), thus although the prefix $\boldsymbol{x}_{\leq 4}$="*Bùshí zǒngtǒng zài Mòsikē*" (lit. "Bush president in Moscow") does not contain the verb, it still provides enough clue to predict "met".

2.1.0.2 Wait-k Policy

As a very simple example within the prefix-to-prefix framework, we present a wait-k policy, which first wait k source words, and then translates concurrently with the rest of source sentence, i.e., the output is always k words behind the input. This is inspired by human simultaneous interpreters who generally start translating a few seconds into the

speakers' speech, and finishes a few seconds after the speaker finishes. For example, if $k = 2$, the first target word is predicted using the first 2 source words, and the second target word using the first 3 source words, etc.; see Fig. 4.2. More formally, its $g(t)$ is defined as follows:

$$g_{\text{wait-}k}(t) = \min\{k + t - 1,\ |\boldsymbol{x}|\} \tag{2.4}$$

For this policy, the cut-off point $\tau_{g_{\text{wait-}k}}(|\boldsymbol{x}|)$ is exactly $|\boldsymbol{x}| - k$. From this step on, $g_{\text{wait-}k}(t)$ is fixed to $|\boldsymbol{x}|$, which means the remaining target words (including this step) are generated using the full source sentence, similar to conventional MT. We call this part of output, $\boldsymbol{y}_{\geq |\boldsymbol{x}|-k}$, the "tail", and can perform beam search on it (which we call "tail beam search"), but all earlier words are generated greedily one by one (see Appendix).

Test-Time Wait-k. As an example of test-time prefix-to-prefix in the above subsection, we present a very simple "test-time wait-k" method, i.e., using a full-sentence model but decoding it with a wait-k policy (see also Fig. 1.2(c)). Our experiments show that this method, without the anticipation capability, performs much worse than our genuine wait-k when k is small, but gradually catches up, and eventually both methods approach the full-sentence baseline ($k = \infty$).

2.1.1 Latency Metric: Average Lagging

Beside translation quality, latency is another crucial aspect for evaluating simultaneous translation. We first review existing latency metrics, highlighting their limitations, aand then propose our new latency metric that address these limitations.

2.1.1.1 Existing Metrics: CW and AP

Consecutive Wait (CW) [Gu et al., 2017] is the number of source words waited between two target words. Using our notation, for a policy $g(\cdot)$, the per-step CW at step t is $\mathrm{CW}_g(t) = g(t) - g(t-1)$. The CW of a sentence-pair $(\boldsymbol{x}, \boldsymbol{y})$ is the average CW over all consecutive wait segments:

$$\mathrm{CW}_g(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{t=1}^{|\boldsymbol{y}|} \mathrm{CW}_g(t)}{\sum_{t=1}^{|\boldsymbol{y}|} \mathbb{1}_{\mathrm{CW}_g(t)>0}} = \frac{|\boldsymbol{x}|}{\sum_{t=1}^{|\boldsymbol{y}|} \mathbb{1}_{\mathrm{CW}_g(t)>0}}$$

In other words, CW measures the average source segment length (the best case is 1 for word-by-word translation or our wait-1 and the worst case is $|\boldsymbol{x}|$ for full-sentence MT). The drawback of CW is that CW is local latency measurement which is insensitive to the actual lagging behind.

Another latency measurement, Average Proportion (AP) [Cho and Esipova, 2016] measures the proportion of the area above a policy path in Fig. 1.1:

$$\mathrm{AP}_g(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{|\boldsymbol{x}|\ |\boldsymbol{y}|} \sum_{t=1}^{|\boldsymbol{y}|} g(t) \tag{2.5}$$

AP has two major flaws: First, it is sensitive to input length. For example, consider our wait-1 policy. When $|\boldsymbol{x}| = |\boldsymbol{y}| = 1$, AP is 1, and when $|\boldsymbol{x}| = |\boldsymbol{y}| = 2$, AP is 0.75, and eventually AP approaches 0.5 when $|\boldsymbol{x}| = |\boldsymbol{y}| \to \infty$. However, in all these cases, there is a one word delay, so AP is not fair between long and short sentences. Second, being a percentage, it is not obvious to the user the actual delays in number of words.

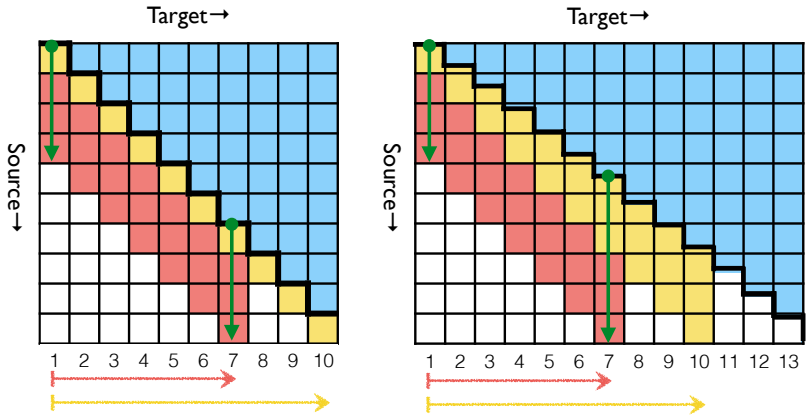


Figure 2.2: Illustration of our proposed Average Lagging latency metric. The left figure shows a simple case when $|\boldsymbol{x}| = |\boldsymbol{y}|$ while the right figure shows a more general case when $|\boldsymbol{x}| \neq |\boldsymbol{y}|$. The red policy is wait-4, the yellow is wait-1, and the thick black is a policy whose AL is 0.

2.1.1.2 New Metric: Average Lagging

Inspired by the idea of "lagging behind the ideal policy", we propose a new metric called "average lagging" (AL), shown in Fig. 2.2. The goal of AL is to quantify the degree the user is out of sync with the speaker, in terms of the number of source words. The left figure shows a special case when $|\boldsymbol{x}| = |\boldsymbol{y}|$ for simplicity reasons. The thick black line indicates the "wait-0" policy where the decoder is alway one word *ahead* of the encoder and we define this policy to have an AL of 0. The diagonal yellow policy is our "wait-1" which is always one word behind the wait-0 policy. In this case, we define its AL to be 1. The red policy is our wait-4, and it is always 4 words behind the wait-0 policy, so its AL is 4. Note that in both cases, we only count up to (but including) the cut-off point (indicated by the horizontal yellow/red arrows, or 10 and 7, resp.) because the tail can be generated instantly without further delay. More formally, for the ideal case where

$|\boldsymbol{x} = |\boldsymbol{y}|$, we can define:

$$\mathrm{AL}_g(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{\tau_g(|\boldsymbol{x}|)} \sum_{t=1}^{\tau_g(|\boldsymbol{x}|)} g(t) - (t-1) \tag{2.6}$$

We can infer that the AL for wait-k is exactly k.

When we have more realistic cases like the right side of Fig. 2.2 when $|\boldsymbol{x}| < |\boldsymbol{y}|$, there are more and more delays accumulated when target sentence grows.For example, for the yellow wait-1 policy has a delay of more than 3 words at decoding its cut-off step 10, and the red wait-4 policy has a delay of almost 6 words at its cut-off step 7. This difference is mainly caused by the tgt/src ratio. For the right example, there are 1.3 target words per source word. More generally, we need to offset the "wait-0" policy and redefine:

$$\mathrm{AL}_g(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{\tau_g(|\boldsymbol{x}|)} \sum_{t=1}^{\tau_g(|\boldsymbol{x}|)} g(t) - \frac{t-1}{r} \tag{2.7}$$

where $\tau_g(|\boldsymbol{x}|)$ denotes the cut-off step, and $r = |y|/|x|$ is the target-to-source length ratio. We observe that wait-k with catchup has an AL $\simeq k$.

2.1.2 Implementation Details

While RNN-based implementation of our wait-k model is straightforward and our initial experiments showed equally strong results, due to space constraints we will only present Transformer-based results. Here we describe the implementation details for training a prefix-to-prefix Transformer, which is a bit more involved than RNN.

2.1.2.1 Background: Full-Sentence Transformer

We first briefly review the Transformer architecture step by step to highlight the difference between the conventional and simultaneous Transformer. The encoder of Transformer works in a self-attention fashion and takes an input sequence $\boldsymbol{x}$, and produces a new sequence of hidden states $\mathbf{z} = (z_1, ..., z_n)$ where $z_i \in \mathbb{R}^{d_z}$ is as follows:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} \; P_{W_{\mathrm{V}}}(x_j) \tag{2.8}$$

Here $P_{W_{\mathrm{V}}}(\cdot)$ is a projection function from the input space to the value space, and α_{ij} denotes the attention weights:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{l=1}^{n} \exp e_{il}}, \;\; e_{ij} = \frac{P_{W_{\mathrm{Q}}}(x_i) P_{W_{\mathrm{V}}}(x_j)^T}{\sqrt{d_x}} \tag{2.9}$$

where e_{ij} measures similarity between inputs. Here $P_{W_{\mathrm{Q}}}(x_i)$ and $P_{W_{\mathrm{K}}}(x_j)$ project x_i and x_j to query and key spaces, resp. We use 6 layers of self-attention and use $\mathbf{h}$ to denote the top layer output sequence (i.e., the source context).

On the decoder side, during training time, the gold output sequence $\boldsymbol{y}^* = (y_1^*, ..., y_m^*)$ goes through the same self-attention to generate hidden self-attended state sequence $\mathbf{c} = (c_1, ..., c_m)$. Note that because decoding is incremental, we let $\alpha_{ij} = 0$ if $j > i$ in Eq. 2.9 to restrict self-attention to previously generated words.

In each layer, after we gather all the hidden representations for each target word

through self-attention, we perform target-to-source attention:

$$c'_i = \sum_{j=1}^{n} \beta_{ij} \; P_{W_{\mathrm{V}'}}(h_j)$$

similar to self-attention, β_{ij} measures the similarity between h_j and c_i as in Eq. 2.9.

2.1.2.2 Training Simultaneous Transformer

Simultaneous translation requires feeding the source words incrementally to the encoder, but a naive implementation of such incremental encoder/decoder is inefficient. Below we describe a faster implementation.

For the encoder, during training time, we still feed the entire sentence at once to the encoder. But different from the self-attention layer in conventional Transformer (Eq. 2.9), we constrain each source word to attend to its predecessors only (similar to decoder-side self-attention), effectively simulating an incremental encoder:

$$\alpha_{ij}^{(t)} = \begin{cases} \frac{\exp e_{ij}^{(t)}}{\sum_{l=1}^{g(t)} \exp e_{il}^{(t)}} & \text{if } \; i, j \leq g(t) \\ 0 & \text{otherwise} \end{cases}$$

$$e_{ij}^{(t)} = \begin{cases} \frac{P_{W_{\mathrm{Q}}}(x_i) \; P_{W_{\mathrm{K}}}(x_j)^T}{\sqrt{d_x}} & \text{if } \; i, j \leq g(t) \\ -\infty & \text{otherwise} \end{cases}$$

Then we have a newly defined hidden state sequence $\mathbf{z}^{(t)} = (z_1^{(t)}, ..., z_n^{(t)})$ at decoding

step t:

$$z_i^{(t)} = \sum_{j=1}^{n} \alpha_{ij}^{(t)} \; P_{W_{\mathrm{V}}}(x_j) \tag{2.10}$$

When a new source word is received, all previous source words need to adjust their representations.

2.1.3 Experiments

2.1.3.1 Datasets and Systems Settings

We evaluate our work on four simultaneous translation directions: German↔English and Chinese↔English. For the training data, we use the parallel corpora available from WMT15[1] for German↔English (4.5M sentence pairs) and NIST corpus for Chinese↔English (2M sentence pairs). We first apply BPE [Sennrich et al., 2015] on all texts in order to reduce the vocabulary sizes. For German↔English evaluation, we use newstest-2013 (dev) as our dev set and newstest-2015 (test) as our test set, with 3,000 and 2,169 sentence pairs, respectively. For Chinese↔English evaluation, we use NIST 2006 and NIST 2008 as our dev and test sets. They contain 616 and 691 Chinese sentences, each with 4 English references. When translating from Chinese to English, we report 4-reference BLEU scores, and in the reverse direction, we use the second among the four English references as the source text, and report 1-reference BLEU scores.

Our implementation is adapted from PyTorch-based OpenNMT [Klein et al., 2017]. Our Transformer is essentially the same as the base model from the original paper

[1] http://www.statmt.org/wmt15/translation-task.html

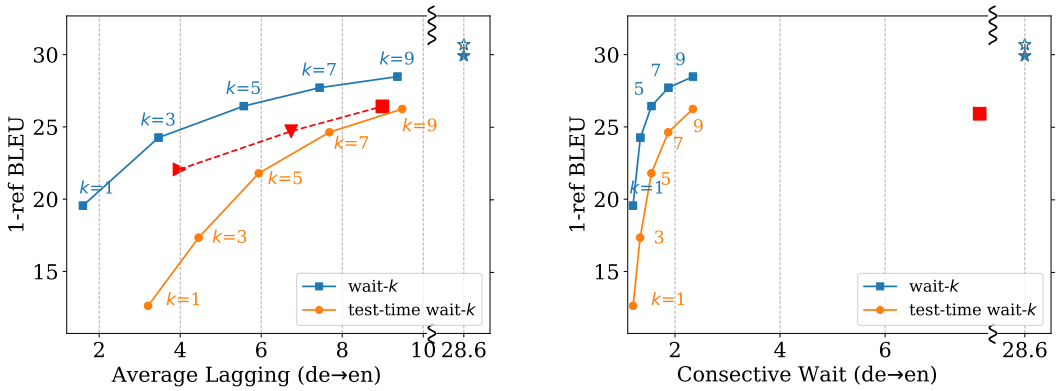


Figure 2.3: Translation quality against latency metrics (AP and CW) on German-to-English simultaneous translation, showing wait-k models (for k=1, 3, 5, 7, 9), test-time wait-k results, full-sentence baselines, and our reimplementation of Gu et al. [2017], all based on the same Transformer. ★☆:full-sentence (greedy and beam-search), Gu et al. [2017]: ▶:CW=2; ▼:CW=5; ■:CW=8.

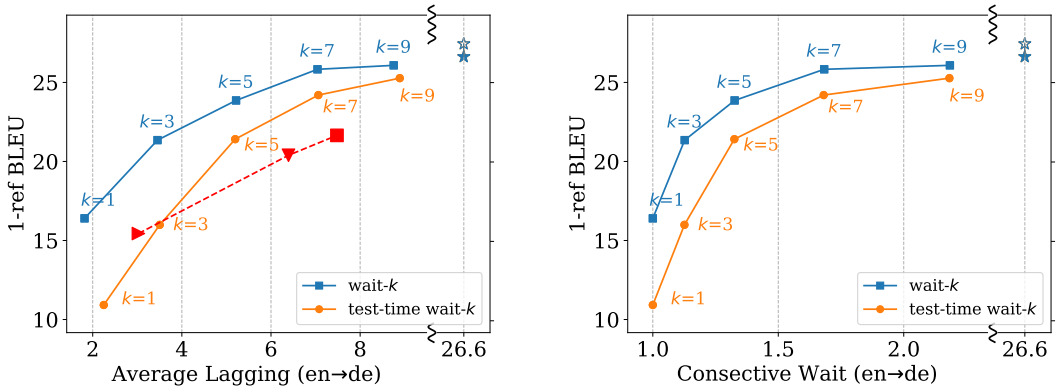


Figure 2.4: Translation quality against latency metrics on English-to-German simultaneous translation. Gu et al. [2017]: ▶:CW=2; ▼:CW=5; ■:CW=8.

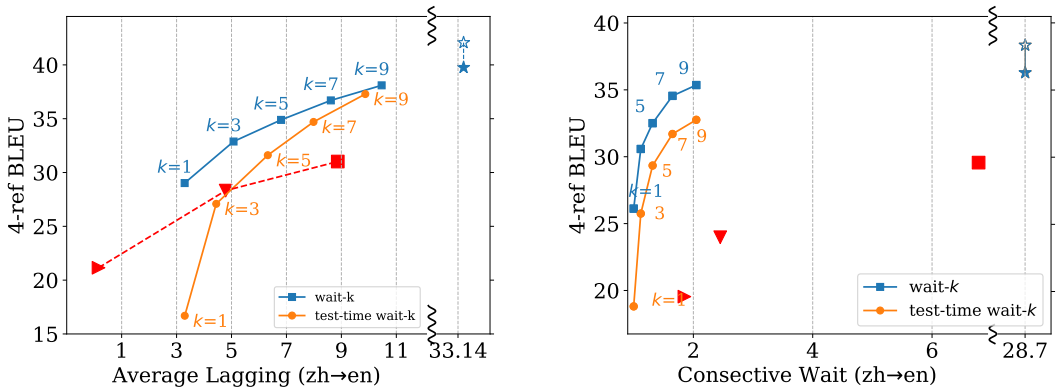


Figure 2.5: Translation quality against latency on zh→en. Gu et al. [2017]: ▶:CW=2; ▼:CW=5; ■:CW=8.

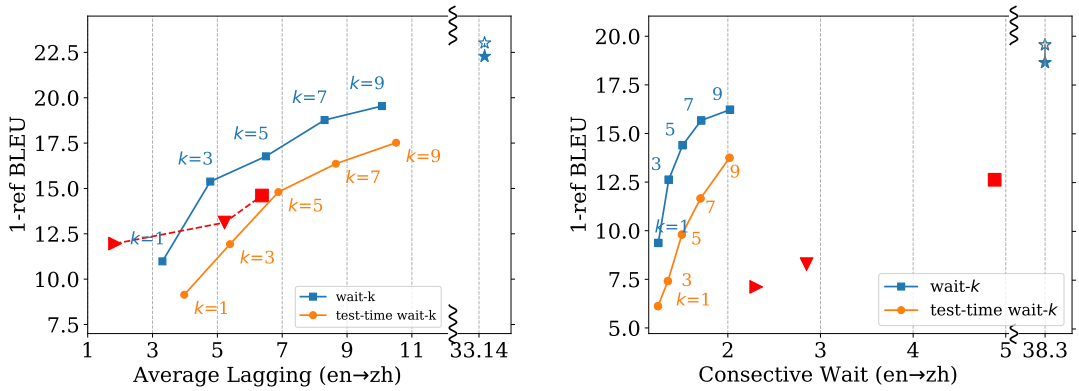


Figure 2.6: Translation quality against latency on en→zh. We use encoder catchup policy here . We encode one extra word for every even step. Gu et al. [2017]: ▶:CW=2; ▼:CW=5; ■:CW=8.

Train \ Test	k=1	k=3	k=5	k=7	k=9	k=∞
k'=1	*34.1*	33.3	31.8	31.2	30.0	15.4
k'=3	**34.7**	36.7	*37.1*	36.7	36.7	18.3
k'=5	30.7	36.7	37.8	38.4	*38.6*	22.4
k'=7	31.0	**37.0**	**39.4**	*40.0*	39.8	23.7
k'=9	26.4	35.6	39.1	**40.1**	***41.0***	28.6
k'=∞	21.8	30.2	36.0	38.9	39.9	***43.2***

Table 2.1: wait-k policy in training and test (4-ref BLEU, zh→en dev set). The bottom row is "test-time wait-k". Bold: best in a column; italic: best in a row.

[Vaswani et al., 2017].

2.1.3.2 Quality and Latency of Wait-k Model

Tab. 2.1 shows the results of a model trained with wait-k' but decoded with wait-k (where ∞ means full-sentence). Our wait-k is the diagonal, and the last row is the "test-time wait-k" decoding. Also, the best results of wait-k decoding is often from a model trained with a slightly larger k'.

Figs. 2.3–2.6 plot translation quality (in BLEU) against latency (in AP and CW) for full-sentence baselines, our wait-k, test-time wait-k (using full-sentence models), and our reimplementation of Gu et al. [2017][2] on the same Transformer baseline. In all these figures, we observe that, as k increases, (a) wait-k improves in BLEU score and worsens in latency, and (b) the gap between test-time wait-k and wait-k shrinks. Eventually, both wait-k and test-time wait-k approaches the full-sentence baseline as $k \to \infty$. These results are consistent with our intuitions.

[2]However, it is worth noting that, despite our best efforts, we have failed to reproduce their work on their original RNN, regardless of using their code or our own implementation. That being said, our successful implementation of their work on Transformer is also a notable contribution of this work.

	k=3	k=5	k=7	k=3	k=5	k=7
	zh→en			en→zh		
sent-level %	33	21	9	52	27	17
word-level %	2.5	1.5	0.6	5.8	3.4	1.4
accuracy	55.4	56.3	66.7	18.6	20.9	22.2
	de→en			en→de		
sent-level %	44	27	8	28	2	0
word-level %	4.5	1.5	0.6	1.4	0.1	0.0
accuracy	26.0	56.0	60.0	10.7	50.0	n/a

Table 2.2: Human evaluation for all four directions (100 examples each from dev sets). We report sentence- and word-level anticipation rates, and the word-level anticipation accuracy (among anticipated words).

