Transforming Neural Machine Translation Into Simultaneous Text and Speech Translation

by

Ashkan Alinejad

M.Sc., University of Tehran, 2016
B.Sc., Shahid Beheshti University, 2014

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2021

Declaration of Committee

Name:	**Ashkan Alinejad**
Degree:	**Doctor of Philosophy**
Thesis title:	**Transforming Neural Machine Translation Into Simultaneous Text and Speech Translation**
Committee:	**Chair:** Angel Xuan Chang Assistant Professor, Computing Science

Anoop Sarkar
Supervisor
Professor, Computing Science

Fred Popowich
Committee Member
Professor, Computing Science

Greg Mori
Examiner
Professor, Computing Science

Colin A. Cherry
External Examiner
Adjunct Professor, University of Alberta
Research Scientist, Google Research

Abstract

Simultaneous neural Machine Translation (SiMT) aims to maintain translation quality while minimizing the delay between reading the input and incrementally producing the output. The eventual goal of SiMT is to match the performance of highly skilled human interpreters who can simultaneously listen to a speaker in a source language and produce a translation in the target language with minimal delay. In this thesis, we explore attempts at building reliable simultaneous translation systems that can produce fluent translations with minimal latency.

We present two distinct methods for finding an optimal policy that tells us if current input is enough for generating accurate translations, or we need to wait for more information. Our first method employs a prediction mechanism to inform the model about incoming input stream. We show as the length of sentences grows, predicting future time steps become essential due to more complex re-orderings that can happen more often in long sentence pairs. Our second method introduces a new algorithmic approach for finding optimal policy as a reference in a supervised learning model. The resulting system translates more accurately with less delay. Our third project focuses on improving the performance of an end-to-end speech translation system, which many simultaneous speech systems rely on. We propose a new loss function that allows us to use available huge datasets for Machine Translation task in order to improve the performance of speech translation system.

Keywords: Simultaneous Machine Translation; Machine Translation; Speech Translation; Real-time, Simultaneous

Acknowledgements

I would like to thank my supervisor, Anoop Sarkar, for his generous support and guidance in every step of my PhD. His knowledgeable insights and visions taught me a great deal of research skills. I would also thank the members of my committee, Fred Popowich, Greg Mori, Colin A. Cherry and Angel Xuan Chang for their feedback.

Collaborating with faculty and students at NatLang lab was a pleasure. I had many fruitful discussions with Maryam Siahbani, Hassan Shavarani, Anahita Mansouri, Golnar Sheikhshab, Logan Born, Nishant Kambhatla, Jetic Gu, Nadia Ghobadi, Pooya Moradi and Vincent Huang.

I would like to express my gratitude to my family and friends who made my life full of joy. Especially my parents Keivan and Roodabeh, my brother Arash, my sister Ana and her husband Ali whose love and kindness are always with me. Finally, this thesis would not have been possible without the patience, love and support of my partner, Kiana.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations

Simultaneous machine translation removes a central assumption made by machine translation systems: The translation in the target language is produced only after the input source language utterance or sentence is fully received. This type of translation is particularly suited for speech to speech translation (also called conference interpretation) where waiting for the end of the sentence creates a long delay and creates an unnatural interaction between the speaker and the hearer. Most contemporary speech-to-speech translation systems (such as Skype Translator) use pauses in the speaker output to produce the output translation. However, a fully trainable simultaneous translation system can be more adaptive, have less latency and produce more fluent translations. The application of such systems is as diverse as translating classroom lectures [36], travel assistance [94], and subtitling movies and digital media [64]. There are many challenges in building a simultaneous translation model: the divergent syntax of different languages, knowing when to wait and when to translate (Also known as Policy) is tricky, and sometimes the translation system might need to predict what the speaker might say or paraphrase what the speaker has said already.

This thesis focuses on simultaneous translation systems where the generated output token cannot be revised and usually referred to as *streaming translation* [8]. These models are in contrast with re-translation methods, which take advantage of repairing previously translated words [71, 70, 7, 8]. While in both streaming and re-translation settings designing agile and high accuracy policies is challenging, the cost of making a mistake in streaming translation is much higher. This is because waiting conservatively will lead to high latency, and early translation can negatively affect the final translation quality.

Using acoustic speech signals as input to the simultaneous translation system can make the problem more complicated. The available datasets for the speech translation task are often noisy, and the shortage of training data pairs is interrupting the performance of current models on most language directions [13, 14]. We will discuss how simultaneous speech translation systems can be trained more efficiently by using available resources for other tasks.

1.2 Summary of Contributions

The primary contributions of this thesis can be summarized in the following directions:

- **Adding prediction mechanism to Simultaneous Machine Translation**: Deciding when to stop reading and start translating based on already seen words can become more accurate by informing the model about upcoming words in the source stream. We propose a technique to incorporate a prediction mechanism in a trainable Agent, which can lower the delay and improve translation quality compared to our strong baselines.

- **Designing effective supervision for policy generation**: A method for generating optimal segments in the source and target sentences by comparing the output of translation component in the simultaneous system with the translations generated by the offline translation model. Our method finds the shortest segments that do not disturb final translated words by fixing the translation model. We also show that our supervised agent can be trained using previously proposed translation components and generate better policies with lower latency and higher translation quality compared to what has been reported before.

- **Improving the performance of end-to-end Speech translation systems**: A new setting for pretraining speech translation systems by making use of available datasets for the Machine Translation task. We propose a new adversarial loss that can align the latent representations of the ASR and MT encoders, enabling us to effectively use the pretrained parameters of the MT decoder for training our speech translation system.

1.3 Thesis Outline

The rest of the chapters in this dissertation are organized as follows:

Chapter 2 studies the relevant background for simultaneous tasks. We start by looking at the structure of offline machine translation (section 2.1) and speech translation (section 2.2) tasks, which serve as the backbone of most structures in the simultaneous setting. Then we discuss the general structure for transforming an offline model into a structure that can be used in real-time environments in section 2.3. The evaluation metrics for both translation quality and delay are briefly described in section 2.4.

Chapter 3 covers a comprehensive overview of previous attempts to address the simultaneous translation task. The chapter begins with a brief discussion of traditional methods and significant challenges for this task (section 3.1). The chapter continues with recent advances in sections 3.2 and 3.3.

Chapter 4 Introduces a new prediction mechanism for the simultaneous translation task.

We propose a new prediction action for an agent trained with reinforcement learning technique which can inform the model about the incoming input stream.

Chapter 5 presents a new supervised technique in order to have a more stable training with higher accuracy. We demonstrate how the new technique can be applied to various translation components and achieve better translation quality and lower latency with more optimal policies.

Chapter 6 Describes a method to improve the efficiency of the training process in simultaneous speech translation models where the available resources for fully training are not available. Our experimental results (section 6.3) demonstrate by using huge datasets for MT, we can achieve noticeably improvements in the performance over the baseline.

Chapter 7 concludes this thesis by summarizing major contributions described in previous chapters. Then describe some future research direction in section 7.1.

Chapter 2

Background

Traditionally, most machine translation research has focused on the problem of translating entire sentences at once. In the last decade, the development of robust neural network architectures and the availability of large-scale datasets allowed building effective machine translation systems for text and speech that can interpret with high accuracy. The promising performance of offline translation models was crucial for creating simultaneous translation frameworks. Even recently proposed models for SiMT heavily rely on the performance of the underlying offline translation model. This chapter will study an accurate yet straightforward NMT framework (section 2.1) with attention mechanism, followed by a brief review of speech translation frameworks (section 2.2). Then we will discuss the requirements for turning an offline setting into a simultaneous model (section 2.3). We finish this chapter by explaining various metrics for evaluating a simultaneous model (section 2.4).

2.1 Neural Machine Translation

Neural Networks can be seen as an essential component in the most recent approaches in machine translation. This chapter will describe how we can use them to build translation systems that can extract semantic information from the source language and follow the syntactic structure of the target language to produce translated words.
Most of the state-of-the-art approaches toward solving Machine Translation use the Encoder-Decoder architecture with attention; However, since adding attention mechanism makes the structure more complex, we will start describing simple Encoder-Decoder translation systems in the next section. Later, in 2.1, we will demonstrate how attention mechanism affects the translation process.

Encoder-Decoder structure

The very basic neural structure that can generate reasonable translations is what is called the **Encoder-Decoder** model [28, 93, 26]. The concept of this model is to use stacked layers of LSTM or GRU cells to *encode* the whole source sentence into a real-valued vector. Then we can feed this vector to our second neural network to *decode* it and produce translated words one at a time. See Figure 2.1 for illustration.
More mathematically, our network at time step i will receive a numerical representation of word x_i from the source sentence $\mathrm{X} = \{x_1, \ldots, x_I\}$. Then it will combine them with the

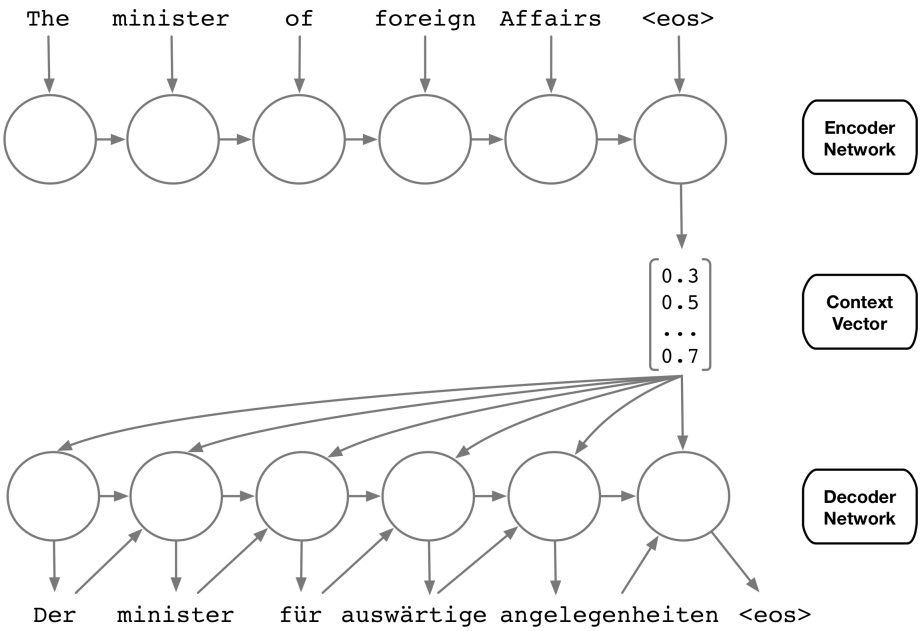


Figure 2.1: Structure of an Encoder-Decoder model.

output of encoder at previous time step and passes them to a non-linear function. In other words: $h_i = \mathcal{F}(x_i, h_{i-1})$, where $\mathcal{F}$ is a non-linear function (e.g. Tanh, Sigmoid, ...). Once the encoder receives the <eos> word, it will compute the final encoder's context vector h_I. In the decoder component, we will use the predicted word from previous time step y_{j-1}, previous decoder output s_{j-1} , and h_I, as input for the decoder's neural network. We will compute s_j, similar to the encoder's context vector, with $s_j = \mathcal{G}(y_{j-1}, s_{j-1}, h_I)$, where $\mathcal{G}$ is a non-linear function. Then we will pass s_j through a softmax layer in order to compute the probability $P(y_j|\mathrm{X}, y_{<j})$. We will choose y_j that maximizes the probability. i.e. $y_j = \arg\max_{y_j} P(y_j|\mathrm{X}, y_{<j})$.

This is the basic, yet powerful structure of the encoder-decoder model, and while there are lots of neural models for machine translation, these can be seen as extensions to this model. In [93] it is shown that with some refinements (E.g. using BiLSTM instead of LSTM as encoder), the encoder-decoder structure can beat most of the approaches with many years of research in their background. We will talk about these techniques in the next two sections.