We next compare our results with our reimplementation of Gu et al. [2017]'s two-staged full-sentence model + reinforcement learning on Transformer. We can see that while on BLEU-vs-AP plots, their models perform similarly to our test-time wait-k for de→en and zh→en, and slightly better than our test-time wait-k for en→zh, which is reasonable as both use a full-sentence model at the very core. However, on BLEU-vs-CW plots, their models have much worse CWs, which is also consistent with results in their paper (Gu, p.c.). This is because their R/W model prefers consecutive segments of READs and WRITEs (e.g., their model often produces R R R R R W W W W R R R W W W W R ...) while our wait-k translates concurrently with the input (the initial segment has length k, and all others have length 1, thus a much lower CW). We also found their training to be extremely brittle due to the use of RL whereas our work is very robust.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	doch	während	man	sich	im	kongre-	ss	nicht	auf	ein	vorgehen	einigen	kann	,	warten	mehrere	bs.	nicht	länger
	but	while	they	-self	in	congress		not	on	one	action	agree	can	,	wait	several	states	no	longer
k=3					but	,	while	congress	has	not	agreed	on	a	course	of	action	,	several	states	no	longer	wait

Figure 2.7: German-to-English example in the dev set with anticipation. The main verb in the embedded clause, “einigen” (agree), is correctly predicted 3 words ahead of time (with “sich” providing a strong hint), while the aux. verb “kann” (can) is predicted as “has”. The baseline translation is “but , while congressional action can not be agreed , several states are no longer waiting”. bs.: bunndesstaaten.

	1	2	3	4	5	6	7	8	9	10	11	12	
	tā	*hái*	*shuō*	*xiànzài*	*zhèngzài*	*wèi*	*zhè*	*yī*	*fǎngwèn*	*zuò*	*chū*	*ānpái*	
	他	还	说	现在	正在	为	这	一	访问	作	出	安排	
	he	also	said	now	(*prog.*)$^\diamond$	for	this	one	visit	make	out	arrangement	
k=1	he	also	said	that	he	is	now	making	preparations	for	this	visit	
k=3			he	also	said	that	he	is	making	preparations	for	this	visit
k=∞	he also said that arrangements are being made for this visit												

Figure 2.8: Chinese-to-English example in the dev set with anticipation. Both wait-1 and wait-3 policies yield perfect translations, with “making preparations” predicted well ahead of time. $^\diamond$: progressive aspect marker.

	1	2	3	4	5	6	7	8	9	10	11	
	jiāng	*zémín*	*duì*	*bùshí*	*zǒngtǒng*	*lái*	*huá*	*fǎngwèn*	*biǎoshì*	*rèliè*	*huānyíng*	
	江	泽民	对	布什	总统	来	华	访问	表示	热烈	欢迎	
	jiang	zeming	to	bush	president	come-to	china	visit	express	warm	welcome	
k=3			jiang	zemin	expressed	welcome	to	president	bush	’s	visit to china	
k=3†			jiang	zemin	meets	president	bush	in	china	’s	bid to visit china	

Figure 2.9: Chinese-to-English example from online news. Our wait-3 model correctly anticipates both “expressed" and “welcome” (though missing “warm”), and moves the PP (“to ... visit to china") to the very end which is fluent in the English word order. †: test-time wait-k produces nonsense translation.

	1	2	3	4	5	6	7	8	9	10	
	Měiguó	*dāngjú*	*duì*	*Shātè*	*jìzhě*	*shīzōng*	*yī*	*àn*	*gǎndào*	*dānyōu*	
(a)	美国	当局	对	沙特	记者	失踪	一	案	感到	担忧	
	US	authorities	to	Saudi	reporter	missing	a	case	feel	concern	
k=3				the	us	authorities	are	very	concerned	about	the saudi reporter ’s missing case
k=3†				the	us	authorities	have	dis-	appeared	from	saudi reporters
										bùmǎn	
(b)	美国	当局	对	沙特	记者	失踪	一	案	感到	不满	
k=3				the	us	authorities	are	very	concerned	about	the saudi reporter ’s missing case
k=5						the	us	authorities	have	expressed	**dissatisfaction** with the incident of saudi arabia ’s missing reporters

Figure 2.10: (a) Chinese-to-English example from more recent news, clearly outside of our data. Both the verb “*gǎndào*” (“feel”) and the predicative “*dānyōu*” (“concerned”) are correctly anticipated, probably hinted by “missing”. (b) If we change the latter to *bùmǎn* (“dissatisfied”), the wait-3 result remains the same (which is wrong) while wait-5 translates conservatively without anticipation. †: test-time wait-k produces nonsense translation.

Figure 2.11: English-to-Chinese example in the dev set with incorrect anticipation due to mandatory long-distance reorderings. The English sentence-final clause "since the founding of new china" is incorrectly predicted in Chinese as "近 几 年 来"("in recent years"). Test-time wait-3 produces translation in the English word order, which sounds odd in Chinese, and misses two other quantifiers ("in the medical and health system" and "nationwide"), though without prediction errors. The full-sentence translation, "据 了解，这 是 新 中国 成立 以来 ， 全国 医疗 卫生 系统 发生 的 最 大 的 一 起 火 灾 事故", is perfect.

2.1.3.3 Human Evaluation on Anticipation

Tab. 2.2 shows human evaluations on anticipation rates and accuracy on all four directions, using 100 examples in each language pair from the dev sets. As expected, we can see that, with increasing k, the anticipation rates decrease (at both sentence and word levels), and the anticipation accuracy improves. Moreover, the anticipation rates are very different among the four directions, with

$$\text{en}\rightarrow\text{zh} \;>\; \text{de}\rightarrow\text{en} \;>\; \text{zh}\rightarrow\text{en} \;>\; \text{en}\rightarrow\text{de}$$

Interestingly, this order is exactly the same with the order of the BLEU-score gaps between our wait-9 and full-sentence models:

$$\text{en}\rightarrow\text{zh: } 2.7 > \text{de}\rightarrow\text{en: } 1.1 > \text{zh}\rightarrow\text{en: } 1.6^{\dagger} > \text{en}\rightarrow\text{de: } 0.3$$

(†: difference in 4-ref BLEUs, which in our experience reduces by about half in 1-ref BLEUs). We argue that this order roughly characterizes the relative difficulty of *simultaneous* translation in these directions. In our data, we found en→zh to be particularly difficult due to the mandatory long-distance reorderings of English sentence-final temporal clauses (such as "in recent years") to much earlier positions in Chinese; see Fig. 2.11 for an example. It is also well-known that de→en is more challenging in simultaneous translation than en→de since SOV→SVO involves prediction of the verb, while SVO→SOV generally does not need prediction in our wait-k with a reasonable k, because V is often shorter than O. For example, human evaluation found only 1.3%, 0.1%, and 0% word anticipations in en→de for k=3, 5 and 7, and 4.5%, 1.5%, and 0.6% for de→en.

2.1.3.4 Examples and Discussion

We showcase some examples in de→en and zh→en from the dev sets and online news in Figs. 2.7 to 2.10. In all these examples except Fig. 2.10(b), our wait-k models can generally anticipate correctly, often producing translations as good as the full-sentence baseline. In Fig. 2.10(b), when we change the last word, the wait-3 translation remains unchanged (correct for (a) but wrong for (b)), but wait-5 is more conservative and produces the correct translation without anticipation.

Fig. 2.11 demonstrates a major limitation of our fixed wait-k policies, that is, sometimes it is just impossible to predict correctly and you have to wait for more source words. In this example, due to the required long-distance reordering between English

and Chinese (the sentence-final English clause has to be placed very early in Chinese), any wait-k model would not work, and a good policy should wait till the very end.

2.2 Learning Flexible Policy based on Pre-trained NMT

Simultaneous translation outputs target words while the source sentence is being received, and is widely useful in international conferences, negotiations and press releases. However, although there is significant progress in machine translation (MT) recently, simultaneous machine translation is still one of the most challenging tasks. This is because it is difficult to make good translation decisions to keep both good translation quality and small latency, especially for the syntactically divergent language pairs, such as German and English.

Researchers previously study simultaneous translation as a part of real-time speech recognition system [Yarmohammadi et al., 2013, Bangalore et al., 2012] Recent simultaneous translation research falls into two main categories: (1) learn a fixed-latency policy [Ma et al., 2019a, Dalvi et al., 2018] and (2) learn an adaptive policy with reinforcement learning (RL) method [Gu et al., 2017, Alinejad et al., 2018]. The fixed-latency policy usually starts by waiting for the first k source words and then outputs w translated words after receiving w source words until source sentence ends, when the remaining translation will be outputted altogether. It is easy to see that this kind of policy will inevitably need to guess the future content, which always can be incorrect. Therefore, an adaptive policy (see Table 2.3 as an example), which can decides on the fly whether to wait for another source word (READ) or to emit a target word (WRITE) using an

German	Ich	bin	mit	dem	Bus	nach	Ulm	gekommen	
Gloss	I	am	with	the	bus	to	Ulm	come	
Action	R W	R R	R	R	R W W W	R R	R		W W W W
Translation	I				took the bus				to come to Ulm

Table 2.3: An example for READ/WRITE action sequence. R represents READ and W represents WRITE.

MT model, is more desirable for simultaneous translation. However, previous work [Gu et al., 2017, Alinejad et al., 2018] depends on reinforcement learning (RL) method to learn such an adaptive policy, whose training process is very unstable and inefficient due to its exploration process. Furthermore, such a learned policy cannot control latency in applications, which reduces its applicability in many scenarios.

To combine the merits of both approaches and resolve their issues, in this work we propose a simple supervised learning framework to learn an adaptive policy, and show how to apply it with controllable latency. This framework is based on sequences of READ/WRITE actions for parallel sentence pairs, so we present a simple method to generate such an action sequence for each sentence pair with a pre-trained neural machine translation (NMT) model. Our experiments on German↔English dataset show that our method can lead to better policy model than previous methods without retraining the underlying NMT model. And our model can achieve better performance than the retrained wait-k models for small latency.

2.2.1 Supervised-Learning for Simultaneous Translation Policy

Given a sentence pair and an action sequence for this pair, we can apply supervised learning method to learn a parameterized policy for simultaneous translation. For the policy to be able to choose the correct action, its input should include information

from both source and target sides. Since we use Transformer [Vaswani et al., 2017] as our underlying NMT model in this work, the policy input o_i at step i consists of three components from this model:

- h_i^s: the last hidden state from encoder for the first source word at step i;
- h_i^t: the last hidden state from decoder for the first target word at step i;
- c_i: cross-attention scores at step i for the current input target word on all attention layers in decoder, averaged over all current source words.

That is $o_i = [h_i^s, h_i^t, c_i]$.

Let a_i the i-th action in the given action sequence $\mathbf{a}$. Then the decision of our policy on the i-th step depends on all previous inputs $\mathbf{o}_{\leq i}$ and all taken actions $\mathbf{a}_{<i}$. We want to maximize the probability of the next action a_i given those information:

$$\max p_\theta(a_i|\mathbf{o}_{\leq i}, \mathbf{a}_{<i})$$

where p_θ is the action distribution of our policy parameterized by θ.

2.2.2 Generating Action Sequences

In this section, we show how to generate action sequences for parallel text. Our simultaneous translation policy can take two actions: READ (input a new source word) and WRITE (output a new target word). A sequence of such actions for a sentence pair $(\mathbf{s}, \mathbf{t})$ defines one way to translate $\mathbf{s}$ into $\mathbf{t}$. Thus, such a sequence must have $|\mathbf{t}|$ number of WRITE actions. However, not every action sequence is good for simultaneous translation. For instance, a sequence without any READ action will not provide any source information,

while a sequence with all $|\mathbf{s}|$ number of READ actions before all WRITE actions usually has large latency. Thus the ideal sequences for simultaneous translation should have the following two properties:

- there is no anticipation during translation, i.e. when choosing WRITE action, there is enough source side information for the MT model to predict the correct target word;
- the latency is as small as possible, i.e. the WRITE action for each target word appears as early as possible.

Table 2.3 gives an example for such a sequence.

In the following, we present a simple method to generate such an action sequence for a sentence pair $(\mathbf{s}, \mathbf{t})$ using a pre-trained NMT model, assuming this model can make reasonable prediction given incomplete source sentence. Our method is based on this observation: *if the rank of the next ground-truth target word is high enough in the prediction of the model, then this implies that there is enough source-side information for the model to make right prediction.* Specifically, we sequentially input the source words to the pre-trained model, and use it to predict next target word. If the rank of the gold target word is high enough, we will append a WRITE action to the sequence and then try the next target word; otherwise, we append a READ action and input a new source word. Let r be a positive integer, M be a pre-trained NMT model, $\mathbf{s}_{\leq i}$ be the source sequence consisting of the first i words of $\mathbf{s}$, and $rank_M(t_j|\mathbf{s}_{\leq i})$ be the rank of the target word t_j in the prediction of model M given sequence $\mathbf{s}_{\leq i}$. Then the generating process can be summarized as Algorithm 1.

Algorithm 1 Generating Action Sequence

Input: sentence pair $(\mathbf{s}, \mathbf{t})$, integer r, model M
$id_s \leftarrow 1$, $id_t \leftarrow 1$
Seq $\leftarrow [R]$
while $id_t \leq |\mathbf{t}|$ **do**
  if $rank_M(t_{id_t}|\mathbf{s}_{\leq id_s}) \leq r$ or $id_s = |\mathbf{s}|$ **then**
    Seq $\leftarrow$ Seq $+[W]$
    $id_t \leftarrow id_t + 1$
  else
    Seq $\leftarrow$ Seq $+[R]$
    $id_s \leftarrow id_s + 1$
return Seq

Although we can generate action sequences balancing the two wanted properties with appropriate value of parameter r, the latency of generated action sequence may still be large due to the word order difference between the two sentences. To avoid this issue, we filter the generated sequences with the latency metric *Average Lagging* (AL) proposed by Ma et al. [2019a], which quantifies the latency in terms of the number of source words and avoids some flaws of other metrics like *Average Proportion* (AP) [Cho and Esipova, 2016] and *Consecutive Wait* (CW) [Gu et al., 2017]. After the filtering process, each action sequence has AL less than a fixed constant α.

2.2.3 Experiments

Dataset We conduct experiments on English↔German (EN↔DE) simultaneous translation. We use the parallel corpora from WMT 15 for training[3], newstest-2013 for validation and newstest-2015 for testing. All datasets are tokenized and segmented into

[3]http://www.statmt.org/wmt15/translation-task.html

sub-word units with byte-pair encoding (BPE) [Sennrich et al., 2016], and we only use the sentence pairs of lengths less than 50 (on both sides) for training.

Model Configuration We use Transformer-base [Vaswani et al., 2017] as our NMT model and our implementation is based on PyTorch-based OpenNMT.We add an `<eos>` token on the source side, which is not included in the original OpenNMT codebase. Our policy model consists of an RNN of 512 units, a fully-connected layer of dimension 64, and a softmax function to produce the action distribution. We use BLEU [Papineni et al., 2002] as the translation quality metric and "Averaged Lagging" (AL) [Ma et al., 2019a] as the latency metric.

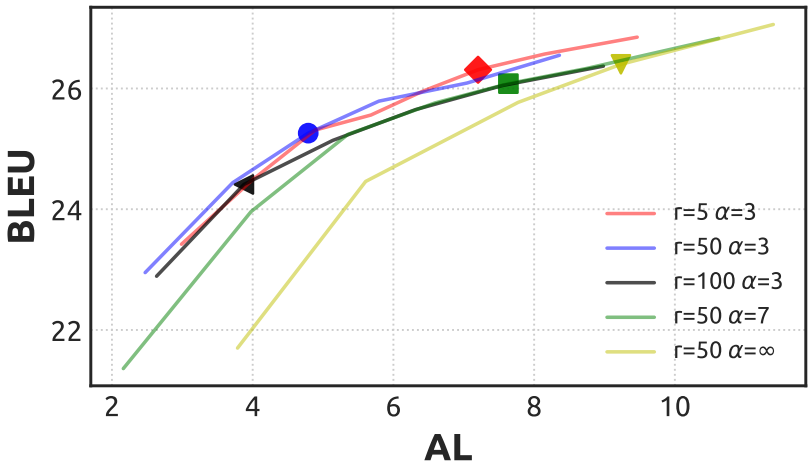
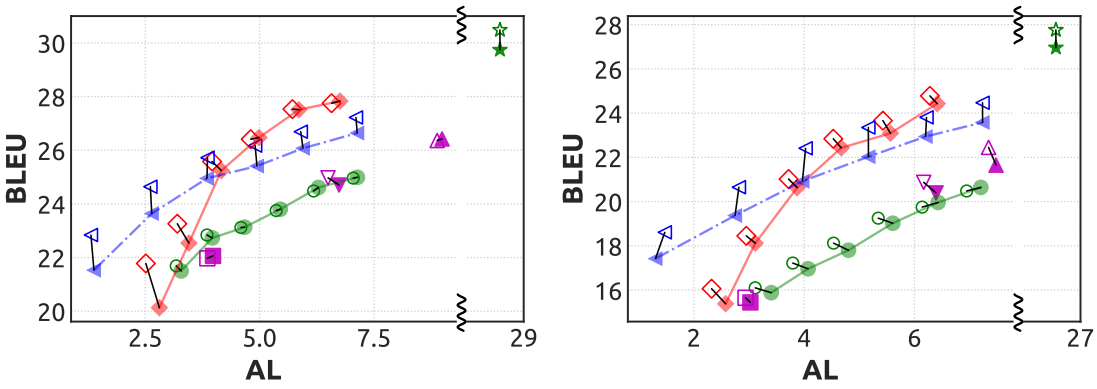


Figure 2.12: Translation quality against latency on DE→EN dev set. Lines are obtained with different probability thresholds ρ. Markers represent $\rho = 0.5$.

Effects of Generated Action Sequences We first analyze the effects of the two parameters in the generation process of action sequences: the rank r and the filtering



(a) DE→EN (b) EN→DE

Figure 2.13: Comparing performances of different methods on testing sets. The shown pairs are results of greedy decoding and beam search (beam-size = 5).

latency α. We fix $\alpha = 3$ and choose the rank $r \in \{5, 50, 100\}$; then we fix $r = 50$ and choose the latency $\alpha \in \{3, 7, \infty\}$ to generate action sequences on DE→EN direction, whose statistical information is provided in the supplemental material. Figure 2.12 shows the performances of resulting models with different probability thresholds ρ. We find that smaller α helps achieve better performance and our model is not very sensitive to different values of rank $r \in \{5, 50, 100\}$. Therefore, in the following experiments, we report results with $r = 50$ and $\alpha = 3$.

Performance Comparison We compare our method on EN↔DE directions with three different previous methods: RL method of [Gu et al., 2017], wait-k models and test-time wait-k methods of [Ma et al., 2019a]. For RL method[4], we use the same kind of input and architecture for policy model. Test-time wait-k means decoding with wait-k policy

[4]we find the reward function designed with CW metric works better than the AP-based reward function so we report models trained with CW-based reward function.

using the pre-trained NMT model. All methods share the same underlying pre-trained NMT model except for the wait-k method, which retrains the NMT model from scratch, but with the same architecture as the pre-trained model.

Figure 2.13 shows the performance comparison. In this figure, ● represents wait-k models for $k \in \{1, 2, 3, 4, 5, 6\}$, ● represents test-time wait-k for $k \in \{1, 2, 3, 4, 5, 6\}$, ◀ represents our SL model with threshold $\rho \in \{0.65, 0.6, 0.55, 0.5, 0.45, 0.4\}$, ★ represents full-sentence translation with pre-trained NMT model, ■ represents RL with CW = 2, ▼ represents RL with CW = 5, ▲ represents RL with CW = 8. Our models on both directions can achieve higher BLEU scores with the similar latency than the RL model and test-time wait-k method, implying that our method learns a better policy model than the other methods when the underlying NMT model is not retrained. Compared with wait-k models, our models with beam search achieve higher BLEU scores when latency AL is less than 4, which we think will be the most useful scenarios of simultaneous translation. When latency is larger, the gap between two methods is less than 1 BLEU point. Furthermore, this figure also shows that our model can achieve good performance on different latency conditions by controlling the threshold ρ, so we do not need to train multiple models for different latency requirements. We also provide a translation example in the supplemental material to compare different methods.

Learning Process Analysis We analyze two aspects of the learning processes of different methods: stability and training time. Figure 2.14 shows the learning curves of training process of RL method and our SL method, averaged over four different runs with different random seeds on DE→EN direction. The training process of our method is more stable

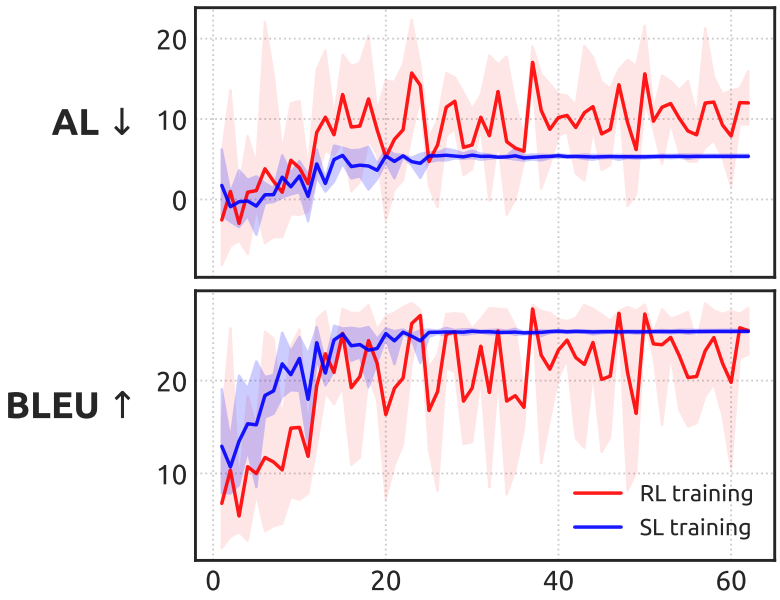


Figure 2.14: Learning curves averaged over four independent training runs. x-axis: training steps ($\times$ 50).

and converges faster than the RL method. Although there are some steps where the RL training process can achieve better BLEU scores than our SL method, the corresponding values of AL are usually very big, which are not appropriate for simultaneous translation. We present the training time of different methods[5] in Table 2.4. Our method only need about 12 hours to train the policy model with 1 GPU, while the wait-k method needs more than 600 hours with 8 GPUs to finish the training process, showing that our method is very efficient.

[5]Since RL training is unstable, we report the training times to obtain the models used in the performance comparisons.

DE→EN	pre-train: 64	SL: +12	wait-1	wait-3	wait-5
		RL: +15	938	966	945
EN→DE	pre-train: 68	SL: +11	wait-1	wait-3	wait-5
		RL: +15	665	665	630

Table 2.4: Training times (in hours) of different methods. Wait-k training uses 8 GPUs, while others use 1 GPU.

2.3 Learning Flexible Policy from Scratch

Simultaneous translation, which translates sentences before they are finished, is useful in many scenarios such as international conferences, summits, press releases, and negotiations. However, it is widely considered one of the most challenging tasks in NLP, and one of the holy grails of AI. A major challenge in simultaneous translation is the word order difference between source and target languages, e.g., from SOV languages (German, Japanese, etc.) to SVO languages (English, Chinese, etc.).

Recently, there have been two encouraging efforts in simultaneous translation with promising but limited success. Gu et al. [2017] propose a two-stage model that is also trained in two stages. The base model, responsible for producing target words, is a conventional full-sentence seq2seq model, and on top of that, the READ/WRITE (R/W) model decides, at every step, whether to wait for another source word (READ) or to emit a target word (WRITE) using the pretrained base model. This R/W model is trained by reinforcement learning without updating the base model. Ma et al. [2019a], on the other hand, propose a much simpler architecture, which only need one model and can be trained with end-to-end local training method. Their model can implicitly learn anticipation ability but it can only follow fixed-latency policy, which is not adaptive and flexible.

We aim to combine the merits of both efforts, that is, a flexible, adaptive, policy that

decides on the fly whether to wait or translate, as in Gu et al. [2017], but instead of using a two-stage framework with a full-sentence translation model ill-suited for simultaneous translation, we design a single model *trained* to perform simultaneous translation from scratch, as with Ma et al. [2019a]. The first key idea is to add a "delay" token (similar to the READ action in Gu et al. [2017]) to the target-side vocabulary, and if the model emits ..., it will read one source word. The second key idea is to train it using (restricted) imitation learning by designing a (restricted) dynamic oracle.

To obtain a flexible and adaptive policy, we need our model to be able to take both READ and WRITE actions. Conventional translation model already has the ability to write target words, so we introduce a "delay" token $\langle\varepsilon\rangle$ in target vocabulary to enable our model to apply the READ action. Given a prefix pair, the model will continue predicting target words until its next prediction is the delay token, and then it will read a new source word and continue prediction. Formally, for the target vocabulary V, we define an extended vocabulary

$$V_+ = V \cup \{\langle\varepsilon\rangle\} \tag{2.11}$$

Each word in this set can be defined as an action, which is applied with a transition function δ on a sequence pair $(\mathbf{s}, \mathbf{t})$ for a given source sequence $\boldsymbol{x}$ where $\mathbf{s} \preceq \boldsymbol{x}$. We assume $\langle\varepsilon\rangle$ cannot be applied with the sequence pair $(\mathbf{s}, \mathbf{t})$ if $\mathbf{s} = \boldsymbol{x}$, then we have the transition function δ as follows.

$$\delta(state, a) = \begin{cases} (\mathbf{s} \circ x_{|\mathbf{s}|+1}, \mathbf{t}) & \text{if } a = \langle\varepsilon\rangle \\ (\mathbf{s}, \mathbf{t} \circ a) & \text{otherwise} \end{cases}$$

Since the those tokens do not provide any semantic information, those predicted delay tokens may introduce some noise in the translation process. So we propose to remove delay token in the attention layers except for the current input one. However, this removal may reduce the explicit latency information which will affect the outputs of the model since the model cannot observe previous outputted delay tokens. Therefore, to provide this information explicitly, we embed the number of previous outputted delay tokens to a vector and add this to the sum of the word embedding and position embedding as the inputs of the model.

2.3.1 Training via Restricted Imitation Learning

We first introduce a restricted dynamic oracle based on our extended vocabulary for simultaneous translation. Then we show how to use this dynamic oracle to train a simultaneous translation model.

Restricted Dynamic Oracle Given a pair of full sequences $(\boldsymbol{x}, \boldsymbol{y})$ in data, the state of our restricted dynamic oracle will be a pair of prefixes $(\mathbf{s}, \mathbf{t})$ where $\mathbf{s} \preceq \boldsymbol{x}$, $\mathbf{t} \preceq \boldsymbol{y}$ and $(\mathbf{s}, \mathbf{t}) \neq (\boldsymbol{x}, \boldsymbol{y})$. The whole action set is V_+ defined in last section. The objective of our dynamic oracle is to obtain the full sequence pair $(\boldsymbol{x}, \boldsymbol{y})$ and maintain a reasonably low latency.

For a prefix pair $(\mathbf{s}, \mathbf{t})$, the difference of the lengths of the two prefixes can be used to measure the latency of translation. So we would like to bound this difference as a latency constraint. This idea can be illustrated in the prefix grid, where we can define

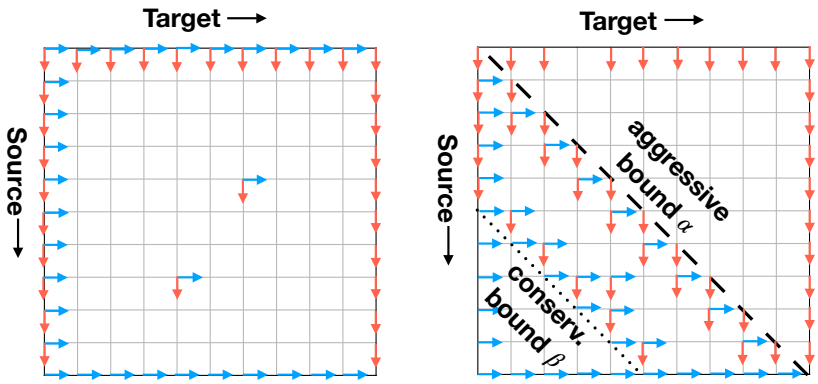


Figure 2.15: Illustration of our proposed dynamic oracle on a prefix grid. The right arrow represents extending next ground-truth target word, and the red downward arrow represents the delay token. The left figure shows a simple dynamic oracle without without delay constraint. The right figure shows the dynamic oracle with delay constraint.

a band region and always keep the states in this band during the translation process (see Figure 2.15). For simplicity, we first assume the two full sequences have the same lengths, i.e. $|\boldsymbol{x}| = |\boldsymbol{y}|$. Then we can bound the difference $d = |\mathbf{s}| - |\mathbf{t}|$ by two constants: $\alpha < d < \beta$. The conservative bound (β) guarantees relatively small difference and then low latency; while the aggressive bound (α) guarantees the target sequence cannot predict too many target words before seeing enough source words. Formally, this dynamic oracle is defined as follows.

$$\pi^{\star}_{\boldsymbol{x},\boldsymbol{y}}((\mathbf{s},\mathbf{t})) = \begin{cases} \{\langle\varepsilon\rangle\} & \text{if } \mathbf{s} \neq \boldsymbol{x} \text{ and } |\mathbf{s}| - |\mathbf{t}| \leq \alpha \\ \{y_{|\mathbf{t}|+1}\} & \text{if } \mathbf{t} \neq \boldsymbol{y} \text{ and } |\mathbf{s}| - |\mathbf{t}| \geq \beta \\ \{\langle\varepsilon\rangle, y_{|\mathbf{t}|+1}\} & \text{otherwise} \end{cases}$$

By this definition, we know that this oracle can always find an action sequence to obtain $(\boldsymbol{x}, \boldsymbol{y})$. When the state does not satisfy the latency constraint, then this dynamic

oracle will provide only one action applying which will improve the length difference. Note that this dynamic oracle is restricted in the sense that it is only defined on the prefix pair instead of any sequence pair. And since we only want to obtain the exact sequence from data, this oracle can only choose the next ground-truth target word other than $\langle\varepsilon\rangle$.

In many cases, the assumption that $|\boldsymbol{x}| = |\boldsymbol{y}|$ does not hold. To overcome this issue, we can utilize the length ratio $\gamma = |\boldsymbol{y}|/|\boldsymbol{x}|$ to modify the length difference: $d' = |\mathbf{s}| - \gamma|\mathbf{t}|$, and use this new difference d' in our dynamic oracle. Although we cannot obtain this ratio during testing time, we may use averaged length ratio obtained from training data (Huang et al. [2017]).

2.3.2 Training with Restricted Dynamic Oracle

To learn the proposed dynamic oracle, we apply imitation learning to train our model. Recall that the prediction of our model depends on the whole generated prefix including $\langle\varepsilon\rangle$ (as the input contains the embedding of the number of $\langle\varepsilon\rangle$), which is also an action sequence. Following such an action sequence will result in a prefix pair of $\boldsymbol{x}$ and $\boldsymbol{y}$ if the action sequence is obtained from our oracle. Let s be such a resulting prefix pair. The probability of choosing the oracle actions conditioned on the prefix pair s obtained from an action sequence will be

$$p(\pi^{\star}_{\boldsymbol{x},\boldsymbol{y},\alpha,\beta}(s)|s) = \frac{\sum_{a \in \pi^{\star}_{\boldsymbol{x},\boldsymbol{y},\alpha,\beta}(s)} p(a \mid s)}{|\pi^{\star}_{\boldsymbol{x},\boldsymbol{y},\alpha,\beta}(s)|} \tag{2.12}$$

To train our model to learn from the dynamic oracle, we can sample from our oracle

a set $S(\boldsymbol{x}, \boldsymbol{y})$ of action sequences for a translation pair $(\boldsymbol{x}, \boldsymbol{y})$. The loss for each sampled sequence $\mathbf{a} \in S(\boldsymbol{x}, \boldsymbol{y})$ will be

$$\ell(\mathbf{a}|\boldsymbol{x}) = -\sum_{i=1}^{|\mathbf{a}|} \log p(\pi^{\star}_{\boldsymbol{x},\boldsymbol{y},\alpha,\beta}(s_i)|s_i)) \tag{2.13}$$

where s_i is the prefix pair obtained from $\mathbf{a}_{<i}$. For a parallel text D, the training loss is

$$\ell(D) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in D} \sum_{\mathbf{a}\in S(\boldsymbol{x},\boldsymbol{y})} \tfrac{1}{|S(\boldsymbol{x},\boldsymbol{y})|}\ell(\mathbf{a}|\boldsymbol{x}) \tag{2.14}$$

Directly optimizing the above loss may require too much computation resource since for each pair of $(\boldsymbol{x}, \boldsymbol{y})$, the number of different action sequences is exponentially large. To reduce the computation cost, we propose to use these two special action sequences as our sample set. Recall that our dynamic oracle $\pi^{\star}_{\boldsymbol{x},\boldsymbol{y},\alpha,\beta}$ is sensitive to the latency constraint, which is defined by two bounds: α and β. For each bound, there is a unique action sequence, which corresponds to a path in the prefix grid, such that it has the most number of prefix pairs that make this bound tight. Let $\mathbf{a}_{\alpha}$ ($\mathbf{a}_{\beta}$) be such an action sequence for α (β). Then the above loss for dataset D becomes

$$\ell(D) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in D} \tfrac{1}{2}(\ell(\mathbf{a}_{\alpha}|\boldsymbol{x}) + \ell(\mathbf{a}_{\beta}|\boldsymbol{x})) \tag{2.15}$$

2.3.3 Experiments

To investigate the empirical performances of our proposed methods, we conduct experiments on NIST Chinese-to-English 1M translation dataset using Transformer [Vaswani

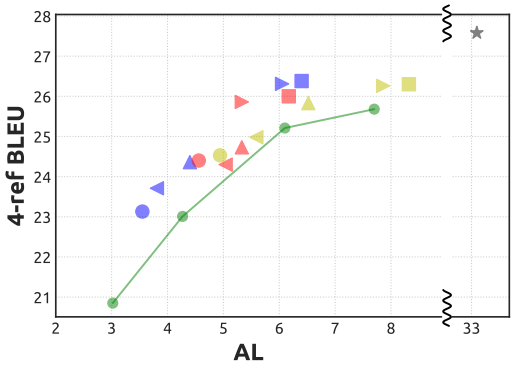


Figure 2.16: BLEU and latency on test-set. ● (forced wait-1 policy), ◀ ($t = -2$), ▲ ($t = -0.5$), ▶ (free decoding), ■ (forced wait-5 policy) are trained using model $\alpha = 1, \beta = 5$. ● (forced wait-3 policy), ◀ ($t = -2$), ▲ ($t = -0.5$), ▶ (free decoding), ■ (forced wait-5 policy) are trained using model $\alpha = 3, \beta = 5$. ● (forced wait-3 policy), ◀ ($t = -2$), ▲ ($t = -0.5$), ▶ (free decoding), ■ (forced wait-7 policy) are trained using model $\alpha = 3, \beta = 7$. ● represents fixed wait-k baseline by our own implementation ($k = 1, 3, 5, 7$), ★ strands for full-sentence translation baseline.

et al., 2017].

We first apply BPE [Sennrich et al., 2015] on both sides in order to reduce the vocabulary for both source and target sides. We then exclude the sentences pairs whose length are longer than 50 and 256 words. We use NIST 06 (616 sentence pairs) and NIST 08 (691 sentence pairs) as our development and testing set. Our implementation is adapted from PyTorch-based OpenNMT [Klein et al., 2017]. Our Transformer's parameters are as the same as the base model's parameter settings in the original paper [Vaswani et al., 2017].

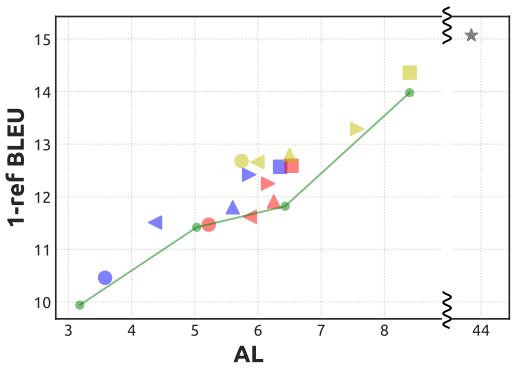


Figure 2.17: BLEU and latency on three models. ● (forced wait-1 policy), ◀ (free decoding), ▲ ($T = 4.5$), ▶ ($T = 9$), ■ (forced wait-5 policy) are trained using model $\alpha = 1, \beta = 5$. ● (forced wait-3 policy), ◀ ($T = 4.5$), ▲ ($T = 9$), ▶ (free decoding), ■ (forced wait-5 policy) are trained using model $\alpha = 3, \beta = 5$. ● (forced wait-3 policy), ◀ ($T = 4.5$), ▲ ($T = 9$), ▶ (free decoding), ■ (forced wait-7 policy) are trained using model $\alpha = 3, \beta = 7$. ● represents fixed wait-k baseline ($k = 1, 3, 5, 7$), ★ strands for full-sentence translation baseline.

2.3.3.1 Results

We use Average Lagging (AL) defined in [Ma et al., 2019a] as our latency metrics, which measures the average delayed words. It avoids some limitations from other metrics, for example, insensitivity to actual lagging like Consecutive Wait [Gu et al., 2017], sensitivity to input length like Average Proportion [Cho and Esipova, 2016] .

Fig. 2.16 shows our test results on Chinese-to-English translation. Compare with fixed policy wait-k models from [Ma et al., 2019a], most of the ours results achieves higher BLEU score or lower latency. Since the latency of free decoding are always close to it's corresponding conservative bound, We apply a discount constant e^t to the probability of the delay token in inference time ($t = -0.5, -2$) to further improve its flexibility.

Fig. 2.16 shows our test results on English-to-Chinese translation. Since the English

Chinese	一	名	不		愿		具名		的		欧盟		官员		指出		，		每个		人		都		紧张		兮兮...	
pinyin	yì	míng	bú		yùan		jùmíng		de		Ōuméng		gūanyúan		zhǐchū		,		měigè		rén		dōu		jǐngzhāng		xīxī ...	
gloss	one	-	not		willing		named		-		EU		official		point out		,		every		people		are		nervous		- ...	
wait-3				a		us		official		who		declined		to		be		named		said		that		each		and		every one was caught in tensions ...
ours				a								eu		official		,		who		declined to		be		named		,		pointed out that everyone was nervous ...

Table 2.5: Chinese-to-English example in the dev set using model trained with $\alpha = 3, \beta = 7$,

source sentences are always much longer than Chinese, we use catchup [Ma et al., 2019a] in bounding policies during training with catchup frequency of $c = 0.25$, which is derived from the dev set tgt/src length ratio of 1.25. Different from the previous zh-en experiments, the latency of free decoding are always close to it's corresponding aggressive bound. Thus, we apply the discount constant temperature $t = 4.5, 9$. These two experiment results show higher translation quality, lower latency and more flexible in decoding policy.

2.3.3.2 Example

Table 2.5 shows an example from dev set using model trained with $\alpha = 3, \beta = 7$. Our proposed model translates "eu" after "Ōuméng", while the fixed wait-3 policy forced to anticipate the wrong word "us" before seeing "Ōuméng".

Chapter 3: Beam Search for Simultaneous Translation

Beam search has been widely used in neural text generation such as machine translation [Sutskever et al., 2014, Bahdanau et al., 2014b], summarization [Rush et al., 2015, Ranzato et al., 2016], and image captioning [Vinyals et al., 2015, Xu et al., 2015]. It often leads to substantial improvement over greedy search and becomes an essential component in almost all text generation systems.

However, beam search is easy for the above tasks because they are all *full-sequence* problems, where the whole input sequence is available at the beginning and the output sequence only needs to be revealed in full at the end. By contrast, in language and speech processing, there are many *incremental processing* tasks with *simultaneity requirements*, where the output needs to be revealed to the user incrementally without revision (word by word, or in chunks) and the input is also being received incrementally. Two most salient examples are streaming speech recognition [Chiu et al., 2018], widely used in speech input and dialog systems (such as Siri), and simultaneous translation [Bangalore et al., 2012, Oda et al., 2015, Grissom II et al., 2014, Jaitly et al., 2016], widely used in international conferences and negotiations. In these tasks, the use of full-sentence beam search becomes seemingly impossible as output words need to be committed on the fly.

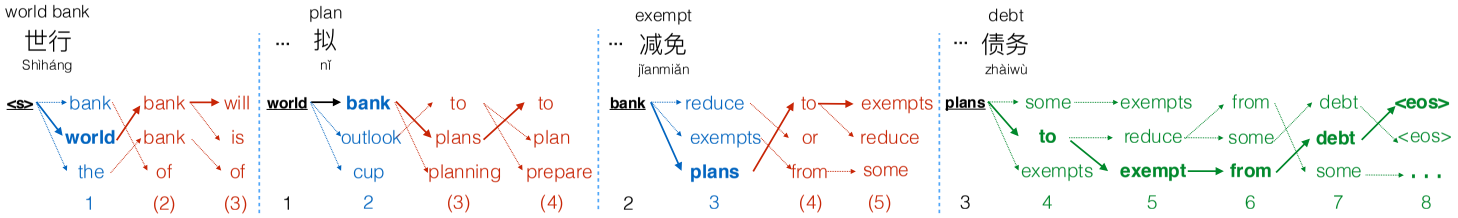


Figure 3.1: Wait-1 policy example to illustrate the procedure of SBS. The top Chinese words are the source side inputs which are incrementally revealed to the encoder. Gloss is annotated above Chinese word and Pinyin is underneath. There are two extra steps (speculative window) are taken (red part) beyond greedy. When source reaches the last word "债务" (debt), the decoder gets into tail and performs conventional beam search (in green).

3.1 Full Sentence NMT and Beam Search

How to adapt beam search for such incremental tasks in order to improve their generation quality? We propose a general technique of *speculative beam search* (SBS), and apply it to simultaneous translation. At a very high level, to generate a single word, instead of simply choosing the highest-scoring one (as in greedy search), we further speculate w steps into the future, and use the ranking at step $w+1$ to reach a more informed decision for step 1 (the current step); this method implicitly benefits from a target language model and alleviates the label bias problem well-known in neural generation [Murray and Chiang, 2018].

We apply this algorithm to two representative approaches to simultaneous translation: the fixed policy method [Ma et al., 2019a] and the adaptive policy method [Gu et al., 2017]. In both cases, we show that SBS improves translation quality while maintaining latency (i.e., simultaneity).

3.2 Speculative Beam Search for Simultaneous Translation

We first present our speculative beam search on the fixed-latency wait-k policy (generating a single word per step), and then adapt it to the adaptive policies (generating multiple words per step).