Bidirectional RNNs

Bidirectional RNNs (BRNN) [87] extend the unidirectional RNN by introducing a second hidden layer with connections that flow in the opposite temporal view. The first hidden layer passes the activations forward, which connects the previous time steps to the current time step; While the second layer has connections in the opposite direction so that the current time step can observe the information from the future. The nodes in both hidden layers are connected to the nodes to previous and next layers. The computations in a BRNN layer can be described by three equations:

$$\overrightarrow{h}_i = \mathcal{F}(x_i, \overrightarrow{h}_{i-1})$$
$$\overleftarrow{h}_i = \mathcal{F}(x_i, \overleftarrow{h}_{i+1})$$
$$y_i = \mathcal{G}(\overrightarrow{h}_i, \overleftarrow{h}_i)$$

Where $\overrightarrow{h}_i$ and $\overleftarrow{h}_i$ are forward and backward hidden states at time step i. One limitation of BRNN is that it cannot be applied to online or real-time applications, since it's not possible to receive information from future. But in batch procedures having access to the whole sequence is a reasonable assumption, and BRNNs can improve results.

Attention Mechanism

We are only one step away to look at a powerful translation model, which is the encoder-decoder model with attention mechanism. The encoder-decoder networks force the encoder to keep all the information required for decoding into a fixed-dimensional context vector. On the other side of this structure, the decoder only has access to this context vector, and it is supposed to produce the whole translation using this fixed representation of the input. With keeping these restrictions in mind, although the architecture works well, as the length of sentences grows, the decoder's performance decreases a lot. In order to fix these constraints, Bahdanau et al. [11] propose the attention mechanism. The main idea is that instead of using the last state of the encoder's context vector, the decoder can use a weighted combination of the encoder's output at different time steps. As a result, Not only would the decoder be powerful, but it would also be much easier for the encoder to encode input sentences at each time step.

More concretely, the first component of this structure encodes the embeddings of input words $\mathrm{X} = \{x_1, \ldots, x_I\}$ into context vectors $\mathrm{H} = \{h_1, \ldots, h_I\}$. It can be done by utilizing a bidirectional RNN:

$$h_i = \mathcal{F}_{\mathrm{BiRNN}}(h_{i-1}, x_i)$$

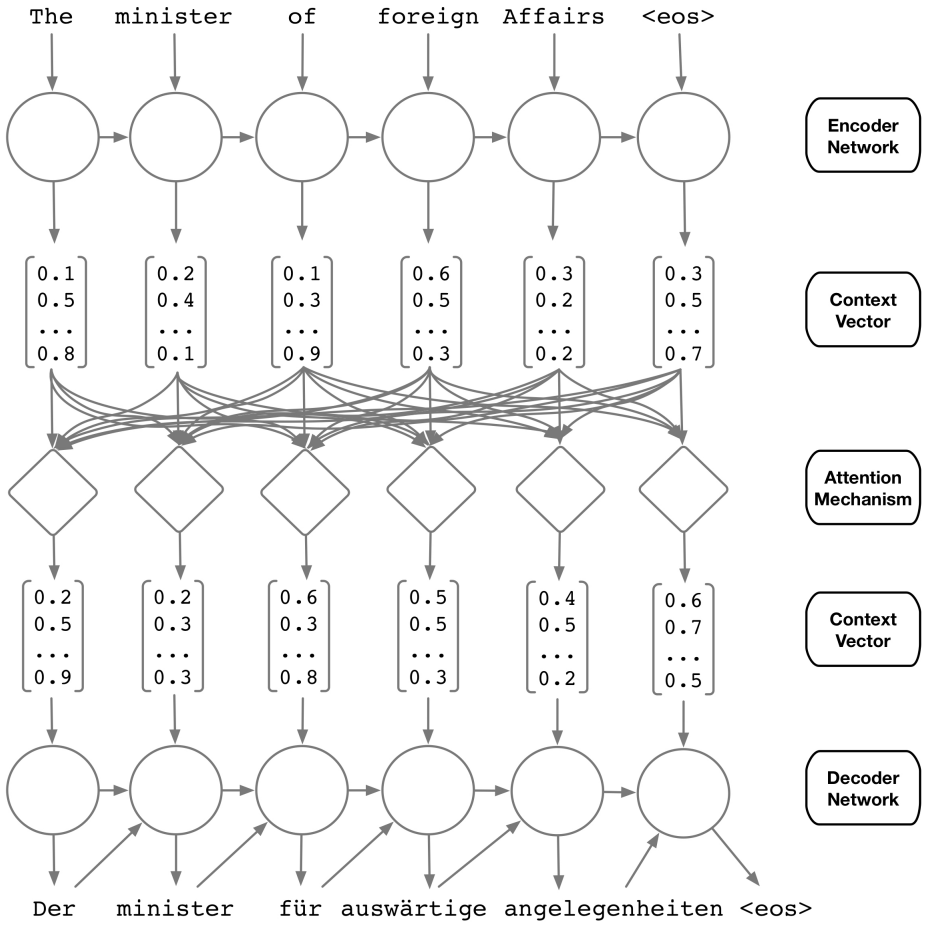


Figure 2.2: Structure of an Encoder-Decoder model with Attention mechanism.

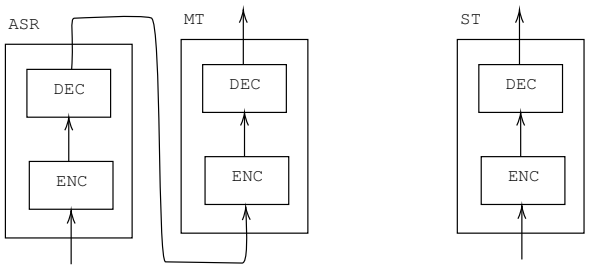
On decoder side we will have:

$$\alpha_i^j = \mathcal{F}_{\text{ATTN}}(s_{j-1}, h_i) \tag{2.1}$$

$$c_j = \sum_{n=1}^{I} \alpha_n^j h_n \tag{2.2}$$

$$s_j = \mathcal{F}_{\text{DEC}}(s_{j-1}, y_{j-1}, c_j) \tag{2.3}$$

$$P(y|y_{<j}, \text{H}) \propto \exp[\mathcal{F}_{\text{OUT}}(s_j)] \tag{2.4}$$



(a) Cascaded model (b) End-to-end model

Figure 2.3: Schematic of overview of cascaded structure and end-to-end model for speech translation.

$$y_j = \arg\max_y P(y|y_{<j}, \mathrm{H}) \tag{2.5}$$

In equation 2.1, s_{j-1} is the decoder's context vector at previous time step and $\mathcal{F}_{\mathrm{ATTN}}$ is the attention function which can be any function that measures how well the input at position i is related to output at time step j. The most commonly used function is the original formula presented by Bahdanau et al [11] which employs a multi-layer feedforward neural network. We then pass these scores to a softmax function in order to make their summation equal to one. The output of attention layer in equation 2.2 is using α_i^j as probability for computing a weighted average over the encoder's context vector.

2.2 Speech Translation

Speech Translation (ST), the task of transforming acoustic signals in the source language into the translated text words in the target language, is a complex combination of Automatic Speech Recognition (ASR) and Machine Translation (MT) tasks. Most of the challenges from ASR and MT, like word reordering, information loss and data efficiency, should also be addressed in ST models. Conventional models for ST [92, 69, 97, 23] are based on the cascaded pipeline of ASR and MT systems, where the ASR component transcribes incoming signals, and the MT component interprets the transcriptions. In recent years, attempts to remove the transcription step and create an end-to-end model have shown promising results [1, 2, 4].

Cascaded Models

As depicted in figure 2.3, the cascaded ST models build ASR and MT components separately and combine them by sending the output of ASR to the input of MT models. Waibel et al. [66] extend this structure by adding Text-to-Speech (TTS) system in order to generate translated speech.

Considering the fact that cascaded ST models are using a pipeline of distinct models trained on two different corpora, these models are dealing with several challenges:

- The most critical challenge in these models is the problem of **error propagation**. The error generated by the ASR systems gets propagated through the rest of models and affects the accuracy of the translated sentence.

- Since the ASR and MT models are trained separately, the output of ASR may not be compatible with the MT component. The generated transcriptions in most ASR models do not contain punctuation marks, which can decrease the accuracy of the MT model. Removing the punctuation from input during training of MT also drops its performance noticeably. Aside from punctuation, the output of the ASR model does not remove disfluencies from speech or lacks the ability to tokenize or normalize generated sentences, which are crucial for generating high-quality translations.

- The transcription process drops lots of critical information contained in speech signals. Many acoustic cues, such as stress, pause, or even tone of the speaker, can be helpful for easier recognition of sentence structure and improving the translation quality.

Improving the robustness of the underlying ASR and MT components can directly impact the overall performance of the ST system. So adding distorted sentences with ASR mistakes to the MT training data [77, 95] can introduce the translation component to some of the probable errors it can face during inference. Another idea is to modify the training process of the ASR to make its output more compatible with the translation component [32, 44].

End-to-End models

Successful application of end-to-end architecture yielded breakthroughs in many research areas and motivated researchers to investigate the effect of end-to-end architecture on the ST task. By removing the transcription step, the number of model parameters decreases, and most of the problems in cascaded models will be addressed.

One of the first end-to-end ST systems proposed by Bérard et al. [17], where an attention-based encoder-decoder model (similar to the structure described in section 2.1) was modified to work with speech signals. They replaced the standard attention mechanism with the convolutional attention proposed by Chorowski et al. [27], which applies convolution filter F to attention weights at previous time steps. Given the encoder representations $\mathrm{H} = (h_1, \ldots, h_I)$ and decoder state s_j, the modified formulation of attention mechanism can be formulated as follows:

$$e_i^j = v^\top \tanh(Wh_i + Vs_j + +Uf_{j,i} + b) \tag{2.6}$$

$$\alpha_i^j = \mathrm{softmax}(e_i^j) \tag{2.7}$$

$$c_j = \sum_{n=1}^{I} \alpha_n^j h_n \tag{2.8}$$

Where W, V, and U are weight matrices, and $f_{j,i}$ is generated by convolving the filter F with every position i of the previous alignment α_{j-1}:

$$f_j = F * \alpha_{j-1}$$

For the choice of input features, Chorowski et al. use 40-dimensional MFCCs + frame energy and pass them through a hierarchical encoder with three layers, which results in a sequence of 1/4th the size of the original input.

The alteration of this architecture has been thoroughly explored in recent years [15, 98, 37]. While these models can benefit from simple structure and have access to acoustic cues to understand the structure of sentences better, the cascaded models still outperform the end-to-end model by a large margin. One major obstacle that is shared between all end-to-end ST structures is the lack of enough training data. The key to success of end-to-end models for ASR and MT is training on huge datasets available for their task. Except for a few language pairs, acquiring a moderate amount of speech-to-text data pair is not easy and result in under-trained models.

2.3 Moving from Offline Models to Simultaneous Structures

We start this section by explaining how we can transform an offline setting to a simultaneous setting in text-to-text models. Then we will briefly discuss how speech-to-text models can be converted.

In order to translate a sentence simultaneously, we are required to have a mechanism to decide if the current amount of input words are enough for accurately translating the next word or we need to wait longer. This decision, which we will refer to as *Policy*, plays a crucial role in streaming translation[1] where we do not modify previously translated words. As depicted in figure 2.4, the main structure of a simultaneous model can be divided into two primary components:

- INTERPRETER also referred to as *Interpreter* or *Translation component*, which receives the stream of input words in the source language and takes care of translation based on the schedule that the agent provides

- AGENT or *programmer* receives information about the current time step from the interpreter and decides if we should receive more input from the source stream (READ) or send the current translation to the output (WRITE). While the agent is a separate component from the interpreter in most frameworks, it might not be easy to draw a clear line to split the model into agent and interpreter components.

[1]The focus of this thesis is on streaming translation, and except for brief comparisons we will not discuss any the techniques introduced for re-translation or their counterparts.