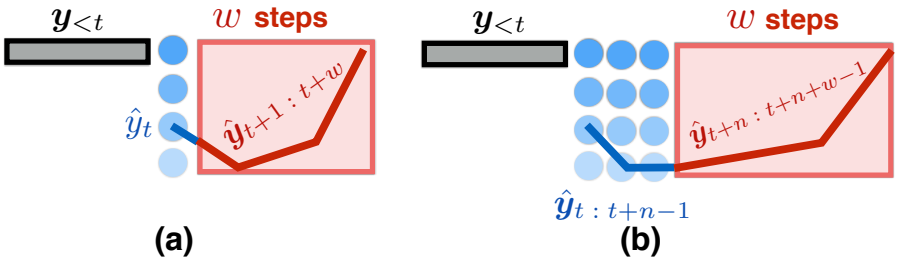


Figure 3.2: Illustration of SBS: (a) wait-k policy (Eqs. 3.1–3.2); (b) adaptive policy (Eqs. 3.3–3.4). Speculations in red.

3.2.1 Single-Step SBS

The wait-k policy conducts translation concurrently with the source input, committing output words one by one while the source sentence is still growing. In this case, conventional beam search is clearly inapplicable.

We propose to perform speculative beam search at each step by hallucinating w more steps into the future, and use the ranking after these $w+1$ steps to form a more informed decision for the current step. More formally, at step t, we generate y_t based on already committed prefix $\boldsymbol{y}_{<t}$:

$$\langle \hat{\boldsymbol{y}}, s_t \rangle = \mathbf{top}^1\big(\mathrm{next}^b_{1+w}([\langle \boldsymbol{y}_{<t}, 1 \rangle])\big) \tag{3.1}$$

$$\boldsymbol{y}_{\leq t} = \boldsymbol{y}_{<t} \circ \hat{y}_t \tag{3.2}$$

where $\hat{\boldsymbol{y}} = \boldsymbol{y}_{<t} \circ \hat{y}_t \circ \hat{\boldsymbol{y}}_{t+1:t+w}$ has three parts, with the last one being a speculation of w steps (see Fig. 3.2). We use $\mathrm{next}^b_{1+w}(\cdot)$ to speculate w steps. The candidate $\hat{y}_t$ is selected based on the accumulative model score w steps later. Then we commit $\hat{y}_t$ and move on to step $t+1$.

In the running example in Fig. 3.1, we have $w = 2$ and $b = 3$. In the greedy mode, after the wait-1 policy receives the first source word, "世行" (world bank), the basic wait-1 model commits "bank" which has the highest score. In SBS, we perform a beam search for $1 + w = 3$ steps with the two speculative steps marked in red. After 3 steps, the path "world bank will" becomes the top candidate, thus we choose to commit "world" instead of "bank" and restart a new speculative beam search with "world" when we receive a new source word, "拟"(plan to); the speculative part from the previous step (in red) is removed.

3.2.2 Chunk-based SBS

The RL-based adaptive policy system [Gu et al., 2017] can commit a chunk of multiple words whenever there is a series of consecutive WRITEs, and conventional beam search can be applied on each chunk to improve the search quality within that chunk, which is already used in that work.

However, on top of the obvious per-chunk beam search, we can still apply SBS to further speculate w steps after the chunk. For a chunk of length n starting at position t, we adapt SBS as:

$$\langle \hat{\boldsymbol{y}}, s_t \rangle = \mathbf{top}^1\left(\mathrm{next}^b_{n+w}([\langle \boldsymbol{y}_{<t}, 1 \rangle])\right) \tag{3.3}$$

$$\boldsymbol{y}_{\leq t+n-1} = \boldsymbol{y}_{<t} \circ \hat{\boldsymbol{y}}_{t:t+n-1} \tag{3.4}$$

policy	static NMT	retrained NMT
fixed	test-time wait-k [Dalvi et al., 2018]	wait-k [Ma et al., 2019a]
adaptive	RL [Gu et al., 2017]	N/A

Table 3.1: Three approaches to simultaneous translation.

Here $\text{next}^b_{n+w}(\cdot)$ does a beam search of $n+w$ steps, with the last w speculated. Similarly,

$$\hat{\boldsymbol{y}} = \boldsymbol{y}_{<t} \circ \hat{\boldsymbol{y}}_{t:t+n-1} \circ \hat{\boldsymbol{y}}_{t+n:t+n+w-1}$$

has three parts, with the last being a speculation of w steps, and the middle one being the chunk of n steps returned and committed (see Fig. 3.2).

3.2.3 Experiments

3.2.3.1 Datasets and Latency Metrics

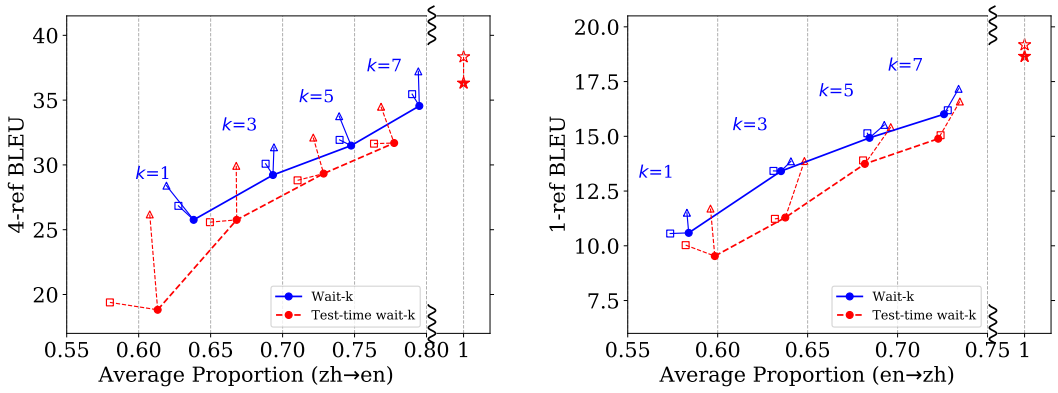


Figure 3.3: BLEU against AP using Wait-k model. □ □: conventional beam search only in target tail (when source finishes). △ △: speculative beam search. ★☆:full-sentence (greedy and beam-search).

We evaluate our work on Chinese↔English simultaneous translation tasks. For the training data, we use the NIST corpus for Chinese↔English (2M sentence pairs). We first apply BPE [Sennrich et al., 2015] on all texts in order to reduce the vocabulary sizes. For Chinese↔English evaluation, we use NIST 2006 and NIST 2008 as our dev and test sets with 4 English references. For English→Chinese, we use the second among the four English references as the source text.

We re-implement all models in Table 3.1: wait-k model [Ma et al., 2019a], test-time wait-k model [Dalvi et al., 2018] and adaptive policy model [Gu et al., 2017] based on PyTorch-based OpenNMT [Klein et al., 2017]. To reach state-of-art performance, we use Transformer based wait-k model and also use Transformer based pre-trained full sentence model for learning adaptive policy. The parameters of Transformer are the same as the base model from the original paper [Vaswani et al., 2017]. We use Average Proportion (AP) [Cho and Esipova, 2016] as the latency metrics. AP measures the normalized amount of word delay for translating a given source sentence.

b \ w	0	1	2	3	4	5
1	34.28	-	-	-	-	-
3		+1.7	+0.9	+1.4	+1.4	+1.4
5		+1.9	+1.5	+2.0	+1.3	+1.5
7		+2.1	+1.5	+2.0	+1.3	+1.6
10		+2.2	+1.5	+1.7	+1.4	+1.6

Table 3.2: Zh→En wait-1 model BLEU improvement of SBS against greedy result ($b = 1$, $w = 0$) on dev-set. When $b \geq 5$ the performance of SBS becomes stable.

		shìháng	*nǐ*	*jǐanmǐan*	*zùi*	*qióng*	*gúojiā*	*zhàiwù*	
		世行	拟	减免	最	穷	国家	债务	
Gloss		world bank	plan to	remit & reduce	most	poor	country	debt	
k=1†	Greedy		world	bank	to	reduce	poverty	-	stricken countries
	SBS		world	bank	to	exemp-	t	po-	or- est countries from debt
k=1‡	Greedy		world	bank	to	reduce	or	exemp-	t debt of poorest countries
	SBS		the	world	bank	inten-	ds	to	reduce or exemp- t the debt of the po- or- est countries
k=∞*									world bank plans to remit and reduce debts of po- or- est countries

Table 3.3: Chinese-to-English example on dev set. †: test-time wait-k; ‡: wait-k. *: full-sentence beam search.

3.2.3.2 Performance on Wait-k Policy

We perform experiments on validation set using speculative beam search (SBS) with beam sizes $b \in \{3, 5, 7, 10, 15\}$ and speculative window sizes $w \in \{1, 2, 3, 4, 5\}$. Table 3.2 shows the BLEU score of b and w over wait-1 model. Compared with greedy decoding, SBS improves at least 0.9 BLEU score in all cases and achieves best performance by $b = 10, w = 1$. (We recommend $b = 5, w = 3$ which can achieve substantial improvements in most cases.) We search the best b and w for each model on dev-set and apply them on test-set in the following experiments.

Fig. 3.3 shows the performance of conventional greedy decoding, trivial tail beam search (only after source sentence is finished) and SBS on test set on Chinese↔English tasks. SBS largely boost test-time wait-k models with slightly worse latency (especially in English→Chinese because they tend to generate longer sentences). Wait-k models also benefit from speculation (especially in Chinese→English).

Fig. 3.3 shows an running example of greedy and SBS output of both wait-k and test-time wait-k models. SBS on test-time wait-k generate much better outputs compared with the greedy outputs which misses several essential information. Wait-k with speculation correctly translates "拟" into "intends to" instead of "to" in greedy output.

We also evaluate our work using Consecutive Wait (CW) as latency metric, which measures the average lengths of consecutive wait segments, and perform experiments on German↔English corpora available from WMT15[1]. We use newstest-2013 as dev-set and newstest-2015 as test-set.[2]

[1] http://www.statmt.org/wmt15/translation-task.html

[2] The German↔English results are slightly different from those in Ma et al. [2019a] because of different decoding settings. We do not allow that the decoder stops earlier than the finish of source sentence while it

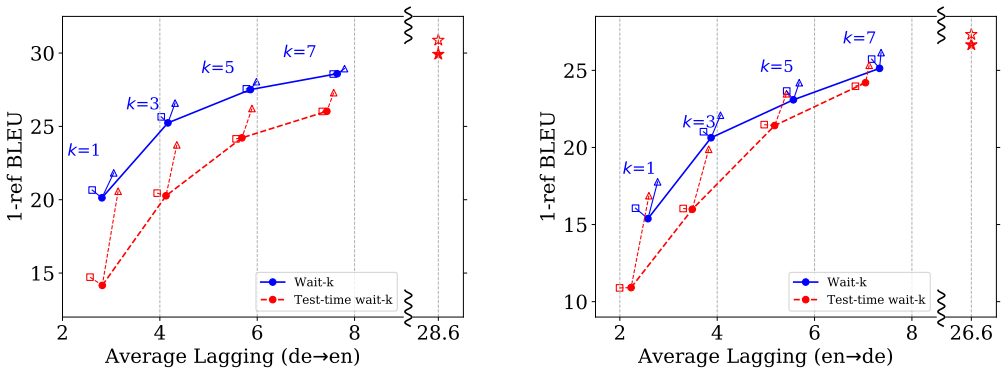


Figure 3.4: Translation quality against AL on English↔German simultaneous translation using wait-k model. □ □: conventional beam search only on target tail. △ △: speculative beam search. ★☆:full-sentence (greedy and beam-search).

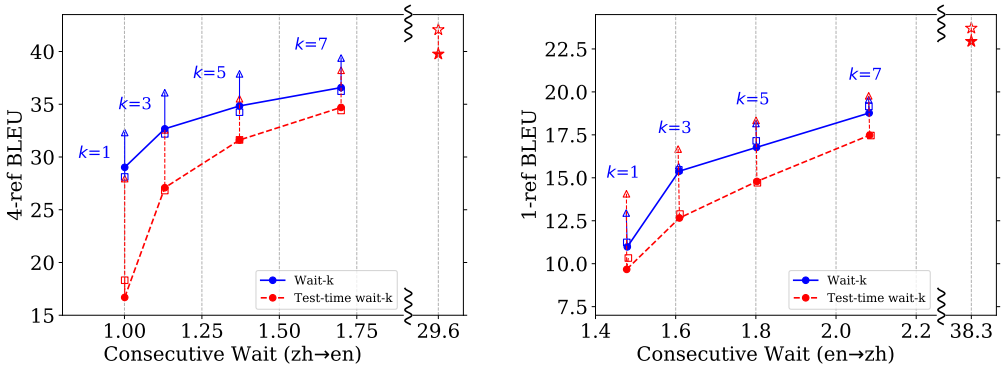


Figure 3.5: Translation quality against CW on Chinese↔English simultaneous translation using wait-k model.

Fig. 3.4 show the translation quality on German↔English against AL of different decoding methods. Consistent to the results of Chinese↔English, our proposed speculative beam search gain large performance boost especially on test-time wait-k. Fig. 3.5 and Fig. 3.6 use CW as latency metrics. Since both the wait-k and test-time wait-k models

is allowed in German↔English experiments of Ma et al. [2019a]. This makes our generated sentences longer and further results in worse AL compared with the results in Ma et al. [2019a].

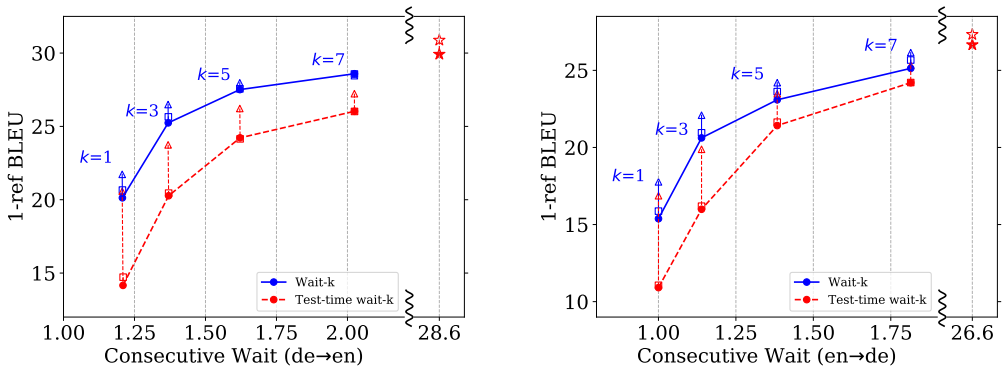


Figure 3.6: Translation quality against CW on English↔German simultaneous translation using wait-k model.

use the same fixed policy, the CW latencies of the same k are identical.

3.2.3.3 Performance on Adaptive Policy Model

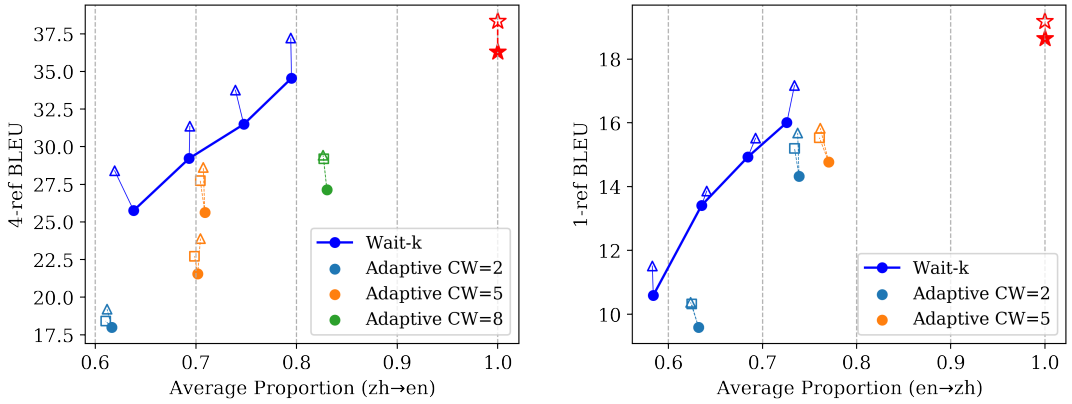


Figure 3.7: BLEU against AP using adaptive policy model (compared with wait-k models) with different beam search methods. □ □ □: conventional beam search in chunk of consecutive write [Gu et al., 2017]. △ △ △: speculative beam search. ★☆:full-sentence baseline (greedy and beam-search).

Fig. 3.7 shows the performance of proposed SBS on adaptive policy. We train adaptive

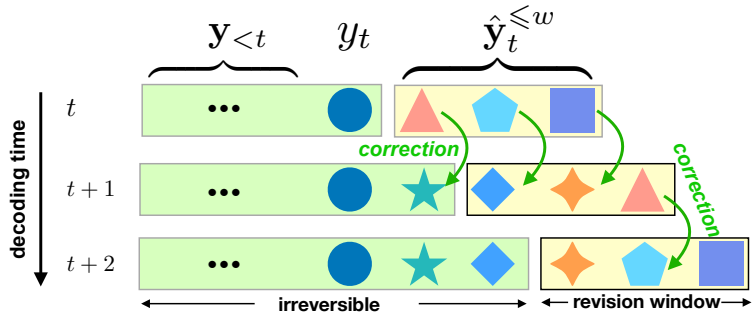


Figure 3.8: Besides y_t, opportunistic decoding continues to generate additional w words which are represented as $\hat{\boldsymbol{y}}_t^{\leqslant w}$. The timely correction only revises this part in future steps. Different shapes denote different words. In this example, from step t to $t+1$, all previously opportunistically decoded words are revised, and an extra triangle word is generated in opportunistic window. From step $t+1$ to $t+2$, two words from previous opportunistic window are kept and only the triangle word is revised.

policies using the combination of Consecutive Wait (CW $\in \{2, 5, 8\}$ [Gu et al., 2017]) and partial-BLEU as reward in reinforcement learning. We vary beam size $b \in \{5, 10\}$ in both conventional consecutive write beam search [Gu et al., 2017] and proposed SBS whose speculative window size $w \in \{2, 4\}$. Our proposed beam search achieves better results in most cases.

3.3 Opportunistic Decoding and Timely Correction with Beam Search

Though the existing efforts improve the performance in both translation latency and quality with more powerful frameworks, it is still difficult to choose an appropriate policy to explore the optimal balance between latency and quality in practice, especially when the policy is trained and applied in different domains. Furthermore, all existing approaches are incapable of correcting the mistakes from previous steps. When the

former steps commit errors, they will be propagated to the later steps, inducing more mistakes to the future.

We propose an opportunistic decoding technique with timely correction mechanism to address the above problems. As shown in Fig. 3.8, our proposed method always decodes more words than the original policy at each step to catch up with the speaker and reduce the latency. At the same time, it also employs a timely correction mechanism to review the extra outputs from previous steps with more source context, and revises these outputs with current preference when there is a disagreement. From the user experience point of view, at each time step, since we only review and modify the last few output words with a relatively low revision rate, in the speech-to-text scenario [Oda et al., 2014, Bangalore et al., 2012, Yarmohammadi et al., 2013] where the outputs are posted on the screen, the audience will not be overwhelmed by the modifications. This is a much better user experience compared with the "always re-translate" strategy.

We also define, for the first time, two metrics for revision-enabled simultaneous translation: a more general latency metric *Revision-aware AL* (RAL) as well as the *revision rate*. We demonstrate the effectiveness of our proposed technique using fixed Ma et al. [2019a] and adaptive [Zheng et al., 2019d] policies in both Chinese-to-English and English-to-Chinese (in appendix) translation.

3.3.1 Opportunistic Decoding

For simplicity, we first apply this method to fixed policies. We define the original decoded word sequence at time step t with y_t, which represents the word that is decoded in time

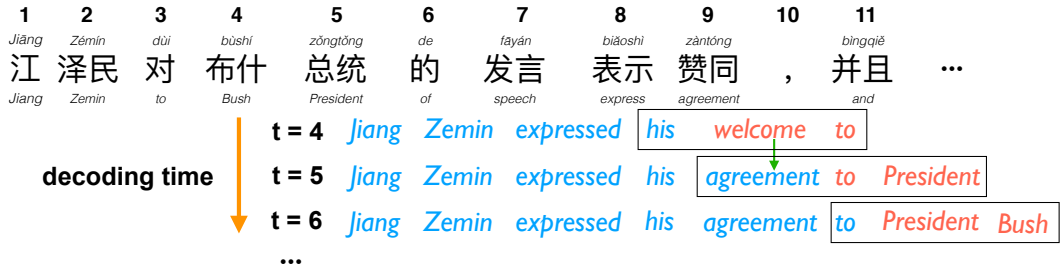


Figure 3.9: Decoder generates the 4^{th} word "his" and two extra words "welcome to" at the 4^{th} decoding step and the 9^{th} source word "zàntóng" (agreement) on source side is not available to the model yet. When the model receives the 9^{th} source word at 5^{th} step, the decoder immediately corrects the previous mistake with "agreement" and makes two more additional decoding. The decoder not only is capable to fix the previous mistake, but also has the enough information to perform more correct generations. Our framework benefits from opportunistic decoding with reduced latency here. Note though the word "to" is generated in step 4, it only becomes irreversible at step 6.

step t with original model. We denote the additional decoded words at time step t as $\hat{\boldsymbol{y}}_t^{\leqslant w} = (y_t^1, ..., y_t^w)$, where w denote the number of extra decoded words. In our setting, the decoding process is as follows:

$$
\begin{aligned}
&p_g(y_t \circ \hat{\boldsymbol{y}}_t^{\leqslant w} \mid \boldsymbol{x}_{\leqslant g(t)}) = \\
&p_g(y_t \mid \boldsymbol{x}_{\leqslant g(t)}) \prod_{i=1}^{w} p_g(\hat{y}_t^i \mid \boldsymbol{x}_{\leqslant g(t)},\ y_t \circ \hat{\boldsymbol{y}}_t^{<i})
\end{aligned}
\tag{3.5}
$$

where $\circ$ is the string concatenation operator.