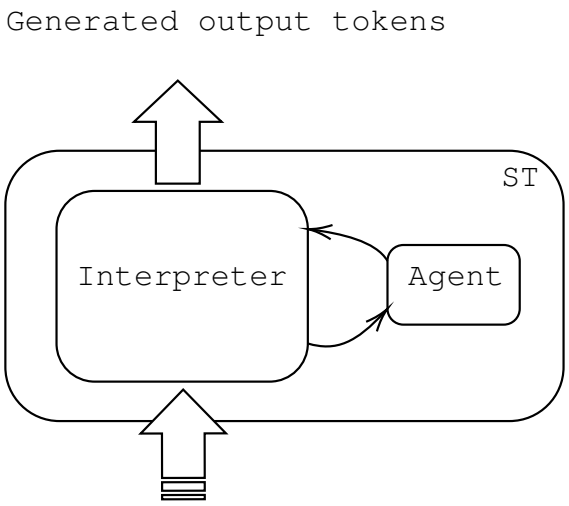


Figure 2.4: Structure of a Simultaneous Neural Machine Translation. The ENVIRONMENT informs the AGENT about current time step and the AGENT decides what should the ENVIRONMENT do for the next time step.

One important thing to note is that the notion of "time step" for the agent is different from the "time step" we usually refer to in offline settings. The formulation for the encoder-decoder structure uses separate indices to keep track of incremental progress in encoders and decoders. In simultaneous environment, if the interpreter uses an encoder-decoder structure, the agent's time step should be a combination of both encoder and decoder time steps to work with both components. In other words, if the interpreter's encoder receives the input words $\mathrm{X} = \{x_1, \ldots, x_I\}$ from the source language and incrementally generates translated words $\mathrm{Y} = \{y_1, \ldots, y_J\}$ in the target language, the number of agent's decisions T can be written as $T = I + J$.

State-of-the-art strategies for modelling the agent will be discussed in sections 3.2 and 3.3. The interpreter in most simultaneous frameworks is modelled similarly to the offline MT models. The main modification in the interpreter is that the translations in decoder are generated based on prefixes of encoder representations.

For simplicity, we will describe the modification required for an attention-based encoder-decoder environment that uses an LSTM network. More complicated structures for the interpreter can be adjusted similarly and will be discussed in chapter 3.

Suppose the Encoder receives the embedded representation of input sentences $\mathrm{X} = \{x_1, \ldots, x_I\}$ and converts them into context vectors $\mathrm{H} = \{h_1, \ldots, h_K\}$ such that:

$$h_i = \mathcal{F}_{\mathrm{ENC}}(x_i, h_{i-1})$$

Since we do not have access to the whole sentence after i READs ($i < I$), we will use H^i which contains those context vectors that have been read so far (i.e. $\mathrm{H}^i = \{h_1, \ldots, h_i\}$). The Decoder is trained to generate target word, given the context vectors H^i, previous predicted word y_{j-1} and previous decoder state s_{j-1} after i READs and j WRITEs. for $t = i + j$ we will have:

$$
\begin{aligned}
c_t &= \mathcal{A}_{\mathrm{ATT}}(s_{j-1}, \mathrm{H}^i) \\
z_t &= \mathcal{F}_{\mathrm{DEC}}(s_{j-1}, y_{j-1}, c_t) \\
P(\hat{y}_t | y_{<j}, \mathrm{H}^i) &= \mathcal{G}(y_{j-1}, s_j, c_t)
\end{aligned}
$$

Where $\mathcal{A}$, $\mathcal{F}$ and $\mathcal{G}$ are nonlinear functions. The word with highest probability will be selected as output at each time step:

$$
\hat{y}_t = \arg\max_{\hat{y}} P(\hat{y} | y_{<j}, \mathrm{H}^i)
$$

One frequent point of confusion in the new translation model is that, in contrast to the standard NMT model, we can have an output at each time t; However, the agent decides whether to ignore the current candidate ($\hat{y}_t$) and wait for better predictions [READ] or accept the candidate ($y_j \leftarrow \hat{y}_t$, $s_j \leftarrow z_t$) and select it as the next translated word [WRITE].

Transforming speech translation models

Building Simultaneous Speech Translation (SiST) is a more challenging task compared to SiMT as the length of the source speech signals are much longer than the translated text, and lots of optimization techniques such as BPE or tokenization can not be applied to the input audio signals. Conventional approaches for SiST modify the cascaded architecture (section 2.2) to generate translations simultaneously [36, 72].

Similar to the cascaded model in the offline setting, an ASR model with streaming architecture [48, 85, 39, 82] allows transcriptions to be generated as soon as the first input is encoded. Each generated transcription will be forwarded to a SiMT model to produce translated target words. Fügen et al. [36] and Bangalore et al. [12] make use of pauses in the middle of speech signals to segment the incoming input. While pauses are a good indicator of a phrase boundary, this method heavily depends on the speech attributes of each speaker. Rangarajan et al. [91] predict the potential positions for commas or periods in the generated transcription and use them as sentence boundaries. Oda et al. [72] introduce an algorithm based on greedy search and dynamic programming to find the best segmentation for the transcriptions before being sent to the machine translation component. In all these cascaded models, the ASR and MT components segment their inputs completely separate from each other. Consequently, their policies cannot be optimized jointly, which results in low accuracies. Recently several end-to-end architectures were proposed for SiMT, and we study them in chapter 3.

2.4 Evaluation Metrics

In the standard NMT framework, translation quality is the primary metric for evaluating various translation systems. However, simultaneous translation requires balancing the trade-off between translation quality and time delay to ensure that users receive translated content in an expeditious manner [67]. Therefore, in this section we will describe various evaluation metrics for both quality and delay in MT systems.

2.4.1 Quality

Although human evaluations for the quality of machine translation (MT) weigh many aspects of translation, including adequacy, fidelity, and fluency [46], they are expensive and time-consuming. As a result, automatic evaluation techniques such as the *BLEU* score [75] have been widely used in MT systems. Here, we will discuss the most commonly used evaluation metrics for quality:

- **Standard BLEU**
  In recent years BLEU became the de facto standard machine translation (MT) evaluation metric [20]. It is based on the degree of n-gram overlapping between the strings of words produced by the machine and the human translation references at the corpus level. BLEU is usually computed for n-grams of size 1 to 4 with the coefficient of brevity penalty (BP).

$$\text{BLEU} = \text{BP} \times \left(\prod_{n=1}^{N} \frac{m_n}{l_n} \right)^{\frac{1}{N}}$$

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{if } c \leq r \end{cases}$$

  Where m_n is the number of matched n-grams between translation and its reference, and l_n is the total number of n-grams in the translation. c is the total length of candidate translation corpus, and r refers to the sum of effective reference sentence length in the corpus.

- **Smooth BLEU**
  One of the main criticisms of BLEU is that since it computes a geometric mean of n-gram precisions, if a higher-order n-gram precision (e.g. n = 4) of a sentence is 0, then the BLEU score of the entire sentence is 0, no matter how many 1-grams or 2-grams are matched. Among several smoothing techniques proposed to address this problem, the smoothing technique suggested by Lin et al. [57] is the most widely used.

It adds 1 to the matched n-gram count and the total n-gram count for n ranging from 2 to N.

$$m'_n = m_n + 1 \qquad \text{for } n \text{ in } 2 \dots N$$

$$l'_n = l_n + 1 \qquad \text{for } n \text{ in } 2 \dots N$$

2.4.2 Delay

Another critical feature in real-time machine translation systems is delay which means how much time is wasted for reading when translating each word. there is no unique formulation for delay, and various methods use their own definition to report rapidity of their systems. The more common and accurate choices in practice for delay function is *Average Proportion* and *Consecutive Wait* which we will describe:

- **Average Proportion (AP)**
  Cho et al. [24] defined AP as an average number of source words needed when translating each word. In other words, AP can be computed as:

  $$\text{AP} = \frac{1}{I \times J} \sum_j \psi(j)$$

  Where I, J are the lengths of source and decoded sequences respectively, and $\psi(j)$ denote the number of source words been waited when decoding each word.

  One major drawback of AP is that the length of the source and target sentences affects the final delay. In other words, the shorter the sentences, the higher the AP would be.

- **Consecutive Wait (CW)**
  CW as is defined by Gu et al. [41] is how many words were waited for (READ) consecutively between translating two words. It is defined at each time step is equal to the length of the current segment. CW can be computed at each time step as:

  $$\text{CW}_t = \begin{cases} \text{CW}_{t-1} + 1 & a_t = \text{READ} \\ 0 & a_t = \text{WRITE} \end{cases}$$

  The drawback of CW is that CW is local latency measurement which is insensitive to the actual lagging behind [59].

- **Average Lagging (AL)**
  To address the problem of CW and AP, Ma et al. [59] proposed a new metric called Average Lagging. The delay in AL is computed based on the number of words the system lag behind an ideal translator, and can be defined as:

  $$\text{AL} = \frac{1}{\tau} \sum_{n=1}^{\tau} g(n) - \frac{(n-1)}{r}$$

where $r = {}^{J}/_{I}$ is the ratio of the target to source length, $g(n)$ is the delay at time n, and τ is the cut-off time step when the model received the whole source words:

$$\tau = \arg\min_{n}[g(n) = I]$$

The τ which contains the arg min operation makes AL not differentiable and cannot be optimized

- **differentiable Average Lagging (DAL)**
  Cherry et al. [21] proposes the differentiable version of AL which is defined as:

$$\text{DAL} = \frac{1}{J}\sum_{n=1}^{J} g'(n) - \frac{(n-1)}{r}$$

Similar to AL, r is defined as the ratio of the target to source length, and g' is defined as:

$$g'(n) = \max \begin{cases} g(n) \\ g'(n-1) + \frac{1}{r} \end{cases}$$

By eliminating the arg min element, DAL can be used as a loss component for training neural networks.

2.4.3 Additional Criteria

In speech translation task, the length of the input stream is much longer than the length of the target sentence. Consequently the AL become extremely low and trivial. To address this problem, two modification to the AL have been proposed:

- The modified version of AL proposed by Ren et al. [83] replaces the length of source *sentence* with the number of source *segments*. The new AL is formulated as:

$$\text{AL} = \frac{1}{\tau(|x|^{\text{seg}})}\sum_{n=1}^{\tau(|x|^{\text{seg}})} g(n) - \frac{(n-1)}{r}$$

Where $|x|^{\text{seg}}$ is the number of speech segments, $g(n)$ is the number of source segments received before writing the n-th target token, and $\tau(|x|^{\text{seg}})$ is the smallest n where all the input sequence has received:

$$\tau(|x|^{\text{seg}}) = \arg\min_{n}(g(n) = |x|^{\text{seg}})$$

and $r = {}^{|y|}/_{|x|^{\text{seg}}}$. While this modification adjust the range of AL for speech translation task, it relies on the segmentation strategy and pre-processings.

- Ma et al. [60] proposed a similar conversion for the AL, but does not depend on the segmentation method. The modified AL is defined as:

$$\text{AL} = \frac{1}{\tau(|x|)} \sum_{n=1}^{\tau(|x|)} g(n) - \frac{|x|}{|y|}.T_s.(n-1)$$

  Where $|y|$ is the length of reference translation, $|x|$ is the length of source input stream, T_s is length of speech frames in milliseconds, $g(n)$ is the speech duration elapsed at time step n, and $\tau(|x|)$ is the index of target token when the whole input is received.

2.5 Summary

In this chapter, we have discussed an RNN-based structure for machine translation and how various mechanisms like bi-directionality or attention can improve performance. Then we studied both cascaded and end-to-end structures for speech translation and explored the main differences between an offline and a simultaneous model. We have also explained the most commonly used terms in SiMT and closed this chapter by discussing evaluation metrics for translation quality and the overall delay of a simultaneous translation model. In the next chapter, we will go through the most notable strategies to find the optimal policy.