We treat the procedure for generating the extra decoded sequence as opportunistic decoding, which prefers to generate more tokens basic on current context. When we have enough information, this opportunistic decoding eliminates unnecessary latency and keep the audience on track. With a certain chance, when the opportunistic decoding tends to aggressive and generates inappropriate tokens, we need to fix the inaccurate token immediately.

3.3.2 Timely Correction

In order to deliver the correct information to the audience promptly and fix previous mistakes as soon as possible, we also need to review and modify the previous outputs.

At $t+1$ step, when encoder obtains more information from $\boldsymbol{x}_{\leqslant g(t)}$ to $\boldsymbol{x}_{\leqslant g(t+1)}$, the decoder is capable to generate more appropriate candidates and may revise and replace the previous outputs from opportunistic decoding. More precisely, $\hat{\boldsymbol{y}}_t^{\leqslant w}$ and $y_{t+1} \circ \hat{\boldsymbol{y}}_{t+1}^{\leqslant w-1}$ are two different hypothesis over the same time chunk. When there is a disagreement, our model always uses the hypothesis from later step to replace the previous commits. Note our model does not change any word in y_t from previous step and it only revise the words in $\hat{\boldsymbol{y}}_t^{\leqslant w}$.

Modification for Adaptive Policy For adaptive policies, the only difference is, instead of committing a single word, the model is capable of generating multiple irreversible words. Thus our proposed methods can be easily applied to adaptive policies.

Correction with Beam Search When the model is committing more than one word at a time, we can use beam search to further improve the translation quality and reduce revision rate [Murray and Chiang, 2018, Yang et al., 2018].

The decoder maintains a beam B_t^k of size b at step t, which is ordered list of pairs $\langle$hypothesis, probability$\rangle$, where k denotes the k^{th} step in beam search. At each step, there is an initial beam $B_t^0 = [\langle \boldsymbol{y}_{t-1}, 1 \rangle]$. We denote one-step transition from the previous beam to the next as

$$
\begin{aligned}
B_t^{k+1} &= \mathrm{next}_1^b(B_t^k) \\
&= \overset{b}{\mathbf{top}}\{\langle \boldsymbol{y}' \circ v,\ u \cdot p(v|\boldsymbol{x}_{\leqslant g(t)}, \boldsymbol{y}')\rangle \mid \langle \boldsymbol{y}', u\rangle \in B_t^k\}
\end{aligned}
$$

where $\mathbf{top}^b(\cdot)$ returns the top-scoring b pairs. Note we do not distinguish the revisable and non-revisable output in $\boldsymbol{y}'$ for simplicity. We also define the multi-step advance beam search function with recursive fashion as follows:

$$\mathrm{next}_i^b(B_t^k) = \mathrm{next}_1^b(\mathrm{next}_{i-1}^b(B_t^k))$$

When the opportunistic decoding window is w at decoding step t, we define the beam search over $w+1$ (include the original output) as follows:

$$\langle \boldsymbol{y}'_t, u_t\rangle = \mathbf{top}^1\big(\mathrm{next}_{n+w}^b(B_t^0)\big) \tag{3.6}$$

where $\mathrm{next}_{n+w}^b(\cdot)$ performs a beam search with $n+w$ steps, and generate $\boldsymbol{y}'_t$ as the outputs which include both original and opportunistic decoded words. n represents the length of $\boldsymbol{y}_t$

3.3.3 Revision-aware AL and Revision Rate

We define, for the first time, two metrics for revision-enabled simultaneous translation.

Revision-aware AL AL is introduced in [Ma et al., 2019a] to measure the average delay for simultaneous translation. Besides the limitations that are mentioned in [Cherry and Foster, 2019], AL is also not sensitive to the modifications to the committed words. Furthermore, in the case of re-translation, AL is incapable to measure the meaningful

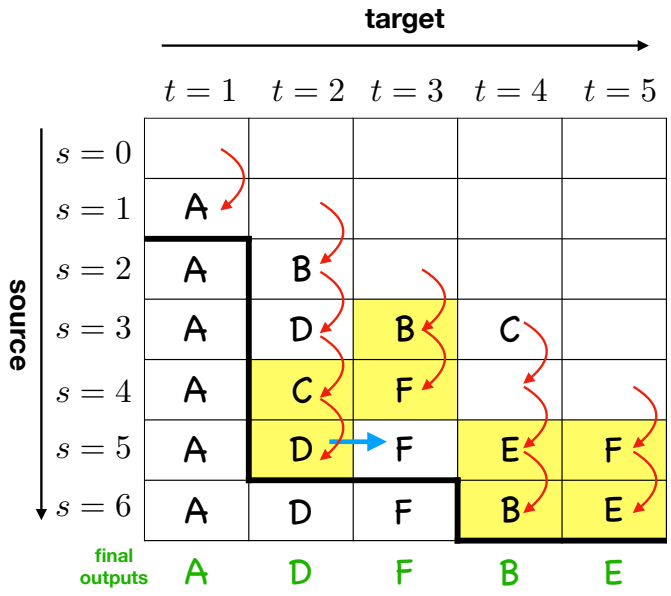


Figure 3.10: The red arrows represent the changes between two different commits, and the last changes for each output word is highlighted with yellow.

latency anymore.

We hereby propose a new latency, Revision-aware AL (RAL), which can be applied to any kind of translation scenarios, i.e., full-sentence translation, use re-translation as simultaneous translation, fixed and adaptive policy simultaneous translation. Note that for latency and revision rate calculation, we count the target side difference respect to the growth of source side. As it is shown in Fig. 3.10, there might be multiple changes for each output words during the translation, and we only start to calculate the latency for this word once it agrees with the final results. Therefore, it is necessary to locate the last change for each word. For a given source side time s, we denote the t^{th} outputs on target side as $f(x_{\leqslant s})_t$. Then we are able to find the Last Revision (LR) for the t^{th} word

on target side as follows:

$$LR(t) = \underset{s<|x|}{\textbf{argmax}} \left(f(x_{\leqslant(s-1)})_t \neq f(x_{\leqslant s})_t \right)$$

From the audience point of view, once the former words are changed, the audience also needs to take the efforts to read the following as well. Then we also penalize the later words even there are no changes, which is shown with blue arrow in Fig. 3.10. We then re-formulate the $\overline{LR}(t)$ as follows (assume $\overline{LR}(0) = 0$):

$$\overline{LR}(t) = max\{\overline{LR}(t-1), LR(t)\} \tag{3.7}$$

The above definition can be visualized as the thick black line in Fig. 3.10. Similar with original AL, our proposed RAL is defined as follows:

$$\mathrm{RAL}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{\tau(|\boldsymbol{x}|)} \sum_{t=1}^{\tau(|\boldsymbol{x}|)} \overline{LR}(t) - \frac{t-1}{r} \tag{3.8}$$

where $\tau(|\boldsymbol{x}|)$ denotes the cut-off step, and $r = |y|/|x|$ is the target-to-source length ratio.

Revision Rate Since each modification on the target side would cost extra effort for the audience to read, we penalize all the revisions during the translation. We define the revision rate as follows:

$$\Big(\sum_{s=1}^{|x|-1} \mathrm{dist}\Big(f(x_{\leqslant s}), f(x_{\leqslant s+1}) \Big) \Big) \Big/ \Big(\sum_{s=1}^{|x|} |f(x_{\leqslant s})| \Big)$$

where dist can be arbitrary distance measurement between two sequences. For simplicity, we design a modified Hamming Distance to measure the difference:

$$\text{dist}(a, b) = \text{hamming}\big(a,\ b_{\leq|a|} \circ \langle\text{pad}\rangle^{\max(|a|-|b|,0)}\big) \tag{3.9}$$

where $\langle\text{pad}\rangle$ is a padding symbol in case b is shorter than a.

3.3.4 Experiments

Datasets and Implementation We evaluate our work on Chinese-to-English and English-to-Chinese simultaneous translation tasks. We use the NIST corpus (2M sentence pairs) as the training data. We first apply BPE [Sennrich et al., 2015] on all texts to reduce the vocabulary sizes. For evaluation, we use NIST 2006 and NIST 2008 as our dev and test sets with 4 English references. We re-implement wait-k model [Ma et al., 2019a] and adaptive policy [Zheng et al., 2019d]. We use Transformer [Vaswani et al., 2017] based wait-k model and pre-trained full-sentence model for learning adaptive policy.

Performance on Wait-k Policy We perform experiments using opportunistic decoding on wait-k policies with $k \in \{1, 3, 5, 7, 9\}$, opportunistic window $w \in \{1, 3, 5\}$ and beam size $b \in \{1, 3, 5, 7, 10, 15\}$. We select the best beam size for each policy and window pair on dev-set.

We compare our proposed method with a baseline called re-translation which uses a full-sentence NMT model to re-decode the whole target sentence once a new source word is observed. The final output sentences of this method are identical to the full sentence

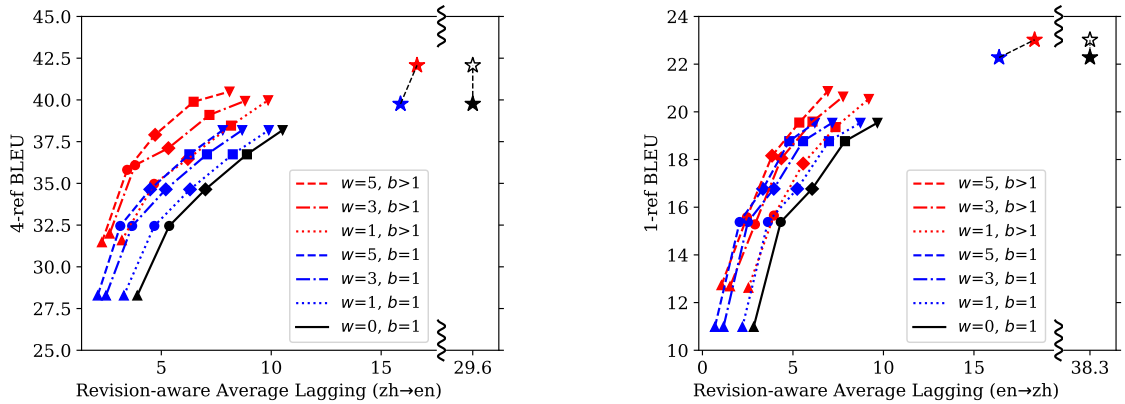


Figure 3.11: BLEU against RAL using wait-k polocies.

translation output with the same model but the latency is reduced.

Fig. 3.11 shows the results of our proposed algorithm. In this figure, ▲ ▲ ▲ represents wait-1 policies, ● ● ● represents wait-3 policies, ♦ ♦ ♦ represents wait-5 policies, ■ ■ ■ represents wait-7 policies, ▼ ▼ ▼ represents wait-9 policies,★ (★) represents re-translation with pre-trained NMT model with greedy (beam search) decoding, ★ (☆) represents full-sentence translation with pre-trained NMT model with greedy (beam search) decoding. The baseline for wait-k policies is decoding with $w = 0, b = 1$. Since our greedy opportunistic decoding doesn't change the final output, there is no difference in BLEU compared with normal decoding, but the latency is reduced. However, by applying beam search, we can achieve 3.1 BLEU improvement and 2.4 latency reduction on wait-7 policy.

Fig. 3.11(right) shows the translation quality and latency improvement by using our proposed algorithm. Compare to the Chinese-to-English translation results in previous section, there is comparatively less latency reduction by using beam search because the output translations are slightly longer which hurts the latency. As shown in

Fig. 3.12(right), the revision rate is still controlled under 8%.

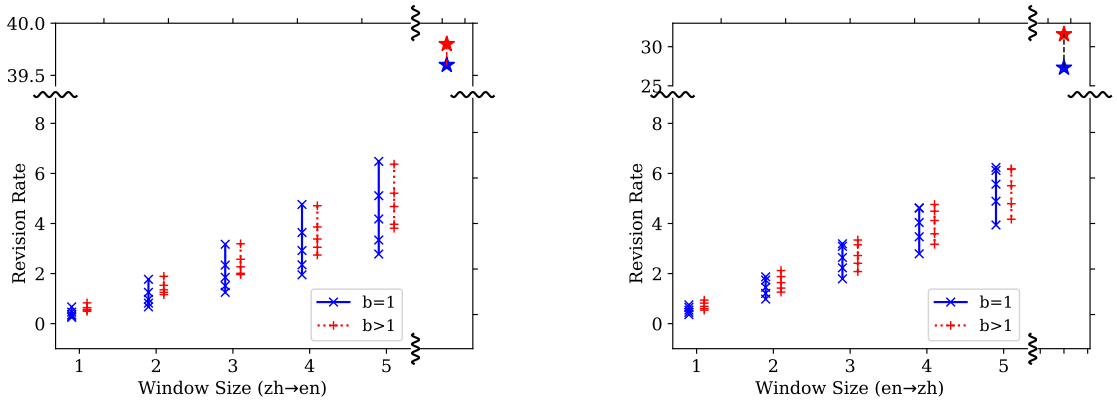


Figure 3.12: Revision rate against window size with different wait-k policies. ★ (★): re-translation with pre-trained NMT model with greedy (beam search) decoding.

Fig. 3.12 shows the revision rate with different window size on wait-k policies. In general, with opportunity window $w \leq 5$, the revision rate of our proposed approach is under 8%, which is much lower than re-translation.

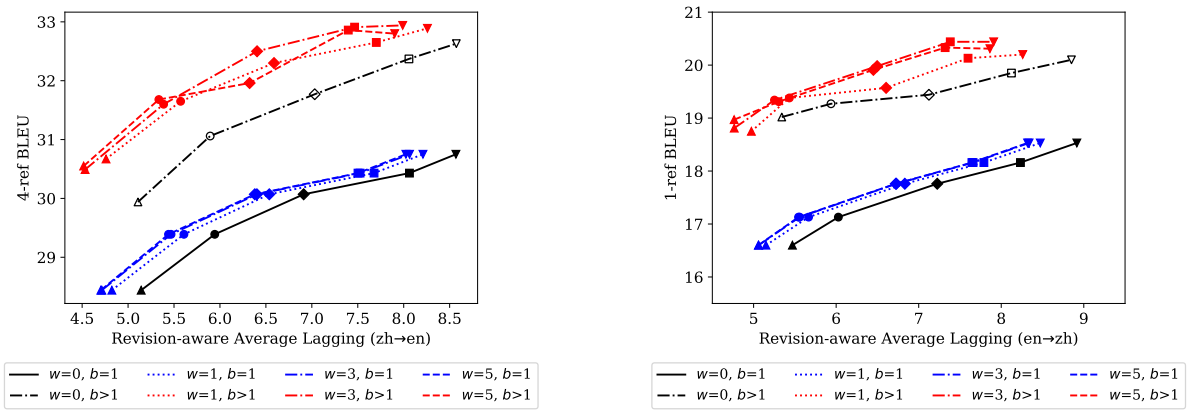


Figure 3.13: BLEU against RAL using adaptive policies. Baseline is decoded with $w = 0, b = 1$ and $w = 0, b > 1$.

Performance on Adaptive Policy Fig. 3.13 shows the performance of the proposed algorithm on adaptive policies. We use threshold $\rho \in \{0.55, 0.53, 0.5, 0.47, 0.45\}$. We vary beam size $b \in \{1, 3, 5, 7, 10\}$ and select the best one on dev-set. Comparing with conventional beam search on consecutive writes, our decoding algorithm achieves even much higher BLEU and less latency.

3.3.4.1 Revision Rate vs. Window Size

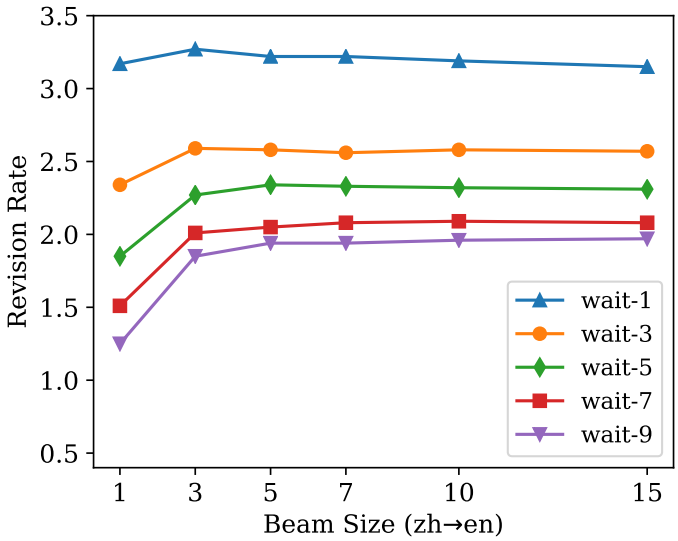


Figure 3.14: Revision rate against beam size with window size of 3 and different wait-k policies.

We further investigate the revision rate with different beam sizes on wait-k policies. Fig. 3.14 shows that the revision rate is higher with lower wait-k policies. This makes sense because the low k policies are always more aggressive and easy to make mistakes.

Moreover, we can find that the revision rate is not very sensitive to beam size.

Chapter 4: Speech-to-Speech Simultaneous Translation with Incremental TTS

Text-to-speech synthesis (TTS), which generates speech from text, is an important task with wide applications in dialog systems, speech translation, natural language user interface, assistive technologies, etc. Recently, TTS research has benefited greatly from advances in deep learning, with neural TTS systems becoming capable of generating audios with near human-level naturalness [Oord et al., 2016, Shen et al., 2018].

4.1 Background

State-of-the-art neural TTS systems generally consist of two stages: the text-to-spectrogram stage which generates an intermediate acoustic representation (linear or mel-spectrogram) from text, and the spectrogram-to-wave stage (vocoder) which converts the aforementioned acoustic representation into actual wave signals. In both stages, there are sequential approaches based on the seq-to-seq framework, as well as more recent parallel methods; the first stage, being relatively fast, is more commonly sequential [Wang et al., 2017, Shen et al., 2018, Ping et al., 2017, Li et al., 2019] with notable exceptions [Ren et al., 2019, Peng et al., 2019], while the second stage, being much slower, is more commonly parallel [Oord et al., 2017, Prenger et al., 2018, Ping et al., 2018].

Despite these successes, standard full-sentence neural TTS systems still suffer from

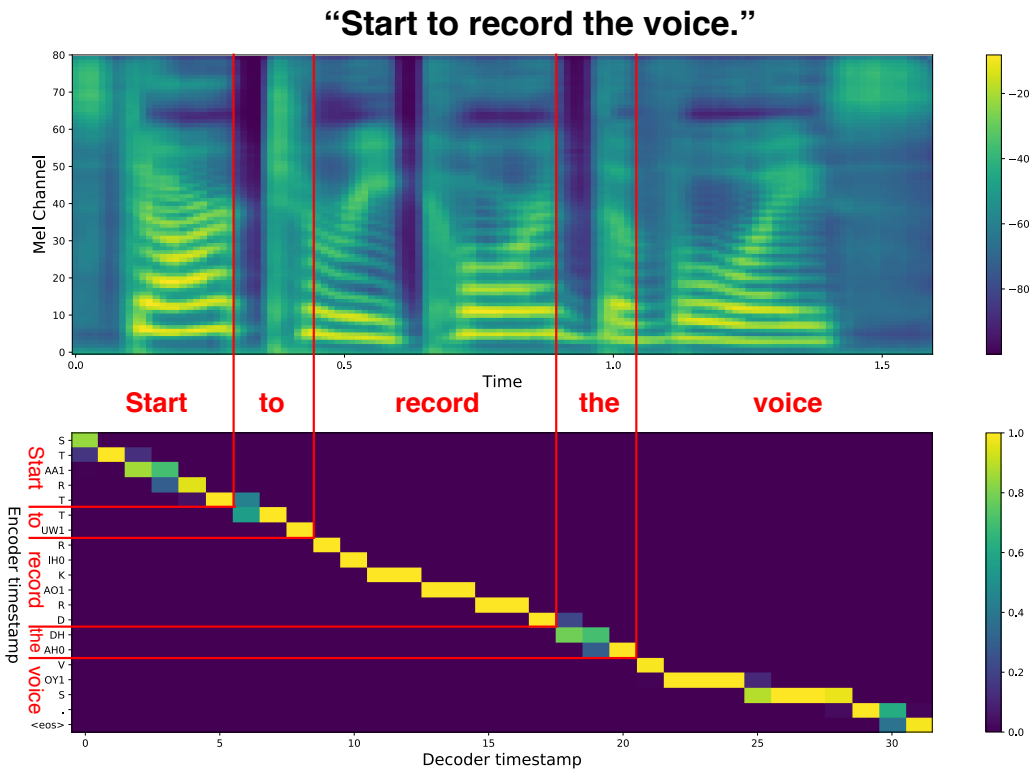


Figure 4.1: Monotonic spectrogram-to-text attention.

two types of latencies: (a) the *computational latency* (synthesizing time), which still grows linearly with the sentence length even using parallel inference (esp. in the second stage), and (b) the *input latency* in scenarios where the input text is incrementally generated or revealed, such as in simultaneous translation [Bangalore et al., 2012, Ma et al., 2019a], dialog generation [Skantze and Hjalmarsson, 2010, Buschmeier et al., 2012], and assistive technologies [Elliott, 2003]. These latencies limit the applicability of neural TTS systems; for example, in simultaneous speech-to-speech translation, the TTS module has to wait until a full sentence of translation is available, causing the undesirable delay of at least one sentence.