Chapter 3

Simultaneous Machine Translation

The field of Simultaneous Machine Translation (SiMT) is growing very quickly. In this chapter, we try to summarize the most relevant works published in this area. Based on the type of information each agent uses, we have grouped different methods of finding sentence boundaries into two categories: **Static** policies where the agent applies a specific rule to all sentences, and **Adaptive** policies where the agent seeks to find the best segments using incoming sentences.

We start this chapter by reviewing traditional systems and challenges we are dealing with in this field. In section 3.2, we discuss proposed models with static policies and how their translation components are being trained to build a powerful model for translating partial sentences. Finally, in section 3.3, we move on to adaptive policies, where we categorize different approaches based on the learning mechanism they employ to train the agent.

3.1 Traditional Systems

Before starting the new wave of modern neural architectures, a number of statistical methods have been proposed to address the problem of simultaneous translation, mostly in the context of speech translation [12, 36]. In this section, we will briefly discuss four major threads and challenges from statistical approaches. Some of these challenges remained unsolved in modern architectures.

Segmentation

In real-time translation, interpreter must split a constant stream of words into translatable segments, in a way that maximizes the translation quality and minimizes segment length. The statistical approaches for splitting sentences can be divided into two main groups: *sentence segmentation* and *incremental decoding*. In incremental decoding, incoming words are fed into the decoder one-by-one, and the decoder updates its internal state. The decoder is responsible to decide when to begin the translation process and when to output the translation. In sentence segmentation, the focus is on splitting sentences and as soon as a segment is recognized, it is given to a decoder to generate and output the translation for that segment.

Prediction

In real-time translation we do not have access to future time steps. Consequently when translating from subject-object-verb (SOV) languages like German to subject-verb-object (SVO) languages like English, since the main verb may appear later in an SOV language, for the translation to be truly incremental, the verb, copula, or other sentence-final components must be predicted and translated before they are actualized in the source sentence.

Konieczny et al. [53] predict verbs with a recurrent neural network, but Matsubara et al. [63] was the first to use verb predictions as part of a simultaneous interpretation system. They use pattern matching-based predictions of English verbs. In contrast, Grissom II et al. [40] use a statistical approach, using n-gram models to predict German verbs and particles.

Rewording

Prediction task is inherently hard, and most of the time it's inaccurate. As another solution for SOV-SVO language pair translation, one can apply syntactic transformations to make the word order of one language closer to the other. In other words, rewording is changing the standard way of wording output to prevent long delays. He et al. [43] proposed to rewrite the reference translation in a way that uses the original lexicon, obeys standard grammar rules of the target language, preserves the original semantics. They then train the MT system with the rewritten references so that it learns how to produce low-latency translations from the data.

Evaluation

Another important question for simultaneous translation systems is that how we should evaluate our translator. It's not trivial what is the best method for rewarding the system according to its accuracy and rapidity. Mieno et al. [67] devised an evaluation measure for simultaneous speech translation that simultaneously considers delay and accuracy and found out that considering both speed and translation accuracy in the evaluation of simultaneous speech translation systems results in more effective evaluation.

3.2 Models with Static Policies

The static policies segment source and target sentences via predefined sentence boundaries, without taking into account sentence structure or word re-orderings required for translation. Despite their basic structures, Static policies achieve good results in balancing the trade-off between translation quality and delay, and formed a strong baseline against many adaptive policies with highly more complex structures. This section will review two static policies: The Static Read and Write proposed by Dalvi et al. [29], and wait-k proposed by Ma et al. [59].

3.2.1 Static Read and Write

Dalvi et al. [29] addresses the problem of segmentation by using a fixed policy and modifying the decoder in NMT. The static Read and Write strategy starts with S number of READs, followed by consecutive READs and WRITEs until the end of the source sentence. The parameter S can be chosen by the user and should be large enough to ensure some future context to be used by the decoder.

Incremental Training

Training the MT model on full-sentences and decoding by partial hypotheses creates discrepancy between training and inference which can damage the performance of the trained model. Two methods proposed to modify the training process and match it with the Static Read and Write decoding scenario:

- **Chunk Training** In chunk training a fully trained NMT model is fine-tuned via fixed-length segments. In order to so, the source sentences will be divided into segments with N words. Then the corresponding translated words can be selected from target sentence using statistical word alignment techniques[1]
- **Add-M Training** To better mimic the decoding process during inference, add-M training starts with N initial READs at the beginning and WRITEing the target words that are aligned to those words. For the next time steps we will READ chunks of M words from source and WRITE their aligned words to the target until the source sentence finishes.

3.2.2 Wait-k

Very similar to the Static Read and Write, Ma et al. [59] proposed Wait-k policy which READs k words at the beginning and starts to consecutively WRITE and READ until the `</s>` is generated on the target side. During training the translation model receives the words from the source language and generates words to the target side in exactly the same order as wait-k suggests. The special cases of the wait-k model are:

- **wait-∞** or offline translation, which waits for the whole source sentence to be received before generating any translation.
- **wait-0** is the policy that generates consecutive sequences of WRITE-READs. This model is translate the first word without receiving it and is considered as an ideal policy. This policy is the basis of formulation for Average Lagging, which is explained in section 2.4.2.

[1]The original paper uses fast-align [34].

Algorithm 1 Simultaneous Greedy Decoding (SGD)

Require: Input buffer X, Output buffer Y.

Init $j \leftarrow 0$, $i \leftarrow 1$
while True **do**
  $t \leftarrow j + i$
  $c_t = \mathcal{A}_{\text{ATT}}(s_{j-1}, \text{H}^i)$
  $\hat{y}_t, z_t \leftarrow \mathcal{F}_{\text{DEC}}(s_{j-1}, y_{j-1}, c_t)$
  if $\Lambda(t) = \text{READ}$ and $i < I$ **then**
    $h_{i+1} \leftarrow \mathcal{F}_{\text{ENC}}(h_i, x_{i+1})$
    $\text{H}^{i+1} \leftarrow \text{H}^i \cup \{h_{i+1}\}$, $i \leftarrow i + 1$
    if $|\hat{Y}| = 0$ **then**
      $s_0 \leftarrow \mathcal{F}_{\text{INIT}}(\text{H}^i)$
  else if $\Lambda(t) = \text{WRITE}$ or $i \geq I$ **then**
    $s_j \leftarrow z_t$, $y_j \leftarrow \hat{y}_t$
    $\text{Y} \Leftarrow y_j$, $j \leftarrow j + 1$
  if $\hat{y}_t = \langle \text{eos} \rangle$ **then**
    break

Eval-wait-k

Decoding a full-sentence MT model using Wait-k policy is called Eval-wait-k [59]. The discrepancy between training and inference time makes this model perform worse than the Wait-k model. But as the k grows both models gradually perform as good as full sentence MT.

3.3 Models with Adaptive Policies

Adaptive policies attempt to find optimal sentence boundaries by attending to the structure of source and target sentences, and generating action sequences that are tailored for each sentence pair. We have categorized previously proposed adaptive policies into 5 different groups based on different techniques they employ to address the problem of segmentation. In each part, we will describe how we can frame the the simultaneous translation problem using the technique introduced in that section. We will discuss various expansions to explore that technique more in depth afterwards.

3.3.1 Rule-based Models

The rule-based models are similar to static policies in the sense that their agents are not being trained, and the action sequence is generated based on some predefined rules. However, unlike static models, rule-based techniques pay attention to the word order in the source and translated sentences and make final decisions based on the output of translation component at each time step. The non-trainable agent proposed by Cho et al. [24], controls the translation environment by heuristic modification of decoding process. The algorithm is depicted in Algorithm 1.

The translation process is as follows: At each time step t, if the full source sentence is already received then the most likely word $\hat{y}_t$ will be printed to the output; Otherwise, more source words will be passed through the network. The Agent compares the log probability of the most likely word at current time step with previous time step based on a predefined criterion Λ. If the agent decides to WRITE, then the environment commits the most likely target symbols $\hat{y}_t$ given the current context set H to the output; And if not, the environment will wait for more source words. Two waiting criteria Λ has been studied in [25]:

- **Wait-If-Worse** The first scenario for the agent is to wait for more source words if the log probability of the most likely prediction decreases with more inputs. In other words, the criteria at time t can be defined as:

$$\Lambda(t) = \begin{cases} \text{READ} & \text{if } \log P(\hat{y}_t | y_{<j}, \mathrm{H}^i) < \log P(\hat{y}_t | y_{<j}, \mathrm{H}^{i+1}) \\ \text{WRITE} & \text{otherwise} \end{cases}$$

  where i and j are the number of READs and WRITEs respectively at time t, and $\hat{y}_t = \arg\max_{\hat{y}} P(\hat{y} | y_{<j}, \mathrm{H}^i)$

- **Wait-If-Diff** The other scenario is to print the predicted word $\hat{y}_t$ to the output, only if it won't change by reading more words from source sentence. Mathematically speaking, $\Lambda(t)$ can be described as:

$$\Lambda(t) = \begin{cases} \text{READ} & \text{if } \hat{y}_t \neq \hat{y}_{t+1} \\ \text{WRITE} & \text{otherwise} \end{cases}$$

  where $\hat{y}_{t+1} = \arg\max_{\hat{y}} P(\hat{y} | y_{<j}, \mathrm{H}^{i+1})$. This is different from previous criteria since decreasing log probability of a predicted word doesn't necessarily indicate that most likely word will be changed.

By changing the policy, one can get various modes of translation. The standard NMT system (or wait-∞) can be achieved by setting policy to wait until the end of sentence which we call it **Wait-Until-End**. Also the fastest translator is the one which translates after receiving each word and we call it **Wait-One-Step**.

3.3.2 Reinforcement Learning Techniques

The first trainable agent, introduced by Satija et al. [86] interacts with the environment (which in our case it is NMT system) using Reinforcement Learning algorithms. At each time step t, the agent receives an observation o_t from the environment and chooses an action $a_t \in \{\text{READ}, \text{WRITE}\}$ which leads to a reward r_t and transition into next observation o_{t+1}. The Q-learning techniques have been applied in order to estimate the value of executing an action from a given state, which are referred as Q-values. Then a deep neural network is

employed to approximate the Q function, parameterized by weights denoted by θ [68]. At time t, for a given observation o_t, the Q-values for all the available actions are predicted using this network and are denoted by $Q(a_t, o_t|\theta)$. The Q-values can be learned by making updates to the network to minimize the differentiable loss function:

$$\mathcal{L}(a, o|\theta_i) \approx (r + \gamma \max_{a'} Q(o', a'|\theta_i) - Q(o, a|\theta_i))^2$$

Where the updates are $Q_{i+1} = \theta_i + \alpha \nabla_\theta \mathcal{L}(\theta_i)$. The other settings of their RL system is as follows:

- **Observations** The observation of the agent represents its current view of the environment. It is represented as concatenation of the current Encoder's context vector h_i, The current Decoder's context vector z_t, and the last word predicted by environment y_j. Hence observation at time step t can be written as $o_t = [\ h_i;\ z_t;\ y_j]$.
- **Reward** At each time t, the reward is computed as $r_t = r_t^Q \times r_t^D$ where r_t^Q is the partial BLEU score for reference Y^* which can be written as:

$$r_t^Q = \begin{cases} \frac{1}{\beta}\mathrm{BLEU}(\mathrm{Y}^t, \mathrm{Y}^*) & t < T \\ \mathrm{BLEU}(\mathrm{Y}, \mathrm{Y}^*) & t = T \end{cases}$$

  Where Y is the output, Y^t is the cumulative output at time t and β is the maximum sentence length permissible. r_t^D is delay component of reward formula which can be calculated as:

$$r_t^D = 1 - \frac{l-1}{\lambda}$$

  Where λ is a fixed constant and l represents the length of consecutive READ steps at time t.

We will now describe the translation process after pre-training of the NMT environment has been completed. Given a sentence, the Agent receives observation from the NMT system o_t and generate the Q-values for the available actions using its own neural network. The agent then selects the action with the largest return and executes that. It gets a corresponding reward from the environment and also the next word from the source sequence, which sends it to the observation o_t. It keeps on executing in such manner until it receives terminal observation (the end of source sentence, represented by $\langle$eos$\rangle$ token) and after that the agent performs WRITE actions until the environment predict the $\langle$eos$\rangle$ token for the target sentence.

Trainable Agent with Policy Gradient

The most recent alternative for agent is presented by Gu et al. [41]. Their model is similar to Satija's agent in a sense that they are using the same set of actions and the agent

is trained with reinforcement learning techniques. However, as illustrated in Algorithm 2, many aspects of the agent has been changed:

Algorithm 2 Trainable agent with policy gradient

Require: Policy π_θ, Input buffer X, Output buffer Y.

Init $j \leftarrow 0$, $i \leftarrow 1$
while TRUE **do**
  $t \leftarrow j + i$
  $\hat{y}_t, z_t, o_t \leftarrow \mathcal{F}(s_{j-1}, y_{j-1}, c_t)$
  $a_t \sim \pi_\theta(a_t; a_{<t}, o_{<t})$
  if $a_t =$ READ and $x_i \neq \langle \text{eos} \rangle$ **then**
    $h_{i+1} \leftarrow \mathcal{F}_{\text{ENC}}(h_i, x_{i+1})$
    $\text{H}^{i+1} \leftarrow \text{H}^i \cup \{h_{i+1}\}$, $i \leftarrow i + 1$
    if $|\hat{\text{Y}}| = 0$ **then**
      $s_0 \leftarrow \mathcal{F}_{\text{INIT}}(H^i)$
  else if $a_t =$ WRITE **then**
    $s_j \leftarrow z_t$, $y_j \leftarrow \hat{y}_t$
    $\text{Y} \Leftarrow y_j$, $j \leftarrow j + 1$
  if $\hat{y}_t = \langle \text{eos} \rangle$ **then**
    break

1. **Observation** The current state in Cho's agent represented by concatenation of the current attention context vector c_t , the current decoder state s_t and the embedding vector of the candidate word $\hat{y}_t$ as the continuous observation, $o(t) = [\ c_t;\ s_t;\ E(\hat{y}_t)]$.

2. **Reward** Although the model computes reward at the end of the sentence, it is calculated as cumulative sum of each time step. Hence, various evaluation metrics has been modified to be calculated at each time step.

   - **BLEU Score**
     They decompose smoothed version of BLEU as described in section 2.4.1 and use the difference of partial BLEU scores as the reward, that is:

$$r_t^Q = \begin{cases} \text{BLEU}(\text{Y}^t, \text{Y}^*) - \text{BLEU}(\text{Y}^{t-1}, \text{Y}^*) & t < T \\ \text{BLEU}(\text{Y}, \text{Y}^*) & t = T \end{cases}$$

     Where Y is the output, Y^t is the cumulative output at t and Y^* is the reference.

   - **Average Proportion**
     Following the definition in section 2.4.2, Average proportion at each time step is

calculated as:

$$\mathrm{AP}_t = \begin{cases} 0 & t < T \\ \mathrm{AP} & t = T \end{cases}$$

In other words, AP_t is calculated at the end of sentence and for all other time steps it would be zero.

- **Consecutive Wait**
  CW has been used as it is described and formulated in 2.4.2.

The reward for delay r_t^D is computed as combination of AP and CW, which has been devised as:

$$r_t^D = \alpha.[\mathrm{sign}(\mathrm{CW}_t - \gamma) + 1] + \beta.\lfloor \mathrm{AP}_t - \delta \rfloor \tag{3.1}$$

Where α, β, γ, and δ are preset constants. Finally the total reward at each time step is achieved by setting $r_t = r_t^Q + r_t^D$.

Instead of estimating state-value function, the new agent makes use of a softmax policy, which means parameters of policy π_θ is estimated by an RNN with a softmax function at final layer:

$$s'_t = \mathcal{F}_\theta(s'_{t-1}, o_t)$$
$$\pi_\theta(a_t|a_{<t}, o_{\leq t}) \propto \mathcal{G}_\theta(s'_t)$$

Where s'_t is the internal state of the agent. the RNN is then trained using policy gradient algorithm [99], which maximizes the following expected rewards:

$$\mathcal{J} = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} r_t\right]$$

Whose gradient is:

$$\nabla_\theta \mathcal{J} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta \sum_{t=1}^{T} r_t\right]$$

Since we are computing this gradient at the end of sentence, it can be calculated as:

$$\nabla_\theta \mathcal{J} = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|.)\mathcal{R}_t\right]$$

$$\mathcal{R}_t = \sum_{k=t}^{T}\left[r_k^Q + r_k^D\right]$$

In practice, this gradient is estimated by sampling multiple action trajectories from the current policy π_θ, collecting the corresponding rewards.

3.3.3 Supervised Learning Techniques

Models that use supervised learning techniques fix the translation component and aims to train the agent in a supervised way. Teaching the agent to find optimal sentence boundaries requires an algorithmic method to generate oracle action sequences for each source and target language pair. Designing a mechanism for obtaining optimal segments to generate accurate translations as fast as possible is under-explored [9] and remained an unresolved challenge. In this section we will explain the supervised model introduced by Zheng et al. [101] which use pairs of (input, reference) sentences to train the oracle agent.

For an input sentence $\mathrm{X} = \{x_1, \ldots, x_I\}$ and its target translations $\mathrm{Y}^* = \{y_1^*, \ldots, y_R^*\}$, Zheng et al. [101] follows the algorithm 3 to generate their oracle action sequences.

Algorithm 3 Oracle action sequences for supervised models

Init $i \leftarrow 1$, $j \leftarrow 0$, $a_0 \leftarrow$ READ
while $j < R$ **do**
  $t = i + j$
  if $i = I$ or $\mathrm{Rank}(y_r^* | x_{\leq i}) \leq \beta$ **then**
    $a_t =$ WRITE
    $j \leftarrow j + 1$
  else
    $a_t =$ READ
    $i \leftarrow i + 1$
return actions

Where β is a user defined positive integer and $x_{\leq i}$ is the prefix of first i-th words in source sentence. $\mathrm{Rank}(y_j^* | x_{\leq i})$ counts the rank of j-th target word in predictions of translation component given the first i source words.

Aside from the parameter β, in order to have more control over balancing the trade-off between translation accuracy and delay, the model filter the action sequences if their AL (described in section 2.4.2) is higher than a fixed constant α. Additionally, the probability ρ is introduced to serve as threshold for choosing the best action; i.e. the READ action will be chosen if its probability is greater than ρ.

A transformer-based [96] NMT model trained on full-sentences serves as translation component. For the choice of Agent, Zheng et al. [101] use a 512 unit GRU layer, a 64 unit fully-connected layer followed by another fully-connected layer with 2 dimensions. The input to the Agent consists of the encoder and decoder representation with cross attention scores from the translation component.

3.3.4 Attention-based Methods

The attention-based models are adaptive policies that rely solely on attention weights to decide the right time to commit an output. The main obstacle in using the conventional attention mechanism in online settings is that it relies on the whole source words to compute the context vector. The monotonic attention process first introduced by Raffel et al. [80] to model a linear-time complexity hard attention mechanism for tasks that input-output attention weights are approximately monotonic. In this section, we will describe the original monotonic attention model and then we describe how this model has been modified by Arivazhagan et al. [6] to attend to previous time steps, and the formulation proposed by ma et al. [61] to use monotonic attention with the transformer model.

Given the input sequence $\mathrm{X} = \{x_1, \ldots, x_I\}$ we want to transform it into translated words $\mathrm{Y} = \{y_1, \ldots, y_J\}$ using an end-to-end model. The original work uses an RNN-based model which updates the parameters as follows:

$$h_i = \mathcal{F}_{\mathrm{RNN-ENC}}(x_i, h_{i-1}) \tag{3.2}$$

$$s_j = \mathcal{F}_{\mathrm{RNN-DEC}}(y_{j-1}, s_{j-1}, c_j) \tag{3.3}$$

$$y_j = \mathcal{F}_{\mathrm{softmax}}(s_j, c_j) \tag{3.4}$$

The Softmax step generates a distribution over output tokens. The context vector c_j in standard soft attention model is computed as:

$$e_i^j = \mathcal{F}_{\mathrm{MLP}}(h_i, s_{j-1}) \tag{3.5}$$

$$\alpha_i^j = \mathcal{F}_{\mathrm{softmax}} = \frac{\exp(e_i^j)}{\sum_{k=1}^{I} \exp(e_k^j)} \tag{3.6}$$

$$c_j = \sum_{n=1}^{I} \alpha_n^j h_i \tag{3.7}$$

To compute the context vector c_j in equation 3.7 we need to receive the encoder representations from all input time steps and it cannot be directly applied to simultaneous settings.

The idea in monotonic attention is to assign the context vector c_j to a particular encoder state h_{q_j} where q_j is the input token at output time step j. Beginning with encoder state at $i = q_{j-1}$, the algorithm produces a Bernoulli selection probability $P_{i,j}$ and decides either to WRITE and set $q_j = i$, or READ one more source token and move to $i + 1$. This process

can be formulated as:

$$e_i^j = \mathcal{F}_{\text{Monotonic}}(s_{j-1}, h_i) \tag{3.8}$$

$$P_{i,j} = \mathcal{F}_{\text{Sigmoid}}(e_i^j) \tag{3.9}$$

$$z_{i,j} \sim \text{Bernoulli}(P_{i,j}) \tag{3.10}$$

If $z_{i,j} = 1$, the next action will be WRITE and we set $q_j = i$ and $c_j = h_{q_j}$; Otherwise, one more token will be read from input sequence and we set $q_j = i$. During training, the hard assignment of c_j impede the back-propagation process. Raffel et al. [80] proposed an alternative alignment α to replace the softmax attention. α_i^j which computes the probability of attending to state h_i at output time step j, can be formulated recurrently as:

$$\alpha_i^j = P_{i,j}\left((1 - P_{i-1,j})\frac{\alpha_{i-1}^j}{p_{i-1,j}} + \alpha_i^{j-1}\right) \tag{3.11}$$

The formula in 3.11 can be transformed to be computed in parallel and more efficiently [80]:

$$\alpha_{:j}^{:} = P_{:,j}\texttt{cumprod}(1 - P_{:,j})\texttt{cumsum}\left(\frac{\alpha_{:}^{j-1}}{\texttt{cumprod}(1 - P_{:,j})}\right) \tag{3.12}$$

Where $\texttt{cumprod}(x) = [1, x_1, x_1x_2, \ldots, \prod_{n=1}^{I-1} x_n]$ and $\texttt{cumsum}(x) = [x_1, x_1{+}x_2, \ldots, \sum_{n=1}^{I} x_n]$. While monotonic attention is successful at decoding in linear time, its attention mechanism is only attending to one word which can hugely affect the translation accuracy. To address this problem, Chiu and Raffel [22] Proposed Monotonic Chunkwise Attention (MoChA) where the model was allowed to apply soft attention to a fixed-length chunk of encoder states.