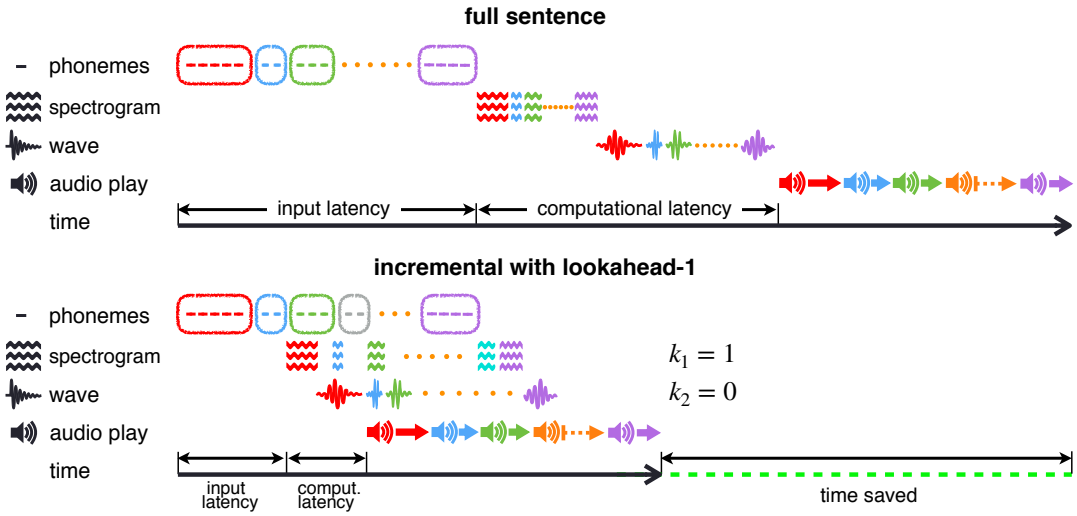


Figure 4.2: Conventional full-sentence TTS vs. our proposed incremental TTS with prefix-to-prefix framework (with $k_1 = 1$ and $k_2 = 0$ in Eq. 4.7).

To reduce these latencies, we devise the first neural incremental TTS approach based on the recently proposed prefix-to-prefix framework [Ma et al., 2019a]. Our idea is based on two observations: (a) in both stages, the dependencies on input are very local (see Fig. 4.1 for monotonic attention between text and spectrogram, for example); and (b) audio playing is inherently sequential in nature, but can be done simultaneously with audio generation, i.e., playing a segment of audio while generating the next. In a nutshell, we start to generate the spetrogram for the first word after receiving the first two words, and this spectrogram is fed into the vocoder right away to generate the waveform for the first word, which is also played immediately (see Fig. 4.2). This results in an $O(1)$ rather than $O(n)$ latency, for the very first in neural TTS. Experiments on English TTS show that our approach achieves similar speech naturalness compared to full sentence methods, but only using a fraction of time and a constant (1–2 words) latency.

Figure 4.3: Pipeline of conventional full-sentence neural TTS.

Prior to the deep learning era, there also exist several efforts on (non-neural) incremental TTS, using very different techniques.

We firstly briefly review the full-sentence neural TTS pipline to set up the notations. Then we review the prefix-to-prefix framework, which has been introduced for simultaneous translation by [Ma et al., 2019a].

4.1.1 Full-sentence TTS Pipeline

As shown in Fig. 4.3, the neural-based text-to-speech synthesis system generally has two main steps: (1) the text-to-spectrogram step which converts a sequence of textual features (e.g. characters, phonemes, words) into another sequence of spectrograms (e.g. mel-spectrogram or linear-spectrogram); and (2) the spectrogram-to-wave step, which takes the predicted spectrograms as inputs and generates the audio wave through one specific vocoder.

4.1.1.1 Step I: Text-to-Spectrogram

Conventional neural-based text-to-spectrogram frameworks (e.g. Tacotron 1[Wang et al., 2017], Tacotron 2[Shen et al., 2018], Deep Voice 3 [Ping et al., 2017], Transformer-

	data type	examples
x	scalar	a phoneme a sample in wave
$\boldsymbol{x}$	vector	a spectrogram frame
$\overline{x}$	sequence of scalars	phonemes in a word, waveform of a word
$\overline{\boldsymbol{x}}$	seq. of vectors	spectrogram of a word
$\overline{\overline{x}}$	seq. of sequences of scalars	phonemes in a sentence, waveform of a sentence
$\overline{\overline{\boldsymbol{x}}}$	seq. of sequences of vectors	spectrogram of a sent.

Table 4.1: Summary of notations. We distinguish vectors (over frequencies) and sequences (over time).

based TTS [Li et al., 2019]) employ the seq-to-seq framework to encode the source text sequence (characters or phonemes) and decode the spectrogram sequentially despite the actual computation unit choice (RNN, CNN or Transformer).

Regardless the actual design of seq-to-seq framework, with the granularity defined on words, the encoder always takes as input a word sequence $\overline{\overline{x}} = [\overline{x_1}, \overline{x_2}, ..., \overline{x_m}]$, where any word $\overline{x_t} = [x_{t,1}, x_{t,2}, ...]$ could be a sequence of phoneme or character indices, and produces another sequence of hidden states $\left[\overline{\mathbf{h}_1}, \overline{\mathbf{h}_2}, ..., \overline{\mathbf{h}_m}\right] = \overline{\overline{\mathbf{h}}} = \overline{\overline{\boldsymbol{f}}}(\overline{\overline{x}})$ to represent the textual features.

On the other side, the decoder produces the new spectrogram $\overline{\boldsymbol{y}_t}$ given the entire sequence of hidden states and the previously generated spectrogram, denoted by $\overline{\overline{\boldsymbol{y}_{<t}}} = [\overline{\boldsymbol{y}_1}, ..., \overline{\boldsymbol{y}_{t-1}}]$, where $\overline{\boldsymbol{y}_t} = [\boldsymbol{y}_{t,1}, \boldsymbol{y}_{t,2}, ...]$ is a sequence of spectrogram frames with $\boldsymbol{y}_{t,i} \in \mathbb{R}^{d_y}$. Formally, on word level, we define the inference process as follows:

$$\overline{\boldsymbol{y}_t} = \overline{\boldsymbol{\phi}}(\overline{\overline{x}}, \overline{\overline{\boldsymbol{y}_{<t}}}), \tag{4.1}$$

and for each frame within one word, we have

$$\boldsymbol{y}_{t,i} = \boldsymbol{\phi}(\overline{\overline{x}}, \overline{\overline{\boldsymbol{y}_{<t}}} \circ \overline{\overline{\boldsymbol{y}_{t,<i}}}), \tag{4.2}$$

where $\overline{\overline{\overline{\boldsymbol{y}_{t,<i}}}} = [[\boldsymbol{y}_{t,1}, ..., \boldsymbol{y}_{t,i-1}]]$, and $\circ$ represents concatenation between two sequences.

During training time, we minimizing the difference between gold spectrogram $\overline{\overline{\boldsymbol{y}^\star}}$ and model's prediction as follows:

$$L(D) = \sum_{(\overline{\overline{x}}, \overline{\overline{\boldsymbol{y}^\star}}) \in D} \sum_{t=1}^{|\overline{\overline{x}}|} \ell(\overline{\boldsymbol{y}_t^\star}, \overline{\boldsymbol{\phi}}(\overline{\overline{x}}, \overline{\overline{\boldsymbol{y}_{<t}^\star}})) \tag{4.3}$$

where ℓ can be different loss criteria, e.g. mean squared error. It is standard to use short-time Fourier transform (STFT) to obtain linear-frequency spectrogram and followed by an non-linear transform to the frequency domain to collect mel-frequency spectrogram as gold signal.

4.1.1.2 Step II: Spectrogram-to-Wave

Given a sequence of acoustic feature $\boldsymbol{y}$, the vocoder generates waveform sample vector $\overline{\overline{w}} = [\overline{w_1}, \overline{w_2}, ..., \overline{w_m}]$, where $\overline{w_t} = [w_{t,1}, w_{t,2}, ...]$, given only the linear- or mel-spectrogram without any phase information. The vocoder model model can be either autoregressive, e.g. WaveNet [Oord et al., 2016], or non-autoregressive, such as Parallel WaveNet [Oord et al., 2017], ClariNet [Ping et al., 2018], WaveGlow [Prenger et al., 2018] and FloWaveNet [Kim et al., 2019].

For the sake of both computation efficiency and near human sound quality, we choose

non-autoregressive model as our vocoder, which can be defined as follows without losing generality:

$$\overline{\overline{w}} = \overline{\overline{\psi}}(\overline{\overline{\boldsymbol{y}}}, \overline{\overline{z}}) \tag{4.4}$$

where the vocoder function $\overline{\overline{\psi}}$ takes as input a random signal $\overline{\overline{z}}$ to generate the wave signal $\overline{\overline{w}}$ conditioning on the given spectrogram $\overline{\overline{\boldsymbol{y}}}$, where $\overline{\overline{z}}$ is drawn from a simple tractable distribution, such as a zero mean spherical Gaussian distribution $\mathcal{N}(0, I)$. The length of each $\overline{z_t}$ is determined by the length of $\overline{\boldsymbol{y}_t}$, and we have $|\overline{z_t}| = \gamma \cdot |\overline{\boldsymbol{y}_t}|$. Based on different STFT procedure, γ can be 256 or 300. More specifically, the wave generation of the t^{th} word with an inverse autoregressive flow (IAF) model can be defined as follows

$$\overline{w_t} = \overline{\psi}(\overline{\overline{\boldsymbol{y}}}, \overline{\overline{z}}, t) \tag{4.5}$$

4.1.2 Prefix-to-prefix Framework

Ma et al. [2019a] propose a prefix-to-prefix framework for simultaneous machine translation. Given a monotonic non-decreasing function $g(t)$, the model would predict each target word b_t based on current available source prefix $\overline{a_{\leq g(t)}}$ and the predicted target words $\overline{b_{<t}}$:

$$p(\overline{b} \mid \overline{a}) = \prod_{t=1}^{|\overline{b}|} p(b_t \mid \overline{a_{\leq g(t)}}, \overline{b_{<t}})$$

As a simple example in this framework, they present a wait-k policy, which first wait k source words, and then alternates emitting one target word and receiving one source word. With this policy, the output is always k words behind the input. This policy can be

defined with the following function

$$g_{\text{wait-}k}(t) = \min\{k + t - 1, |\bar{b}|\}.$$

4.2 Incremental TTS

Both steps in the above conventional TTS pipeline require the fully observed source text or spectrograms as input to start the inference. In this section, we first propose a general framework to do inference at both steps with partial source information, then we present one simple specific example in this framework.

4.2.1 Prefix-to-Prefix for TTS

As shown in Fig. 4.1, there is no long distance reordering between input and output sides in the task of Text-to-Spectrogram, and the alignment from output side to the input side is monotonic. One way to utilize this monotonicity is to generate audio pieces for each word independently, that is to feed the input text each word to predict its spectrogram in step I, and then to generate audio piece based on the spectrogram. After generating audio pieces for all words, we can concatenate those audios together as the result. However, this simple incremental generation approach mostly provides some robotic and abnormal speech voice. In order to generate speech with better prosody, we need to consider some local information around the current word, when generating audio for each word. This is also necessary to connect several audio pieces smoothly.

To solve the above issue, we propose a prefix-to-prefix framework for TTS, which is

inspired by the prefix-to-prefix framework [Ma et al., 2019a] for simultaneous translation. Within this new framework, our $\overline{\boldsymbol{y}_t}$ and $\overline{w_t}$ is generated incrementally as follows:

$$\begin{aligned} \overline{\boldsymbol{y}_t} &= \overline{\boldsymbol{\phi}}(\overline{\overline{x_{\leq g(t)}}}, \overline{\overline{\boldsymbol{y}_{<t}}}), \\ \overline{w_t} &= \overline{\psi}(\overline{\overline{\boldsymbol{y}_{\leq h(t)}}}, \overline{\overline{z_{\leq h(t)}}}, t) \end{aligned} \tag{4.6}$$

where $g(t)$ and $h(t)$ are monotonic functions that define the number of words, whose information is taken as input for each step, when generating results for the t^{th} word.

4.2.2 Lookahead-k Policy

As a simple example in the prefix-to-prefix framework, we define two lookahead polices for the two steps with $h(t)$ and $g(t)$ functions respectively. These are similar to the wait-k policy introduced by Ma et al. [2019a].

$$\begin{aligned} g_{k_1}(t) &= \min\{k_1 + t, |\overline{\overline{x}}|\} \\ h_{k_2}(t) &= \min\{k_2 + t, |\overline{\overline{\boldsymbol{y}}}|\} \end{aligned} \tag{4.7}$$

Intuitively, the function g implies that we wait for the first k_1 number of words, and then generate mel spectrogram for each word continuously until the end of the input sentence. Similarly, the function h implies that we first wait for spectrograms of k_2 number of words, and then start generating audio piece for each word continuously. Combining these together, we can obtain a lookahead-k policy for the whole TTS system, where $k = k_1 + k_2$. An example of lookahead-1 policy is provided in Fig. 4.2, where we take $k_1 = 1$ for the first step and $k_2 = 0$ for the second step.

4.2.3 Incremental Generation of Spectrogram

Different from full sentence scenario, where we feed the entire source text to the encoder, we gradually provide source text input to the model word by word when more input words are available. By our prefix-to-prefix framework, we will predict mel spectrogram for the t^{th} word, when there are $g(t)$ words available. Thus, the decoder predicts the i^{th} spectrogram frame of the t^{th} word with only partial source information as follows:

$$\boldsymbol{y}_{t,i} = \boldsymbol{\phi}(\overline{\overline{x_{\leq g(t)}}}, \overline{\overline{\boldsymbol{y}_{<t}} \circ \overline{\boldsymbol{y}_{t,<i}}}) \tag{4.8}$$

where $\overline{\overline{\boldsymbol{y}_{t,<i}}} = [[\boldsymbol{y}_{t,1}, ..., \boldsymbol{y}_{t,i-1}]]$ represents the first $i-1$ spectrogram frames in the t^{th} word.

In order to obtain the corresponding relationship between the predicted spectrogram and the currently available source text, we rely on the attention alignment applied in our decoder, which is usually monotonic. To the i^{th} spectrogram frame of the t^{th} word, we can define the attention function $\boldsymbol{\sigma}$ in our decoder as follows

$$\mathbf{c}_{t,i} = \boldsymbol{\sigma}(\overline{\overline{x_{\leq t+1}}}, \overline{\overline{\boldsymbol{y}_{<t}} \circ \overline{\boldsymbol{y}_{t,<i}}}) \tag{4.9}$$

The output $\mathbf{c}_{t,i}$ represents the alignment distribution over the input text for the i^{th} predicted spectrogram frame. And we choose the input element with the highest probability as the corresponding input element for this predicted spectrogram, that is, $\mathbf{argmax}\, \mathbf{c}_{t,i}$. When we have $\mathbf{argmax}\, \mathbf{c}_{t,i} > \sum_{\tau=1}^{t} |\overline{x_\tau}|$, it implies that the i^{th} spectrogram frame corresponds to the $(t+1)^{th}$ word, and all the spectrogram frames for the t^{th} word are predicted.

For models like Deep Voice 3, there are usually multiple attention layers in the decoder. For our implementation, we only consider the alignment obtained from the first attention layer. When the encoder observes the entire source sentence, a special symbol <eos> was feed into the encoder, and the decoder continue to generate spectrogram word by word. The decoding process ends when the binary "stop" predictor of the model predicts the probability larger than 0.5.

4.2.4 Generation of Waveform

After we obtain the predicted spectrograms for a new word, we feed them into our vocoder to generate waveform. Since we use a non-autoregressive vocoder, we can generate each audio piece for those given spectrograms in the same way as full sentence generation. Thus, we do not need to make modification on the vocoder model implementation. Then the straightforward way to generate each audio piece is to apply Eq. 4.5 at each step t conditioned on the spectrograms of each word $\overline{\boldsymbol{y}_t}$. However, when we concatenate the audio pieces generated in this way, we observe some noise at the connecting part of two audio pieces.

To avoid such noise, we sample a long enough random vector as the input vector $\overline{\overline{z}}$ and fix it when generating audio pieces. Further, we append additional δ number of spectrogram frames to the each side of the current spectrograms $\overline{\boldsymbol{y}_t}$ if possible. That is, at most δ number of last frames in $\overline{\boldsymbol{y}_{t-1}}$ are added in front of $\overline{\boldsymbol{y}_t}$, and at most δ number of first frames in $\overline{\boldsymbol{y}_{t+1}}$ are added at the end of $\overline{\boldsymbol{y}_t}$. This may give a longer audio piece than we need, so we can remove the extra parts from that. Formally, the generation procedure

of wave for each word can be defined as follows

$$\overline{w_t} = \overline{\psi}(\overline{\overline{\boldsymbol{y}_{[t-1:h(t)]}}}, \overline{\overline{z_{[t-1:h(t)]}}}, t) \tag{4.10}$$

where $\overline{\overline{\boldsymbol{y}_{[t-1:h(t)]}}} = [\overline{\boldsymbol{y}_{t-1}}, \ldots, \overline{\boldsymbol{y}_{h(t)}}]$ and $\overline{\overline{z_{[t-1:h(t)]}}} = [\overline{z_{t-1}}, \ldots, \overline{z_{h(t)}}]$.

4.2.5 Experiments

4.2.5.1 Experimental Setup

For simplicity, we assume the given input are all phonemes in our experiments, and we will include the text-to-phoneme model in the future version. In our experiments, we work on a chunk-level which consists of one or more words depending on a hyper-parameter l. That is, a chunk consists of minimum number of words such that the number of phonemes in this chunk is at least l. We use chunk instead of single word because the pronunciation of some words may be too short and they may affect the performance and efficiency of our system.

Datasets We use an internal English speech dataset containing 13,708 audio clips from a female speaker and the corresponding phoneme transcripts. The total audio lengths is about 20 hours and the sampling rate of the audio is 48 kHz. We downsample that to 24 kHz. We remove all intermediate punctuation marks in the transcripts, and randomly split the dataset into three sets: 13,158 samples for training, 275 samples for validation and 275 samples for testing. Our mel-scale spectrogram has 80 bands, and is computed

through a short time Fourier transform (STFT) with window size of 1200 and hop size of 300.

Model Configuration We use the fully-convolutional text-to-spectrogram architecture introduced in Deep Voice 3 (DV3) [Ping et al., 2017] as our phoneme-to-spectrogram model. The original DV3 architecture consists of three components: an encoder, an decoder and a converter. We do not use the converter part since we only need the mel spectrogram output for our vocoder. Our encoder consists of eleven convolution blocks, and our decoder consists of seven convolution blocks. Different from the original architecture in DV3, our decoder only contains two attention blocks: one for the first convolution block and one for the last convolution block.

We use a 60-layer Gaussian inverse autoregressive flow (IAF) model introduced in ClariNet [Ping et al., 2018] as our waveform synthesizer (i.e. vocoder). The model consists of four stacked Gaussian IAF blocks, which are parameterized by [10, 10, 10, 30]-layer WaveNets with 64 residual channels, 64 skip channels and filter size 3 in dilated convolutions. We distill this model from a 20-layer Gaussian autoregressive WaveNet, which have the same architecture as the teacher WaveNet model in ClariNet.

Training We separately train the DV3 model and the ClariNet vocoder. To train the DV3 model, we follow the original DV3 paper [Ping et al., 2017] and add the *guided attention loss* into our training loss, which is introduced by Tachibana et al. [2018] to improve the efficiency of training. We use the Adam optimizer with batch size of 16 to train this model on NVIDIA GTX TITAN X GPU. We follow the original ClariNet

paper [Ping et al., 2018] to train the teacher autoregressive WaveNet and distill the student Gaussian IAF, which are both trained using ground truth mel-spectrograms and audio waveforms.

Inference For inference, we apply the monotonic constraint introduced for DV3 to obtain a better attention. Specifically, we compute the softmax function over a fixed window instead of all generated encoder hidden states, which starts from the last attended position and has window size of 3. For our proposed method in the following sections, we consider two different policies: lookahead-1 policy for step I and lookahead-0 policy for step II (where we set $\delta = 0$), giving a lookahead-1 policy for the system; and lookahead-1 policy for both steps (where we set $\delta = 30$ for the second step), giving a lookahead-2 policy for the system. In the following sections, we only consider lookahead-k policy for the TTS system instead of each step.

4.2.5.2 Audio Quality

We first compare the audio quality generated from different methods. For this purpose, we choose 50 sentencesfrom our test set and generate audio samples for these sentences with different methods, which include (1) *Ground Truth Audio*; (2) *Ground Truth Mel*, where we convert the ground truth mel spectrograms into audio samples using our vocoder; (3) *Full-sentence*, where we first predict all mel spectrograms given the full sentence text and then convert those to audio samples; (4) *Lookahead-2*, where we incrementally generate audio samples with lookahead-2 policy; (5) *Lookahead-1*, where

we incrementally generate audio samples with lookahead-1 policy. For our method, we choose as 18 the least length of the first chunk and for other chunks we choose 6 as the least length. The MOS (Mean Opinion Score) of the evaluation is provided in Table 4.2.

Method	MOS
Ground Truth Audio	4.12 ± 0.10
Ground Truth Mel	4.05 ± 0.07
Full-sentence	3.56 ± 0.08
Lookahead-2	3.46 ± 0.08
Lookahead-1	3.37 ± 0.09

Table 4.2: Mean Opinion Score (MOS) ratings with 95% confidence intervals.