Monotonic Infinite Lookback Attention (MILk)

In addition to MoChA, Arivazhagan et al. [6] introduced MILk where the model can use soft attention on all the encoder states in previous time steps $h_1, \ldots, h_{q_j}$. During inference, when monotonic attention chooses to stop at q_j, MILk computes soft attentions:

$$e_k^j = \mathcal{F}_{\text{MLP}}(h_k, s_{j-1}), \quad k \in 1, 2, \ldots, q_j \tag{3.13}$$

Then the context c_j can be computed as:

$$c_j = \sum_{i=1}^{q_j} \frac{\exp(e_i^j)}{\sum_{l=1}^{q_j} \exp(e_l^j)} h_i \tag{3.14}$$

The equation 3.14 allows the attention context vector c_j to attend to all previous time steps $1 \dots q_j$. During training, instead of finding optimal value for q_j, MILk compute the expected value of c_j in the following way:

$$\beta_i^j = \sum_{k=i}^{I} \left(\frac{\alpha_k^j \exp(e_i^j)}{\sum_{l=1}^{k} \exp(e_i^j)} \right) \tag{3.15}$$

$$c_j = \sum_{i=1}^{I} \beta_i^j h_i \tag{3.16}$$

Since the model can attend to all previous time steps, a trivial optimal policy is to wait until the end of the sentence, where the attention mechanism has access to all encoder representations to generate highly accurate outputs. To avoid this scenario, Arivazhagan et al. [6] proposed to extend the negative log likelihood with a differentiable cost for loss. The new loss function is defined as:

$$L(\theta) = -\sum_{(\mathrm{X},\mathrm{Y})} \log P(\mathrm{Y}|\mathrm{X};\theta) + \lambda \mathcal{C}(\mathrm{G})$$

Where λ is a user-defined constant, $\mathcal{C}$ is a differentiable metric for delay (e.g. Differentiable Average Lagging (DAL) which described in section 2.4.2), and $\mathrm{G} = \{g_1, \dots g_J\}$ where each $g_j = \sum_{n=1}^{I} n\alpha_n^j$.

Monotonic Multihead Attention (MMA)

To utilize the power of transformers in attention-based models, Ma et al. [61] modified the multihead architecture in transformers to augment it with monotonic attention mechanism. Having multiple heads for attention layer allows transformers to generate different distributions with each head. Given queries Q, keys K and values V, the Multihead attention can be computed as:

$$\mathrm{Attention}(Q,K,V) = \mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.17}$$

$$\mathrm{head_h} = \mathrm{Attention}(QW_h^Q, KW_h^K, VW_h^V) \tag{3.18}$$

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{concat}(\mathrm{head}_1, \dots, \mathrm{head_H})W^O \tag{3.19}$$

Where H is number of attention heads, and d_k is the dimension of attention head. MMA applies monotonic attention to each attention head, by modifying the formulation of

monotonic mechanism for the h-th head in the l-th layer as:

$$e_{i,j}^{l,h} = \left(\frac{m_j W_{l,h}^K (s_{i-1} W_{l,h}^Q)^T}{\sqrt{d_k}} \right)_{i,j} \tag{3.20}$$

$$p_{i,j}^{l,h} = \text{Sigmoid}(e_{i,j}) \tag{3.21}$$

$$z_{i,j}^{l,h} = \text{Bernoulli}(p_{i,j}) \tag{3.22}$$

This way each attention head make independent decisions from each other. For developing selection process, Ma et al. [61] explored two types of alignments: **MMA-H**(ard) which follows the standard monotonic attention mechanism and use hard-alignment formulation (formula 3.11) to compute expected alignments for each head, and **MMA-IL**(infinite lookback) which uses MILk selection criteria to be able to attend all previously visited encoder states. Similar to formula 3.13, the soft attentions in MMA-IL is computed as:

$$u_{i,j}^{l,h} = \phi_{\text{mlp}} \left(\frac{m_j \hat{W}_{l,h}^K (s_{i-1} \hat{W}_{l,h}^Q)^T}{\sqrt{d_k}} \right)_{i,j} \tag{3.23}$$

Then the equations 3.15 and 3.16 will be used to calculate the expected value of c_i. During test time, both MMA-H and MMA-IL for each decoder layer l, attention head h at decoder time step i use the sampling process of standard monotonic attention to set the encoder step at $t_i^{l,h}$. Then the context c_i can be computed as follows:

$$c_i^l = \text{Concat}(c_i^{l,1}, c_i^{l,2}, \ldots, c_i^{l,H}) \tag{3.24}$$

$$\text{where } c_i^{l,h} = f_{\text{context}}(h, t_i^{l,h}) = \begin{cases} m_{t_i^{l,h}} & \text{MMA} - \text{H} \\ \sum_{j=1}^{t_i^{l,h}} \frac{\exp(u_{i,j}^{l,h})}{\sum_{j=1}^{t_i^{l,h}} \exp(u_{i,j}^{l,h})} m_j & \text{MMA} - \text{IL} \end{cases} \tag{3.25}$$

The decoding strategy for both models are depicted in Algorithm 4.

3.3.5 Imitation Learning Methods

Similar to attention-based models, methods that employ Imitation Learning (IL) focus on training the translation component concurrently with a supervised agent. The agent in IL requires an oracle action sequence to use as reference in its supervised training, and since the translation component is not fixed, the oracle action sequences proposed for supervised methods cannot be applied to this framework. Zheng et al. [102] and Arthur et al. [9] proposed two different methods for solving the simultaneous translation task via imitation learning. We will start this part by explaining the general structure in [9], then we will briefly discuss the model proposed in [102].

Algorithm 4 Monotonic decoding for both MMA-IL and MMA-H models.

Input: x = source tokens, h= encoder states, i=1, j=1, $t_0^{l,h} = 1$, y_0 = BOS
while $y_{i-1} \neq$ `<eos>` **do**
  $t_{\max} = 1$
  h = empty sequence
  for $l \leftarrow 1$ to L **do**
    for $h \leftarrow 1$ to H **do**
      for $j \leftarrow t_{i-1}^{l,h}$ to $|x|$ **do**
        $p_{i,j}^{l,h} = \mathcal{F}_{\text{Sigmoid}}\ (\mathcal{F}_{\text{Monotonic}}(s_{i-1}, m_j))$
        if $p_{i,j}^{l,h} > 0.5$ **then**
          $t_i^{l,h} = j$
          $c_i^{l,h} = \mathcal{F}_{\text{context}}(h, t_i^{l,h})$
          Break;
        else if $j > t_{\max}$ **then**
          Read token x_j
          Calculate state h_j and append to h
          $t_{\max} = j$
      $c_i^l = \text{Concat}(c_i^{l,1}, c_i^{l,2}, \ldots, c_i^{l,H})$
    $s_i^l = \text{DecoderLayer}^l(s_{1:i-1}^l, s_{1:i-1}^{l-1}, c_i^l)$
  $y_i = \text{Output}(s_i^L)$
  $i = i + 1$

The agent[2] in the model introduced by Arthur et al. [9] selects the next action based on previously generated actions $a_{<t}$, the prefix of words in the source sentence $x_{<i}$, and the target translations generated so far $y_{<j}$. The translation component[3] on the other hand, responds to the commands from the agent by either reading one more from input stream or generating a new translated token in the target language. Algorithm 5 describes how the agent and translation component interact with each other.

Exposure bias can become a serious problem in IL algorithms with behavioral cloning, that is the agent and interpreter are only perceiving the restricted trajectories of correct predictions. In order to address this problem, Arthur et al. [9] use scheduled sampling, where they disturb each sample with an specific probability β. During training, after applying scheduled sampling, the interpreter generates output and based on that the agent decides the appropriate action for the next time steps, before updating the parameters of both the agent and interpreter.

[2]The paper used the word "Programmer" to refer to the agent

[3]The original paper calls it "Interpreter"

Algorithm 5 How agent and interpreter communicate with each other in IL.

$i, j \leftarrow 0$
while a stopping condition is not met **do**
  $t \leftarrow i + j$
  $s_{t+1} \leftarrow \mathcal{F}_{\text{Agent}}(s_t, [a_t, g_j, h_i])$
  $P_{\text{Agent}} \leftarrow \mathcal{F}_{\text{Softmax}}(s_{t+1})$
  $a_{t+1} \sim P_{\text{Agent}}$
  if $a_{t+1} = \text{READ}$ **then**
    $i \leftarrow i + 1$
    $h_i \leftarrow \mathcal{F}_{\text{ENC}}(h_{i-1}, x_i)$
  else
    $j \leftarrow j + 1$
    $g_j \leftarrow \mathcal{F}_{\text{Interpreter}}(g_{j-1}, y_{j-1}, h_{\leq i})$
    $P_{\text{Interpreter}} \leftarrow \mathcal{F}_{\text{Softmax}}(g_j)$
    $y_j \leftarrow P_{\text{Interpreter}}$

Generating oracle action sequences

In order to compute the optimal sequence of READs and WRITEs, Arthur et al. [9] use fast-align [34] to generate symmetrized alignments between source and target tokens. This helps the model to find shortest phrases δ_j that contain enough information for the interpreter to genrate the target word y_j. The algorithm detect optimal phrases is described in Algorithm 6.

Where α_i^j means x_i is aligned to y_j. To ensure the information in each phrase is enough for translation, In cases of many to one alignment of source words to target words, Algorithm 6 chooses the last aligned word.

Training the agent with restricted imitation learning

Instead of training a separate agent component, Zheng et al. [102] proposed to add a delay token <ε> to target vocabulary. Whenever the translation component is not sure about the prediction and needs to receive more input words, emits <ε> token which is equivalent to a READ action. Otherwise the WRITE action is selected and the predicted token will be sent to output. Based on which action the model choose, for a sequence $(x_{<i}, y_{<j})$ where $x_{<i}$ is a prefix of input sentence X and $y_{<j}$ is the partially generated target sentence, a transition function δ will be applied to input input and output streams:

$$\delta((x_{<i}, y_{<j}), a) = \begin{cases} (x_{<i} \circ x_i, y_{<j}) & \text{if a} = \text{<}\varepsilon\text{>} \\ (x_{<i}, y_{<j} \circ a) & \text{Otherwise} \end{cases} \tag{3.26}$$

Where $x_{<i} \circ x_i$ means the word x_i will be concatenated to the prefix sequence $x_{<i}$.

For defining the oracle policy, instead of choosing an specific optimal trajectory, the model restricts the available action options in a way that our model would be able to

Algorithm 6 Generating oracle action sequences for the agent in an imitation learning method.

Require: **a:** Symmetrized alignment of x and y.

$\delta_{\text{READ}} := -1$
for $j \in \text{range}(0, |y|)$ **do**
  $\delta_j \leftarrow \max[\{i \in a_i^j\}]$
  for $(\delta_j - \delta_{\text{READ}})$ times **do**
    emit READ
  $\delta_{\text{READ}} \leftarrow \max(\delta_j, \delta_{\text{READ}})$
  emit WRITE

generate accurate translations in a reasonably quick way. Given the pair of full-sentences (X, Y) and a pair of prefixes $(x_{<i}, y_{<j})$ where $x_{<i}$ is a prefix of X and $y_{<j}$ is a prefix of Y and $(x_{<i}, y_{<j}) \neq (\text{X}, \text{Y})$, the following formulation can bound the available options for the next time step between two user-defined constants γ and η:

$$\pi^*_{x,y,\gamma,\eta}(x_{<i}, y_{<j}) = \begin{cases} \{\texttt{<}\varepsilon\texttt{>}\} & \text{if } x_{<i} \neq \text{X and } d' \leq \gamma \\ \{y_j\} & \text{if } y_{<j} \neq \text{Y and } d' \geq \eta \\ \{\texttt{<}\varepsilon\texttt{>}, y_j\} & \text{Otherwise} \end{cases} \tag{3.27}$$

Where $d' = i - \lambda j$ and $\lambda = I/J$. The ratio for the λ cannot be obtained for the test set and the model uses the ratio for the training data as an approximate for the test set.

3.4 Policy for End-to-end Speech Translation Models

In section 2.3 we briefly discussed various attempts to build simultaneous translation system based on the cascaded architecture. In this section we will review 3 recently proposed ideas by Ren et al. [83] and Ma et al. [60] and [62].

SimulSpeech

Ren et al. use the standard encoder-decoder Transformer model [96] as the basic structure of their model. The following changes has been applied to allow the model to receive speech signals instead of text words, and generate translations in real-time:

- Multiple convolutional layers added as proposed in speech pre-net [89] to extract speech features.
- The speech frames are very granular compared to their text counterparts. Each speech frame is around 10ms and a word on average needs 27 frames to get recognized. Hence, checking every single frame in the input for the agent to decide the next action is not optimal. To solve this problem, a separate *speech segmenter* component receives extracted audio features and detect word boundaries incrementally using an ASR model with Connectionist Temporal Classification (CTC) [38] loss. The features of all

the audio frames contained in the word are being combined to be used by the agent to detect the best next action.

- The encoder is modified similar to Ma et al. [59] to generate representations unidirectionally. The self attention mechanism masks future time frame to assure that each input can only see its previous inputs.

- The agent use the *wait-k* (section 3.2) policy to find the optimal action for the next time step. Following Ma et al. [59], the decoder is masked to ensure the final outputs are generated based on the wait-k policy.

In addition to these alterations, the authors applied two techniques to improve the model while training:

1. **Attention-level knowledge distillation** that transfer the knowledge from an auxiliary ASR and NMT models that share the encoder and decoder with the SimulSpeech model. distilling knowledge is happening through multiplication of matrices from ASR, NMT and SimulSpeech.

2. **Data-level knowledge distillation** uses full-sentence NMT to help the model converge faster. If the source speech X is transcribed into source text W, the full-sentence NMT generates translation Y, and the SimulSpeech is being trained to generate Y if it receives X.

SimulMT to SimulST

Similar to SimulSpeech, Ma et al. [60] attempts to apply proposed policies for SiMT to end-to-end speech translation systems. The basic structure of SimulST is the offline speech translation model proposed by Di Gangi et al. [37] which uses CNN layers to extract speech features and a S-Transformer to translate audio signals. The model make use of the static policy of wait-k, and adaptive policy of Monotonic Multihead Attention (MMA) mechanism (described in section 3.3) to decide the best action for the next time step.

While the wait-k policy struggles to work with high variance between the input and output lengths in SiST, the MMA mechanism can fairly address the problem by efficiently checking every frame. However, the performance of the model decreases as the level of granularity of the input increases. For solving this problem, Ma et al. uses two different speech segmenters to split incoming stream into smaller groups:

- **Fixed Speech Segmenter:** Every fixed number of speech frames the model asks the agent to decide about the next action. Assuming $\Delta\tau$ is the pre-determined length of the fixed time frame, which consists of multiple of T_s, and $r_e = \text{int}(I/J)$ where I is the length of the input signal and J is the length of the output, then the probability

P_{tr} of choosing a time frame at encoder time step i can be defined as:

$$P_{tr}(i) = \begin{cases} 1 & \text{if } \mathrm{mod}(i.r_e.T_s, \Delta\tau) = 0 \\ 0 & \text{Otherwise} \end{cases}$$

If the probability $P_{tr} > 0.5$, then the model asks the agent whether to READ or WRITE for the next time step, otherwise the model keeps reading.

- **Flexible Speech Segmenter:** The speech signals are segmented at the word or phoneme level. The Gentle aligner[4] generate alignments $\mathcal{A}$ between the source speech and their corresponding words. If the encoder state h_i aligns to the token $\mathcal{A}(h_i)$, then the probability P_{tr} is formulated as:

$$P_{tr}(i) = \begin{cases} 0 & \text{if} \mathcal{A}(h_i) = \mathcal{A}(h_{i-1}) \\ 1 & \text{Otherwise} \end{cases}$$

If $P_{tr} > 0.5$, the model forward the current input to the simultaneous speech translation component, otherwise the model waits for more input.

Augmented Memory Transformer

Ma et al. [62] proposed a new structure for the encoder in an end-to-end speech translation system based on Augmented memory mechanism proposed by Wu et al. [100], which can effectively apply self-attention mechanism on smaller chunks of high dimensional audio signals in the input.

The model can be divided into two components: (1) The *Augmented Memory Encoder* and (2) The *Simultaneous Decoder*. The former make adjustments to the transformer-based encoder [96] to attend to chunks of input instead of the full input stream, and the latter modify decoder to work with wait-k policy. The self-attention mechanism in the transformers maps the input features into Query ($\mathbf{Q}$), Key($\mathbf{K}$), and Value($\mathbf{V}$):

$$\mathbf{Q} = W_q \mathrm{H}$$
$$\mathbf{K} = W_k \mathrm{H}$$
$$\mathbf{V} = W_v \mathrm{H}$$

Where $\mathrm{H} = \{h_1, \dots\}$ is the encoder's input at a certain layer. Subsequently, the self-attention at each position j is computed as:

[4]https://lowerquality.com/gentle/

$$\alpha_{jj'} = \frac{\exp(\beta.\mathbf{Q}_j^T \mathbf{K}_{j'})}{\sum_k \exp(\beta.\mathbf{Q}_j^T \mathbf{K}_k)} \tag{3.28}$$

$$\mathbf{Z}_j = \sum_{j'} \alpha_{jj'} \mathbf{V}_{j'} \tag{3.29}$$

Where $\beta = \frac{1}{\sqrt{D}}$ is a scaling factor. The self-attention at each position is attending to the entire input stream, which is not efficient in real-time calculations. The augmented memory encoder [100] applies self-attention to smaller chunks of S = $\{s_1, \dots\}$, where each segment s_n overlaps with adjacent segments to the left and right. The overlap between s_n and s_{n-1} is marked with $\mathbf{l}_n$ with size L. The overlap between s_n and s_{n+1} is marked with $\mathbf{r}_n$ with size R. The non-overlapping section is marked with $\mathbf{c}_n$ with size C. The Query, Key, Value will be computed for each segment:

$$\begin{aligned}
\mathbf{q}_n &= W_q(\mathbf{l}_n, \mathbf{c}_n, \mathbf{r}_n, \sigma_n) \\
\mathbf{k}_n &= W_k(\mathbf{M}_{n-N:n-1}, \mathbf{l}_n, \mathbf{c}_n, \mathbf{r}_n) \\
\mathbf{v}_n &= W_v(\mathbf{M}_{n-N:n-1}, \mathbf{l}_n, \mathbf{c}_n, \mathbf{r}_n)
\end{aligned}$$

Assuming $\mathbf{x}_k$ is the k-th feature vector in the input sequence X, σ_n can be computed as $\sigma_n = \sum_{\mathbf{x}_k \in s_n} \mathbf{x}_k$. The memory banks in $\mathbf{M}_{n-N:n} = [\mathbf{m}_{n-N}, \dots, \mathbf{m}_{n-1}]$ are being computed as $\mathbf{m}_n = \sum_{j'} \alpha_{-1,j'}(\mathbf{v}_n)_{j'}$. The self-attention for each speech chunk can be calculated as:

$$\begin{aligned}
\alpha_{jj'} &= \frac{\exp \beta.(\mathbf{q}_n^T)_j (\mathbf{k}_n)_{j'}}{\sum_k \exp(\beta.(\mathbf{q}_n^T)_j (\mathbf{k}_n)_k)} \\
(\mathbf{z_n})_j &= \sum_{j'} \alpha_{j,j'} (\mathbf{v}_n)_{j'}
\end{aligned}$$

Where j is bounded between $N + L$ and $N + L + C$. N is predetermined constant which decides how many frames overlap between two adjacent segment.

The decoder uses wait-k policy on fixed segments of speech signals which is selected by an *speech segmenter* proposed by Ma et al. [60] and discussed earlier in this section. Assume the $\mathbf{W} = [\mathbf{w_1}, \dots]$ are referred to sequence of chunks, and $W_s(k)$ and $W_e(k)$ are referring to the start and end encoder state index of $\mathbf{w_1}$. The decoding algorithm is illustrated in algorithm 7.

While the algorithm with any translation policy introduced in sections 3.2 and 3.3, Ma et al. make use of the static policy wait-k.

Algorithm 7 The decoding algorithm for Augmented Memory Transformer.

Require: Chunk-based simultaneous policy P
Require: Streaming input X. Memory banks $\mathbf{M}$. Prediction Y
Require: Maximum memory size N . Decision chunk size W
Require: Central context size C. Encoder pooling ratio R
Require: $i = 1, n = 1, k = 1$
Require: $W_e(1) = 1, y_0 =$BOS

while a stopping condition is not met **do**
  if $W_e(k) + W > n.C.R$ **then**
    $\mathbf{z}_n, \mathbf{m}_n = \text{Encoder}(\mathbf{s}_n, \mathbf{M}_{nN:n-1})$
    $\mathbf{Z} = [\mathbf{Z}, \mathbf{z}_n], \mathbf{M} = [\mathbf{M}, \mathbf{m}_n]$
    $n = n + 1$
  $\mathbf{w_k} = \text{Summarize}(\mathbf{Z}_{W_s(k):W_e(k)})$
  $p_{i,k} = P([\text{Y}_{1:i-1}], \mathbf{w}_k)$
  if $p_{i,k} > 0.5$ **then**
    $y_i = \text{Decoder}([\text{Y}_{1:i-1}], \mathbf{Z})$
    $i = i + 1$
  else
    $W_s(k + 1) = W_e(k) + 1$
    $k = k + 1$

3.5 Summary

In this chapter, we studied traditional challenges for simultaneous translation task that we still need to deal with in our current models. We grouped proposed policies into two categories of *static* and *adaptive* policies and we gave an overview of different architectures introduced for each group. We then briefly reviewed recent approaches to build end-to-end speech translation system. In the chapter 4 we will show how adding prediction mechanism to RL-based methods can improve the performance of the model.

Chapter 4

Prediction Improves Simultaneous Machine Translation

One of the next significant challenges in machine translation research is to make translation ubiquitous using real-time translation. Simultaneous machine translation aims to address this issue by interleaving reading the input with writing the output translation. RL-based architectures (sections 3.3) for Simultaneous Machine Translation (SiMT) systems [86, 24, 41] use an AGENT to control an incremental encoder-decoder (or sequence to sequence) NMT model. Each READ adds more information to the encoder RNN, and each WRITE produces more output using the decoder RNN. In this work, we propose adding a new action to the AGENT: a PREDICT action that predicts what words might appear in the input stream.

Prediction was previously proposed in simultaneous statistical machine translation [40] but has not been studied in the context of Neural Machine Translation (NMT). In SiMT systems, prediction of future words augments the encoder-decoder model with possible future contexts to produce output translations earlier (minimize delay) and/or produce better output translations (improve translation quality).

Prediction for the translation task in simultaneous environments can happen either in the source or target sentences. The decoder in NMT models is capable of working similar to language models and determine the most probable words in the upcoming time steps. Training the MT with partial sentences instead of full sentences can also improve the implicit target-side prediction [59]. Estimating the most probable next word in the source language is harder to achieve, as the components of an encoder-decoder model are not designed to make assumptions about future time steps of the source sentence. Our PREDICT action in this chapter attempts to help the translator to make better assumptions about following words in the source stream.

4.1 Simultaneous Translation Framework

An agent-based framework whose actions decide whether to translate or wait for more input is a natural way to extend neural MT to simultaneous neural MT and has been explored in [86, 41] (also in section 2.3) which contains two main components: The ENVIRONMENT which receives the input words $\mathrm{X} = \{x_1, \ldots, x_I\}$ from the source language and incrementally

generates translated words $\mathrm{Y} = \{y_1, \ldots, y_J\}$ in the target language; And the AGENT which decides an action for each time step, a_t. The AGENT generates an action sequence $\mathrm{A} = \{a_1, \ldots, a_T\}$ to control the ENVIRONMENT.

Previous models only include two actions: READ and WRITE. We extend the model by adding the third action called PREDICT. Action READ is simply sending a new word to the ENVIRONMENT and generating a candidate word in the target language. In action WRITE, the AGENT takes current candidate word and sends it to the output. For PREDICT, the AGENT predicts the next word in the input and treats it like a READ action. The following section explains how the ENVIRONMENT deals with different actions.