From Table 4.2, we can see that with lookahead-2 policy we can generate high quality audio similar to the full-sentence method, and the MOS decreases slightly with lookahead-1 policy. We also provide the relation between lookahead and MOS in Fig. 4.4. We can see there is a trade-off between lookahead numbers and audio quality (measured by MOS).

4.2.5.3 Latency

In this section, we compare the latency of full-sentence method and our proposed method. We consider two different scenarios: (1) when all text input is available; and (2) the text input is provided incrementally as audio playing speed. The first scenario is the common monolingual text-to-speech application, while the second scenario is more similar to the simultaneous translation application.

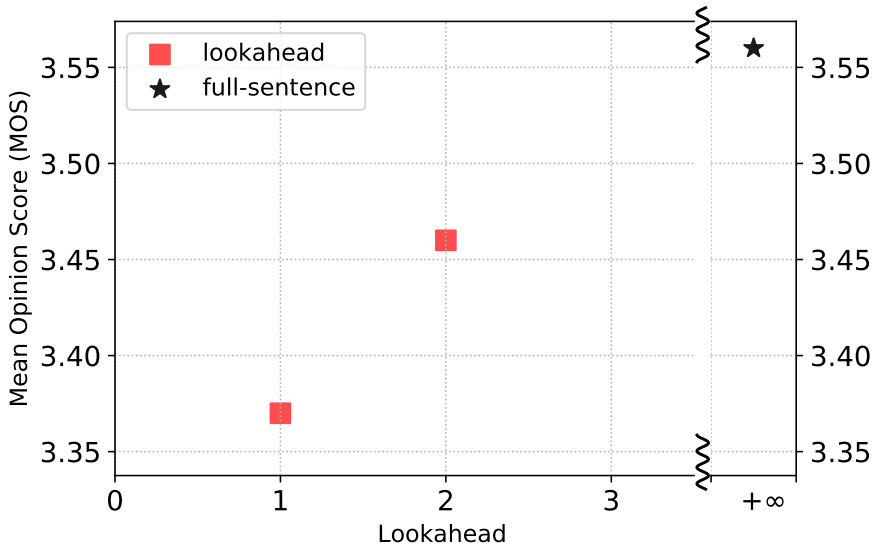


Figure 4.4: Trade-off between lookahead and MOS.

4.2.5.4 All Input Available

For full-sentence generation, the latency will be the generation time of the whole audio sample; while for our proposed incremental method, the latency will be the generation time of the first phoneme chunk if the next audio piece can be generated before the previous audio piece is finished playing. (We will show this later.) So we compare the generation time of sentence with different lengths, which is averaged over sentences with the same length. Here we choose as 18 the least length of the first chunk and for other chunks we choose 6 as the least length. We do inference on the test set on both CPU and GPU and provide the results in Fig. 4.5.

We can see the latency of full-sentence method increases along with the number of the phonemes in the sentence, which is more than 1 second on GPU for sentences with length larger than 60 and could be more than 4 seconds on CPU for those sentences;

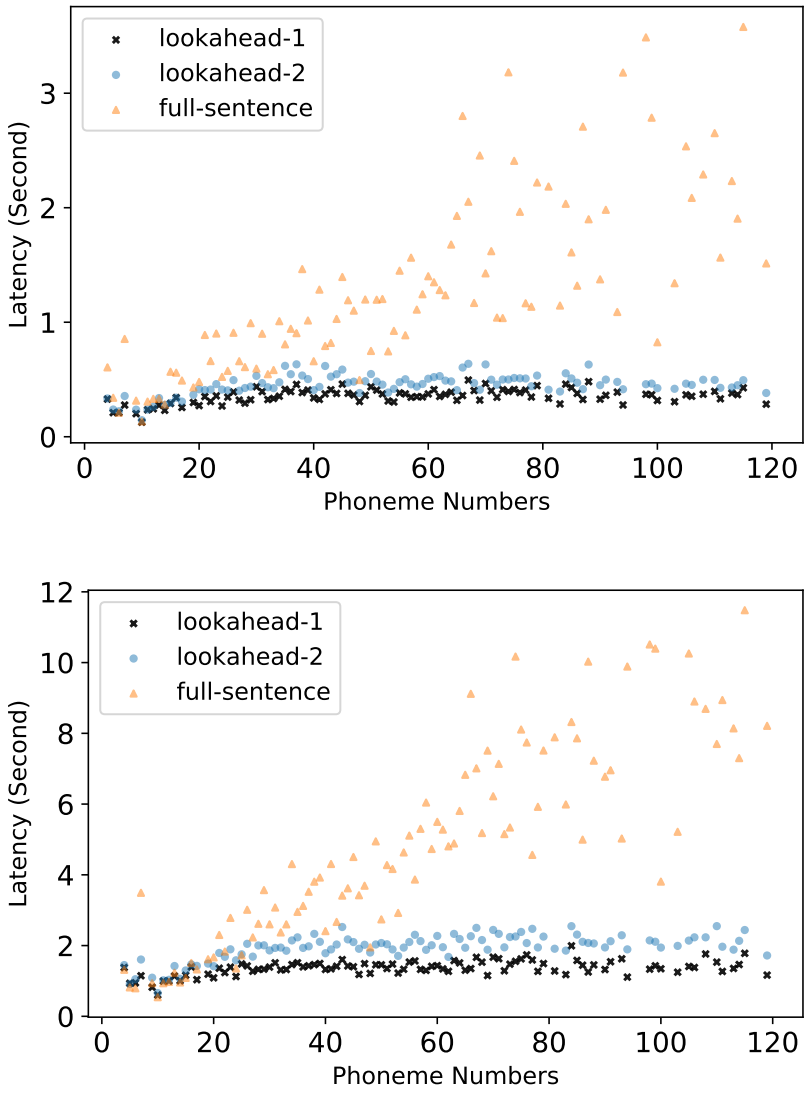


Figure 4.5: Averaged latency of different methods when all input is available. Left: results on GPU. Right: results on 80-core CPU.

while our incremental method has a constant latency for sentences with different lengths: its latency on average is less than 0.5 seconds on GPU and less than 2.5 seconds.

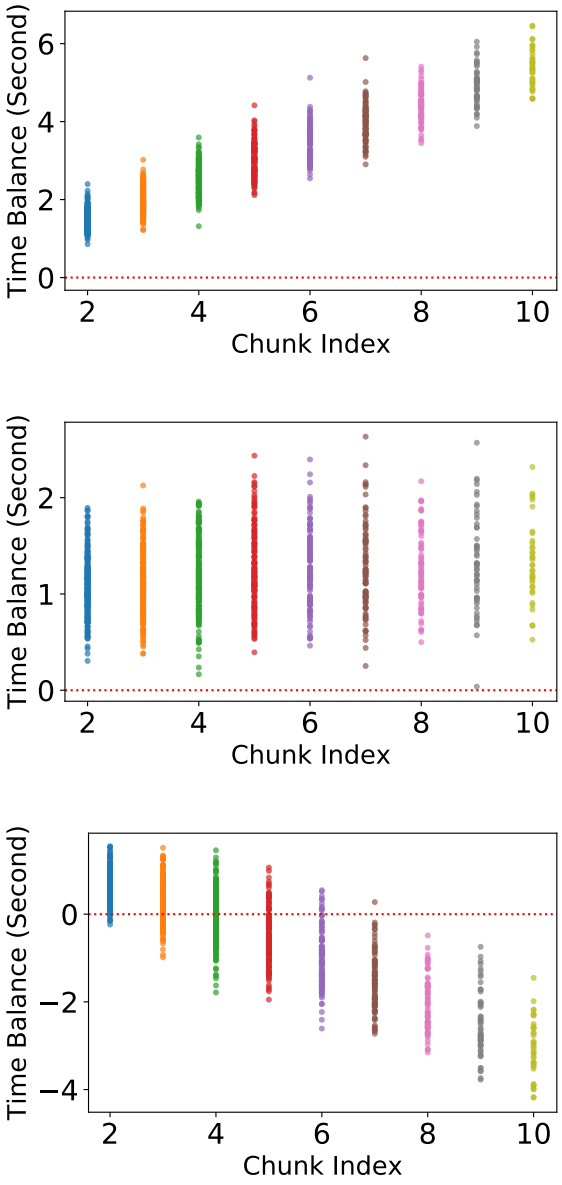


Figure 4.6: Left time on different chunk positions. Left: lookahead-2 results on GPU. Middle: lookahead-1 results on 80-core CPU. Right: lookahead-2 results on 80-core CPU.

Continuity We now show that our method can generate audio piece for each chunk fast enough such that the next audio piece will be generated before we finish playing the previous audios, i.e. the generated audios can be played continuously without interruptions. At each generation step, the available time to generate the current audio piece is equal to the whole previous audio time minus the generation time of all previous chunks but the first one. Starting from the second chunk, as long as the current available time is larger than the generation time of the current chunk, we can continuously play the current generated audio piece without any interruption. That is, the remaining time (time balance) from the available time after taking the generation time should be larger than zero at each step. To evaluate this case, we compute the time balance at several steps and show the results in Fig. 4.6. We find that the time balance is always larger than 0 on GPU with lookahead-2 policy, while on CPU the time balance is less than that on GPU. If we apply lookahead-1 policy, then the time balance is always larger than 0; if we apply lookahead-2 policy on CPU, then the time balance will be less than 0 starting from the second chunk. This is because the lookahead-2 policy needs to generate audio piece with appended mel spectrogram, which causes more computing time.

4.2.5.5 Input Provided Incrementally

When the text input (such as obtained from speech or translation system) is given incrementally, the latency of full-sentence method will be more than the running time, since it needs to wait for the whole text to finish to start generation. Similarly, the latency of our proposed method should be at least the sum of generation time of the first phoneme

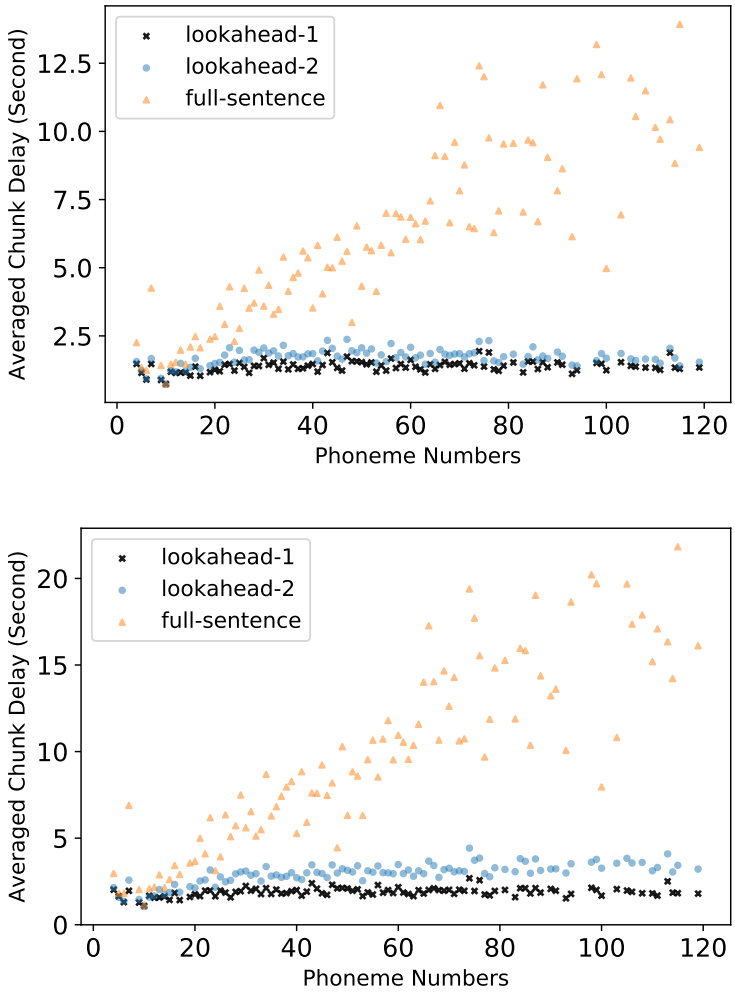


Figure 4.7: Averaged latency results when input is provided incrementally. Left: results on GPU. Right: results on 80-core CPU.

chunk and the time waited for enough input to start generation. To mimic this scenario, we consider the experiment where the goal is to repeat the sentence from the speaker as soon as possible after the speaker starts speaking (this is also called *shadowing*). Here we define *averaged chunk delay* as our latency metrics, which is the averaged lag time

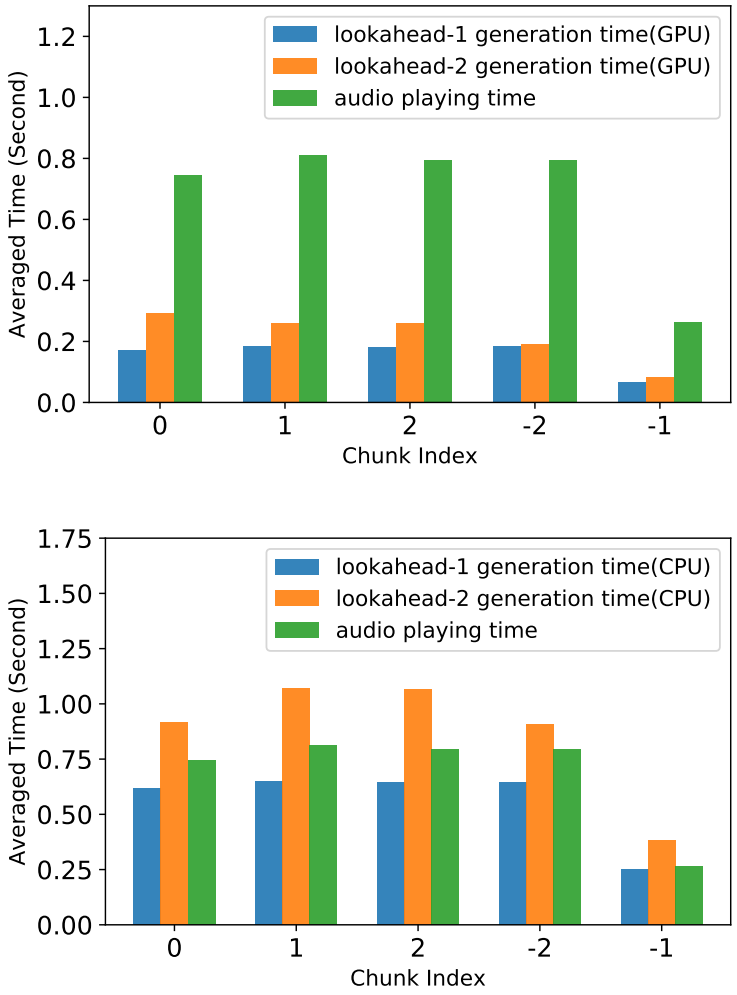


Figure 4.8: Averaged generation time and audio playing time on different chunk positions. Left: results on GPU. Right: results on 80-core CPU.

between the ending time of each input chunk and the ending time of its corresponding generated audio piece.

We choose the ground-truth audios from test set as the inputs and extract the ending time of each chunk in those audios by the Montreal Forced Aligner [McAuliffe et al.,

2017]. The ending time of our chunk can be obtained with the generation time, audio playing time and input chunk ending time. For this experiment, we choose as 7 the least length of all chunks and the latency results are averaged over sentences with the same length. These results are provided in Fig. 4.7.

We find that the latency of our method is almost constant for different sentence lengths, which is less than 3 seconds on GPU and 4.5 seconds on CPU; while the latency of full-sentence method increases along with the number of the phonemes in the sentence. This is similar to the results from previous scenario, but its magnitude is larger than that in the previous scenario because of the waiting time for the input.

Generation Speed for Chunks When input is given incrementally , we cannot always guarantee the continuity of the generated audio pieces since we do not have control on the source audio speed. That is, the interruption is unavoidable when the next chunk is given after the previous audio piece finishes. But we can compare averaged generation time and audio playing time for chunks to show that, the generation is faster than the audio playing, and our method can keep a low latency.

In the next experiment, we consider the sentence that have more than 4 chunks and compute the averaged generation time and audio playing time for the first three chunks and the last two chunks. The results are provided in Fig. 4.8.

We find that the averaged generation time of lookahead-1 policy on GPU is about 25% of the audio time for all the five chunks, and that of lookahead-2 policy is about 30% of the audio time for it has appended mel spectrogram chunk for vocoder. The generation time is much higher on CPU, which is about 80% of the audio playing time

except for the last chunk for lookahead-1 policy, and about 130% of the audio playing time for lookahead-2 policy. These show that with lookahead-1 policy the generation of audio piece for each chunk can be faster than real-time on GPU and CPU.

Effects of CPU Numbers In the above experiments, we use a machine with 80-core CPU, which may be expensive to use in practice. So we evaluate the ratio of generation time and audio time on different number of CPU's to understand the effects of CPU numbers. The results are provided in Fig. 4.9. We can see that the averaged ratio is smaller than 1 for lookahead-1 policy when the CPU number is at least 40, but for lookahead-2 policy this ratio is still larger than 1 even with 80-core CPU.

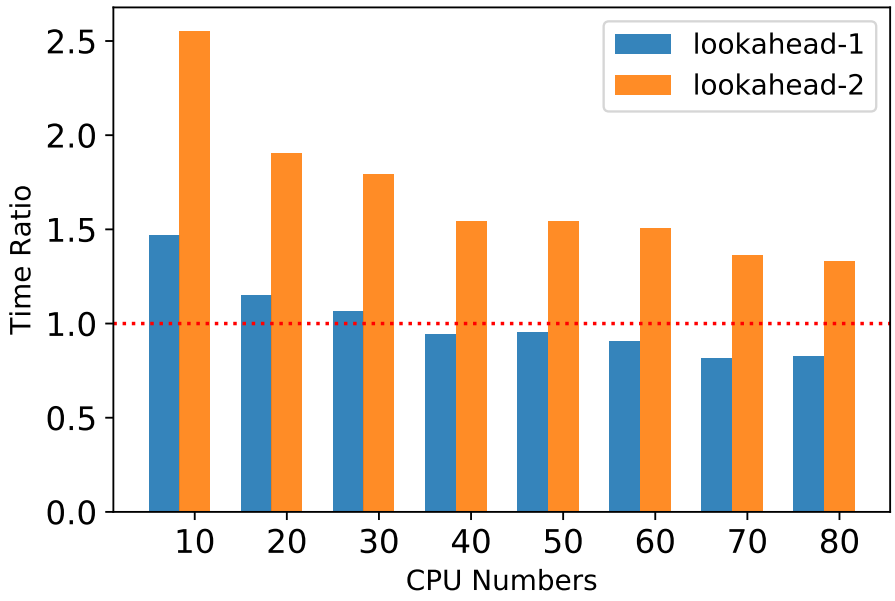


Figure 4.9: Averaged ratio of generation time and audio time on different number of CPU's.

4.2.5.6 Analysis

In this section, we provide some analysis on attention and generated mel spectrogram to understand our method better. Our method trace the relationship between mel spectrogram frame and input phonemes in decoding step based on learned attention alignment. So we compare the alignment of the first attention layer from the full-sentence method and from our lookahead-1 policy in Fig. 4.10. We can see in this figure that the first two alignments are very similar, implying that our method will maintain the attention as full-sentence method, although we incrementally encoding given phonemes. And since the alignment is monotonic, we can usually get the correct correspondence from predicted mel spectrogram to input phonemes in decoding step.

We visualize the predicted mel spectrogram from different methods to make a comparison. We consider mel spectrograms from four different methods: (1) generated from ground truth audio wave, (2) predicted with full-sentence method, (3) predicted with lookahead-1 policy.These are provided in Fig. 4.11. From the visualization, we can see that the mel spectrogram predicted with lookahead-1 policy is very similar to that predicted by the full-sentence method, which is also similar to the ground truth. These show that our method can predict good mel spectrograms.

4.3 Speech-to-Speech Simultaneous Translation

In previous chapters, we have discussed our existing work of different policies and models for simultaneous translation, beam search for simultaneous translation and incremental Text-to-Speech.

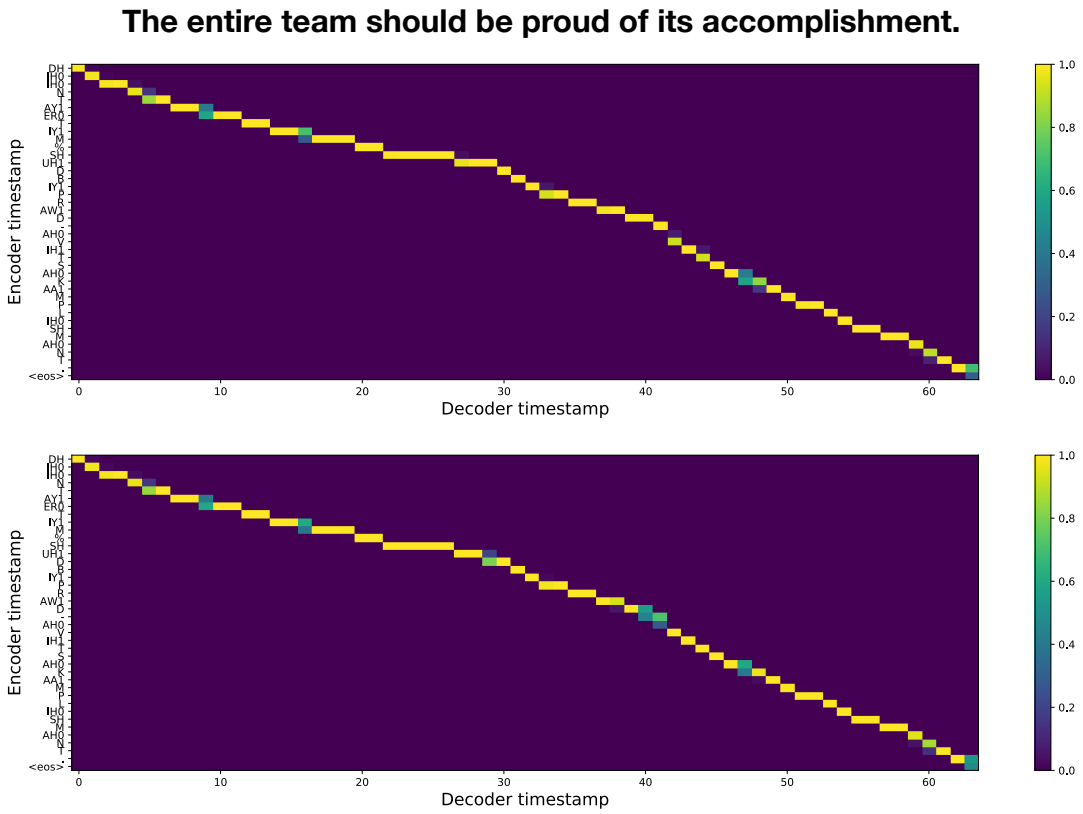


Figure 4.10: Top: alignment from full-sentence method. Bottom: alignment from lookahead-1 policy.

Based on these work, we plan to further explore speech-to-speech simultaneous interpretation and compare the translation quality and latency with human simultaneous interpreter.

We propose to incorporate the streaming Automatic Speech Recognition (ASR) system (e.g. Baidu string ASR) and the our incremental text-to-speech system with the simultaneous translation system. The difficulties are how to control the latency and how to error between different layers. The output of streaming ASR is always very noisy. One special kind of noise is the homophone noise, where words are replaced by other words

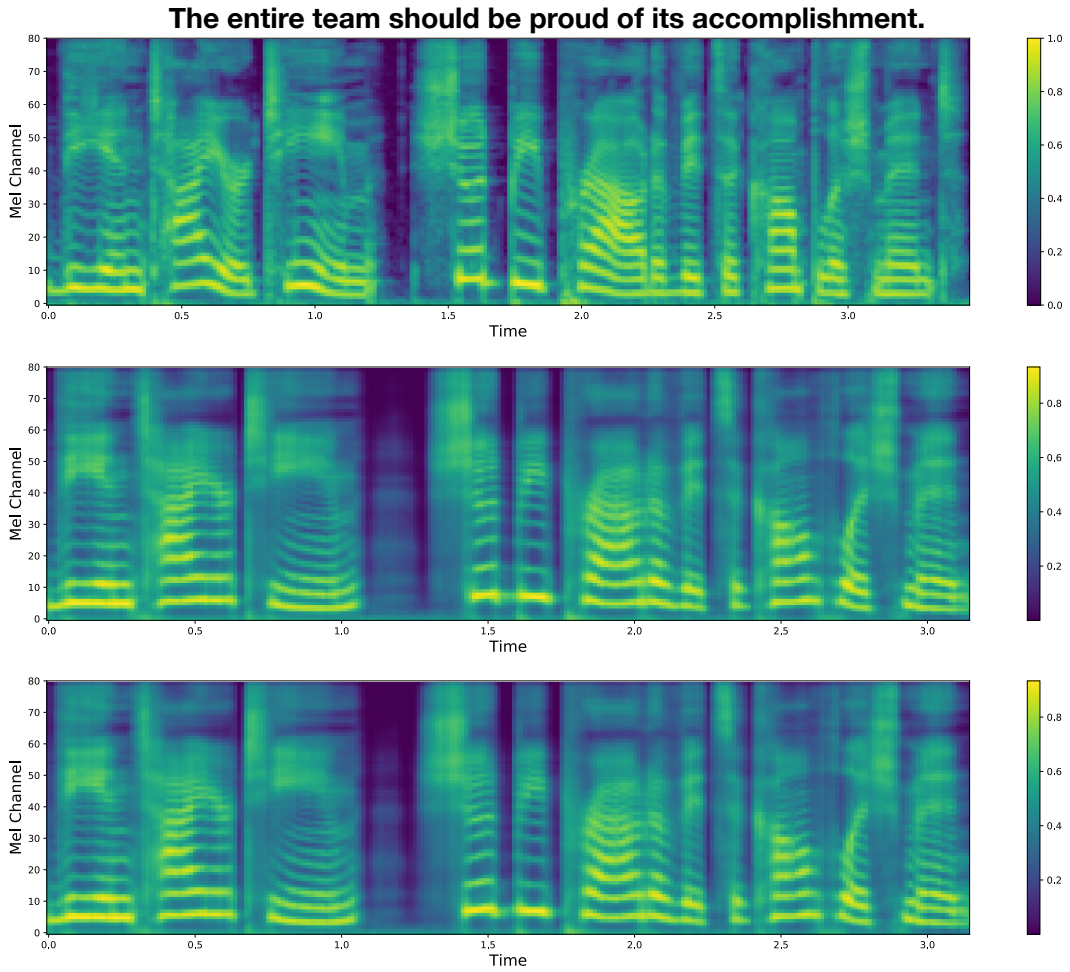


Figure 4.11: Top to Bottom: ground-truth mel, mel predicted by full-sentence method, mel predicted with lookahead-1 policy.

with similar pronunciations, is very popular in the output of ASR. Meanwhile, most streaming ASR product doesn't output punctuations unless a full sentence is recognized. To tackle this problem, we propose to mix the dataset with input sentences without punctuations [Zheng et al., 2019c, 2018b] [Zheng et al., 2018c] [Zheng et al., 2018a].

4.3.1 Speech-to-Text Simultaneous Translation and Human Interpreter

In this section, we first compare the speech-to-text simultaneous translation with human interpreters.

Name	Languages	Hours	Domain
LCD United Nations Proceedings	Ar, Zh, En, Fr, Ru, Es	8,500	Conference
European Parliament Interpreting	En, It, Es	858	Conference
NAIST Simultaneous Translation	Ja, En	22	Talk, News

Table 4.3: Human Simultaneous Interpretation Corpora.

There are many human simultaneous interpretation dataset available (e.g. EPIC, NAIST, LDC UN). Table 4.3 shows three available human simultaneous interpretation corpora. We propose to compare our speech-to-speech simultaneous interpretation system with human simultaneous interpreter.

Figure 4.12 shows the result of our simultaneous translation system with Google Streaming ASR on EPIC Es-En portion compared with the result of human simultaneous interpreter. This result is not fair enough because our system is a speech-to-text simultaneous translation system which can commit several words at once but human interpreter need time to pronounce each word. The BLEU and AL are measure on corpus level. We observe large latency in the streaming ASR whose average lagging is 3.52.

4.3.2 Speech-to-Speech Simultaneous Translation and Human Interpreter

We further explore speech-to-speech simultaneous translation by incorporating our incremental TTS into the full pipeline. Incorporating incremental TTS introduces more

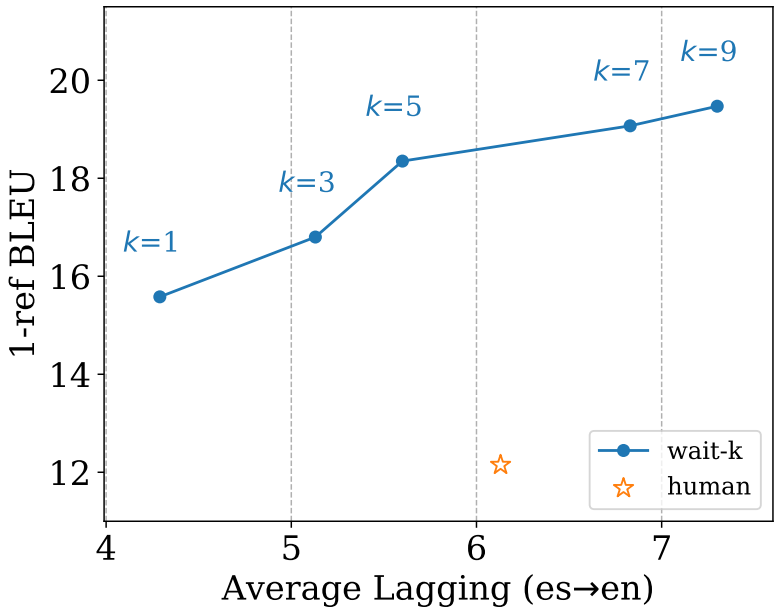


Figure 4.12: Comparison of speech-to-text simultaneous translation quality against latency with human simultaneous interpreter.

difficulties to control the latency. Inspired by human simultaneous interpreters, who automatically adjust their speaking speed to catch-up the source speaker, we propose to add different kinds of pauses in our incremental TTS model to adjust the speed of our output speech. These pause information is regarded as prosody information in classic speech analysis and is involved in most speech synthesis corpus.

In this experiment, we use Baidu public streaming ASR API as speech recognizer. We use WMT 2018 Chinese-to-English translation corpus to train the translation model. We use DataBaker 10,000 sentences Chinese TTS dataset to train the incremental TTS system. We test our system on an English speech from LDC United Nation simultaneous translation dataset. We also use the corresponding Chinese interpreter's translation as

the performance of human interpreter. We use Ear-voice-span, which is popular in simultaneous interpretation literature, as latency metric in this experiment.

	BLEU	Latency (Ear-voice-span)
Human Interpreter	7.87	2.6 seconds
Our System	16.29	2.7 seconds

Table 4.4: Translation quality and latency of human interpreter and our system

Table 4.4 shows the result of comparing our system with human interpreter. We can see that our system can achieve much higher BLEU score with similar latency. This is because human interpreter tends to use shorter phrases or even summarizing the source speech. For example, table 4.5 shows an running example that human interpreter translates "nuclear- and non-nuclear-weapon States" into "核国家" (nuclear country) and "无核国家" (non-nuclear country) while our system still translates it into "核武器和无核武器国家" (nuclear-weapon and non-nuclear-weapon country). The human interpreter's translation is not right, strictly speaking.

Speech	Nuclear- and non-nuclear-weapon States ...
ASR Output	nuclear and non-nuclear weapon States ...
Our Translation	核武器 (nuclear-weapon) 和 (and) 无核武器国家 (non-nuclear-weapon country) ...
Human	核国家 (nuclear country)，无核国家 (non-nuclear country) ...

Table 4.5: Running example of our Speech-to-speech simultaneous translation. Human interpreter summarizes "nuclear-weapon and non-nuclear-weapon country" into "nuclear country, non-nuclear country".

Table 4.6 shows an example that sometimes the error from streaming ASR will propagate to our system's translation. In this example, the ASR recognizes "New Zealand" into "museum" which leads our system to translate "New Zealand" into "museum".

Speech	New Zealand's commitment to nuclear disarmament and nonproliferation is ...
ASR Output	museum's commitment to nuclear disarmament and nonproliferation is ...
Our Translation	博物馆 (museum) 的工作，是致力于核裁军和不扩散的 ...
Human	新西兰 (New Zealand) 对这个，裁军和军控的承诺 ...

Table 4.6: Running example of our Speech-to-speech simultaneous translation. Our system translates "New Zealand" into "museum" because of the error in ASR output.

Chapter 5: Summary

In this dissertation, we reviewed a series of work on neural-based simultaneous translation. We proposed several translation models, decoding algorithms and an incremental Text-to-speech model. We also concatenated these modules and tested a speech-to-speech simultaneous translation system.

We first present several simultaneous translation approaches based on the prefix-to-prefix framework which overcomes the limitations of previous work (Chapt. 2). Besides this wait-k fixed policy, we also proposed a flexible policy learned on pre-trained model and a flexible policy learned from scratch. Experiments show that these methods can achieve high translation accuracy and low latency as well.

Then we investigate a beam search algorithm which is able to further improve the translation quality of different simultaneous translation models (Chapt. 3). Experiments on three approaches to simultaneous translation demonstrate effectiveness of our method. Based on this algorithm, we also proposed an opportunistic decoding timely correction technique which improves the latency and quality for simultaneous translation. We also defined two metrics for revision-enabled simultaneous translation for the first time.

In the end, we introduce an incremental Text-to-speech model which is a critical component of our final speech-to-speech simultaneous translation system (Chapt. 4). We have presented a prefix-to-prefix inference framework for incremental TTS system, and a lookahead-k policy that the audio generation is always k words behind the input. We show

that this policy can maintain good audio quality compared with full-sentence method and can achieve low latency on different scenarios: when all the input are available and when input is given incrementally. We finally concatenated this incremental TTS system with a streaming ASR and previous proposed simultaneous translation models. This pipeline system can achieve very high accuracy as well as low latency at the same time.

Bibliography

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. Prediction improves simultaneous neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3022–3027, October-November 2018. URL https://www.aclweb.org/anthology/D18-1337.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014a.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014b.

Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour, and Aura Jimenez. Real-time incremental speech-to-speech translation of dialogs. In *Proc. of NAACL-HLT*, 2012.

Hendrik Buschmeier, Timo Baumann, Benjamin Dosch, Stefan Kopp, and David Schlangen. Combining incremental language generation and incremental speech synthesis for adaptive information presentation. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 295–303. Association for Computational Linguistics, 2012.

Colin Cherry and George Foster. Thinking slow about latency evaluation for simultaneous machine translation. *arXiv preprint arXiv:1906.00048*, 2019.

Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina, et al. State-of-the-art speech recognition with sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4774–4778. IEEE, 2018.

Kyunghyun Cho and Masha Esipova. Can neural machine translation do simultaneous translation? volume abs/1606.02012, 2016. URL http://arxiv.org/abs/1606.02012.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, and Stephan Vogel. Incremental decoding and training methods for simultaneous translation in neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 493–499, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2079. URL https://www.aclweb.org/anthology/N18-2079.

CARL Elliott. The perfect voice. *C. Elllott, Better than well: American medicine meets the American dream*, pages 1–27, 2003.

T Fujita, Graham Neubig, Sakriani Sakti, T Toda, and S Nakamura. Simple, lexicalized choice of translation timing for simultaneous speech translation. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2013.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proc. of ICML*, 2017.

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP)*, pages 1342–1352, 2014.

Alvin Grissom II, Naho Orita, and Jordan Boyd-Graber. Incremental prediction of sentence-final verbs: Humans versus machines. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, 2016.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O. K. Li. Learning to translate in real-time with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 1053–1062, 2017. URL https://aclanthology.info/papers/E17-1099/e17-1099.

He He, Alvin Grissom II, Jordan Boyd-Graber, and Hal Daumé III. Syntax-based rewriting for simultaneous machine translation. In *Empirical Methods in Natural Language Processing*, 2015.

He He, Jordan Boyd-Graber, and Hal Daumé III. Interpretese vs. translationese: The uniqueness of human strategies in simultaneous interpretation. In *North American Association for Computational Linguistics*, 2016.

Liang Huang, Kai Zhao, and Mingbo Ma. When to finish? optimal beam search for neural text generation (modulo beam size). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2134–2139, 2017.

Navdeep Jaitly, David Sussillo, Quoc V Le, Oriol Vinyals, Ilya Sutskever, and Samy Bengio. An online sequence-to-sequence model using partial conditioning. In *Advances in Neural Information Processing Systems*, pages 5067–5075, 2016.

Sungwon Kim, Sang-Gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. FloWaveNet: A generative flow for raw audio. In *International Conference on Machine Learning*, pages 3370–3378, 2019.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*, 2017.

Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with Transformer network. In *AAAI*, 2019.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. STACL: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, Florence, Italy, July 2019a. Association for Computational Linguistics. doi: 10.18653/v1/P19-1289. URL `https://www.aclweb.org/anthology/P19-1289`.

Mingbo Ma, Baigong Zheng, Kaibo Liu, Renjie Zheng, Hairong Liu, Kainan Peng, Kenneth Church, and Liang Huang. Incremental text-to-speech synthesis with prefix-to-prefix framework. *arXiv preprint arXiv:1911.02750*, 2019b.

Mingbo Ma, Renjie Zheng, and Liang Huang. Learning to stop in structured prediction for neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1884–1889, 2019c.

Mingbo Ma, Liang Huang, Hao Xiong, Kaibo Liu, Chuanqiang Zhang, Renjie Zheng, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, Haifeng Wang, and Baigong Zheng. Systems and methods for simultaneous translation with integrated anticipation and controllable latency (stacl), April 2 2020. US Patent App. 16/409,503.

S Matsubarayx, K Iwashimaz, N Kawaguchizx, K Toyama, and Yasuyoshi Inagaki. Simultaneous japanese-english interpretation based on early prediction of english verb. 2000.

Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. Montreal Forced Aligner: Trainable text-speech alignment using Kaldi. In *Interspeech*, 2017.

Barbara Moser-Mercer, Alexander Künzli, and Marina Korac. Prolonged turns in interpreting: Effects on quality, physiological and psychological stress (pilot study). *Interpreting*, 3(1):47–64, 1998.

Kenton Murray and David Chiang. Correcting length bias in neural machine translation. In *Proceedings of WMT 2018*, 2018.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Optimizing segmentation strategies for simultaneous speech translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2014.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Syntax-based simultaneous translation through prediction of unseen syntactic constituents. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 198–207, 2015.

Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, and Demis Hassabis. Parallel WaveNet: Fast high-fidelity speech synthesis. In *ICML*, 2017.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *Arxiv*, 2016.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*, pages 311–318, Philadephia, USA, July 2002.

Kainan Peng, Wei Ping, Zhao Song, and Kexin Zhao. Parallel neural text-to-speech. *arXiv preprint arXiv:1905.08459*, 2019.

Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan Ömer Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John L. Miller. Deep Voice 3: Scaling text-to-speech with convolutional sequence learning. In *ICLR*, 2017.

Wei Ping, Kainan Peng, and Jitong Chen. ClariNet: Parallel wave generation in end-to-end text-to-speech. 2018.

R. Prenger, R. Valle, and B. Catanzaro. WaveGlow: A flow-based generative network for speech synthesis. 2018.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *ICLR*, 2016.

Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. FastSpeech: Fast, robust and controllable text to speech. *arXiv preprint arXiv:1905.09263*, 2019.

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016. doi: 10.18653/v1/P16-1162. URL `https://www.aclweb.org/anthology/P16-1162`.

Jonathan Shen, Ruoming Pang, Ron Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural TTS synthesis by conditioning WaveNet on MEL spectrogram predictions. In *Interspeech*, 2018. tacotron 2.

Gabriel Skantze and Anna Hjalmarsson. Towards incremental speech generation in dialogue systems. In *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 1–8. Association for Computational Linguistics, 2010.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Hideyuki Tachibana, Katsuya Uenoyama, and Shunsuke Aihara. Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4784–4788. IEEE, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, 2017.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. In *Interspeech*, 2017. URL `https://arxiv.org/abs/1703.10135`.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, volume 14, pages 77–81, 2015.

Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the beam search curse: A study of (re-) scoring methods and stopping criteria for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Mahsa Yarmohammadi, Vivek Kumar Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. Incremental segmentation and decoding strategies for simultaneous translation. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, 2013.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. Simultaneous translation with flexible policy via restricted imitation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5816–5822, 2019a.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. Simpler and faster learning of adaptive policies for simultaneous translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*, 2019b.

Baigong Zheng, Kaibo Liu, Renjie Zheng, Mingbo Ma, Hairong Liu, and Liang Huang. Simultaneous translation policies: From fixed to adaptive. *arXiv preprint arXiv:2004.13169*, 2020a.

Renjie Zheng, Junkun Chen, and Xipeng Qiu. Same representation, different attentions: Shareable sentence representation learning from multiple tasks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4616–4622. ijcai.org, 2018a. doi: 10.24963/ijcai.2018/642. URL https://doi.org/10.24963/ijcai.2018/642.

Renjie Zheng, Mingbo Ma, and Liang Huang. Multi-reference training with pseudo-references for neural translation and text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3188–3197. Association for Computational Linguistics, 2018b. doi: 10.18653/v1/d18-1357. URL https://doi.org/10.18653/v1/d18-1357.

Renjie Zheng, Yilin Yang, Mingbo Ma, and Liang Huang. Ensemble sequence level training for multimodal MT: osu-baidu WMT18 multimodal machine translation system report. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 632–636. Association for Computational Linguistics, 2018c. doi: 10.18653/v1/w18-6443. URL https://doi.org/10.18653/v1/w18-6443.

Renjie Zheng, Hairong Liu, Mingbo Ma, Baigong Zheng, and Liang Huang. Robust machine translation with domain sensitive pseudo-sources: Baidu-osu WMT19 MT robustness shared task system report. In *Proceedings of the Fourth Conference on Machine Translation, WMT 2019, Florence, Italy, August 1-2, 2019 - Volume 2: Shared Task Papers, Day 1*, pages 559–564. Association for Computational Linguistics, 2019c. doi: 10.18653/v1/w19-5367. URL https://doi.org/10.18653/v1/w19-5367.

Renjie Zheng, Mingbo Ma, Baigong Zheng, and Liang Huang. Speculative beam search for simultaneous translation. In *Proceedings of the 2019 Conference on Empirical*

Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, 2019d.

Renjie Zheng, Mingbo Ma, Baigong Zheng, Kaibo Liu, and Liang Huang. Opportunistic decoding with timely correction for simultaneous translation. *arXiv preprint arXiv:2005.00675*, 2020b.

APPENDICES