4.1.1 ENVIRONMENT

The ENVIRONMENT is an attention-based Encoder-Decoder MT system [11] which is adopted to simultaneous translation task. The Encoder receives the embedded representation of the input words (including predicted ones) and converts them into context vectors $\mathrm{H}_i^\rho = \{h_1, \ldots, h_{i+\rho}\}$ using a gated RNN (LSTM) where i is the number of input words so far and ρ is the number of predicted words since the last READ. Whenever the AGENT decides to READ, ρ will be set to 0, and $h_i = \mathcal{F}_{\mathrm{ENC}}(h_{i-1}, x_i)$, where x_i is the next input words ($i \leq I$). But if the action is PREDICT, $\rho > 0$, the AGENT predicts a new word x'_ρ and the context vector $h_{i+\rho} = \mathcal{F}_{\mathrm{ENC}}(h_{i+\rho-1}, x'_\rho)$ will be added to $\mathrm{H}_i^\rho = \{h_1, \ldots, h_{i+\rho}\}$.

At each time step t, the decoder uses the current context vectors (H_i^ρ) to generate the next candidate output ($\hat{y}_t$):

$$
\begin{aligned}
c_t &= \mathcal{A}_{\mathrm{ATT}}(s_{j-1}, \mathrm{H}_i^\rho) \\
z_t &= \mathcal{F}_{\mathrm{DEC}}(s_{j-1}, y_{j-1}, c_t) \\
P(\hat{y}_t | y_{<j}, H_i^\rho) &= \mathcal{G}(y_{j-1}, s_j, c_t) \\
\hat{y}_t &= \arg\max_{\hat{y}} p(\hat{y} | y_{<j}, \mathrm{H}_i^\rho)
\end{aligned}
$$

where y_{j-1} is the previous output word, and $\mathcal{A}_{\mathrm{ATT}}$ is an attention model [11], $\mathcal{F}$ and $\mathcal{G}$ are nonlinear functions (tanh), and c_t is the current context vector.

If the action a_t is either READ or PREDICT the current candidate $\hat{y}_t$ will be ignored (wait for better predictions). But in the case of WRITE, the candidate $\hat{y}_t$ is produced as the next output word y_j and then the decoder state will be updated ($y_j \leftarrow \hat{y}_t$, $s_j \leftarrow z_t$).

Note that as soon as the AGENT decides to READ, all the hidden vectors generated by PREDICT actions will be discarded ($\mathrm{H}_i^0 = \{h_1, \ldots, h_i\}$).

Figure 4.1 shows an example of how a sentence can be translated using our modified translation framework. The pseudo-code is depicted in Algorithm 8.

Algorithm 8 AGENT with prediction mechanism

Require: Policy π_θ, Input buffer X, Output buffer Y.
Init $j \leftarrow 0$, $i \leftarrow 1$, $\rho \leftarrow 0$
while $j < J$ **do**
  $t \leftarrow t + 1$
  $\hat{y}_t, z_t, o_t \leftarrow \mathcal{F}_{\text{DEC}}(s_j, y_j, H_i^\rho)$
  $a_t \sim \pi_\theta(a_t; a_{<t}, o_{<t})$
  if $a_t =$ READ and $x_i \neq \langle eos \rangle$ **then**
    $i \leftarrow i + 1$, $\rho \leftarrow 0$
    $h_i \leftarrow \mathcal{F}_{\text{ENC}}(h_{i-1}, x_i)$
    $\text{H}_i^\rho \leftarrow H_{i-1}^\rho \cup \{h_i\}$
    if $|\hat{\text{Y}}| = 0$ **then**
      $s_0 \leftarrow \mathcal{F}_{\text{INIT}}(H_i^\rho)$
  else if $a_t =$ WRITE **then**
    $j \leftarrow j + 1$
    $s_j \leftarrow z_t$, $y_j \leftarrow \hat{y}_t$
    $\text{Y} \Leftarrow y_j$
    if $y_j = \langle eos \rangle$ **then**
      break
  else if $a_t =$ PREDICT **then**
    $\text{pw} \leftarrow x_i$
    if $\rho > 0$ **then**
      $\text{pw} \leftarrow x'_\rho$
    $\rho \leftarrow \rho + 1$, $x'_\rho \leftarrow \mathcal{F}_{\text{PRED}}(\text{pw})$
    $h_{i+\rho} \leftarrow \mathcal{F}_{\text{ENC}}(h_{i+\rho-1}, x'_\rho)$
    $\text{H}_i^\rho \leftarrow \text{H}_i^{\rho-1} \cup \{h_{i+\rho}\}$

4.1.2 AGENT

The AGENT is a separate component which examines the ENVIRONMENT at each time step and decides on the actions that lead to better translation quality and lower delay. The Agent in the *greedy decoding* framework [41] was trained using reinforcement learning with the policy gradient algorithm [99], which observes the current state of the ENVIRONMENT at time step t as o_t where $o_t = [c_t;\ z_t;\ y_j]$. A RNN with one hidden layer passed through a softmax function generates the probability distribution over the actions a_t at each step. Therefore, policy π_θ will be computed as:

$$s'_t = \mathcal{F}_\theta(s'_{t-1}, o_t)$$
$$\pi_\theta(a_t | a < t, o \leq t) \propto \mathcal{G}_\theta(s'_t)$$

Where s'_t is the hidden state of the AGENT's RNN.

4.2 Training the AGENT with Prediction

In order to speed-up the training process, we have restricted AGENT's options by removing redundant operations. As illustrated in Figure 4.2, after a series of WRITE, the AGENT cannot choose to PREDICT, and after a sequence of PREDICTs, READ is not an option.

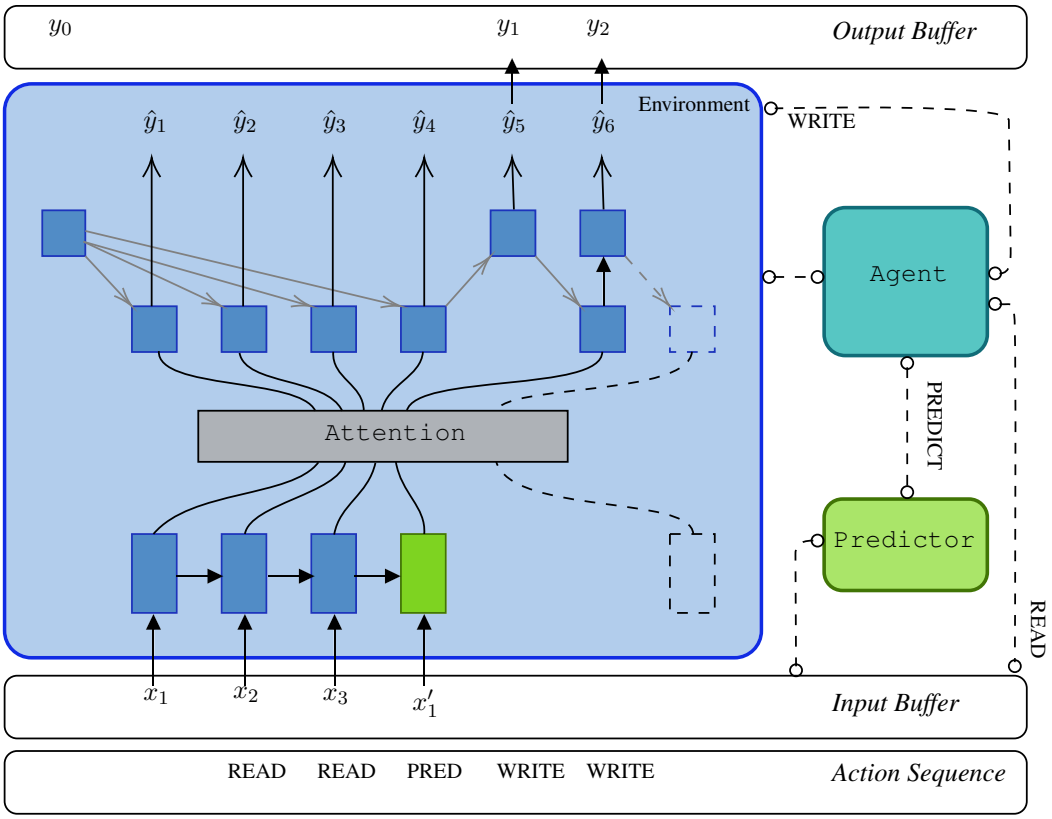


Figure 4.1: A schematic of our model. The ENVIRONMENT starts with reading the 'Start of Sentence' symbol as x_1 and generating y_1 from the decoder. Based on the output of the network at each time step, the AGENT decides whether to READ, WRITE or PREDICT for the following time steps.

Reward Function: The total reward at any time step is calculated as the cumulative sum of rewards for actions at each preceding step. All the evaluation metrics have been modified to be computed for every time step.

Quality: We use a modified smoothed version of BLEU score multiplied by Brevity Penalty [57] for evaluating the impact of each action on translation quality. At each point in time, the reward for translation quality is:

$$r_t^Q = \begin{cases} \Delta\text{BLEU}(t) & t < T \\ \text{BLEU}(\text{Y}, \text{Y}^*) & t = T \end{cases}$$

The $\Delta\text{BLEU}(t)$ is the difference between BLEU score of the translated sentence at the previous time step and the current time step; $\Delta\text{BLEU}(t) = \text{BLEU}(\text{Y}^t, \text{Y}^*) - \text{BLEU}(\text{Y}^{t-1}, \text{Y}^*)$; where Y^t is the prefix of the translated sentence at time t.

Delay: The Delay reward is used to motivate the AGENT to minimize delay. We use Average Proportion (AP) [24] for this purpose, which is the average number of source words needed when translating each word. Given the source words X and translated words Y, AP_t can be

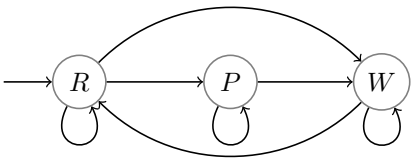


Figure 4.2: Action transition graph. R, P, and W stands for READ, PREDICT and WRITE actions respectively.

computed as:

$$\text{AP} = \frac{1}{I \times J} \sum_{j} \psi(j)$$

$$\text{AP}_t = \begin{cases} 0 & t < T \\ \text{AP} & t = T \end{cases}$$

Where $\psi(j)$ denotes the number of source words the WRITE action uses at time step j (for any other actions, $\psi(j)$ would be zero). The delay reward is smoothed using a *Target Delay* which is a scalar constant denoted by δ [41]:

$$r_t^D = \lfloor \text{AP}_t - \delta \rfloor$$

Prediction Rewards for Quality and Delay alone do not motivate the Agent to choose prediction and in preliminary experiments, after a number of steps, the number of prediction actions became zero. We address this problem by defining Prediction Quality (PQ) which rewards the Agent for changes in BLEU score after each prediction action. By initializing $r_0^p = 0$, the prediction reward can be written as:

$$r_t^p = \begin{cases} \Delta\text{BLEU}(t) & a_t = \text{WRITE}, a_{t-1} = \text{PREDICT} \\ r_{t-1}^p & a_t = \text{WRITE}, a_{t-1} \neq \text{PREDICT} \\ 0 & \text{otherwise} \end{cases}$$

Final Reward The final reward function is calculated as the combination of quality, delay, and prediction rewards:

$$r_t = \alpha \; r_t^Q + \beta \; r_t^D + \lambda \; r_t^p \tag{4.1}$$

The trade-off between better translation quality and minimal delay is achieved by modifying the parameters α, β, and λ.

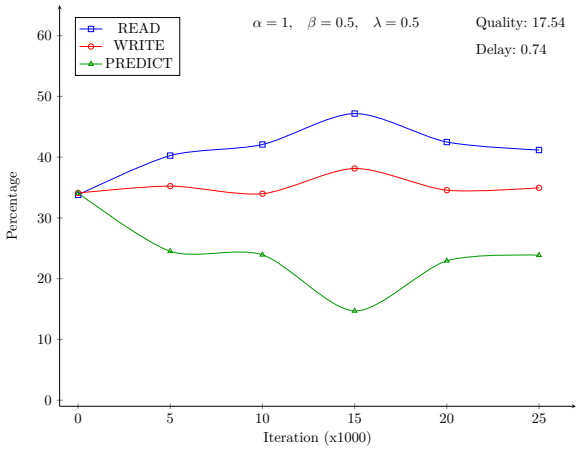
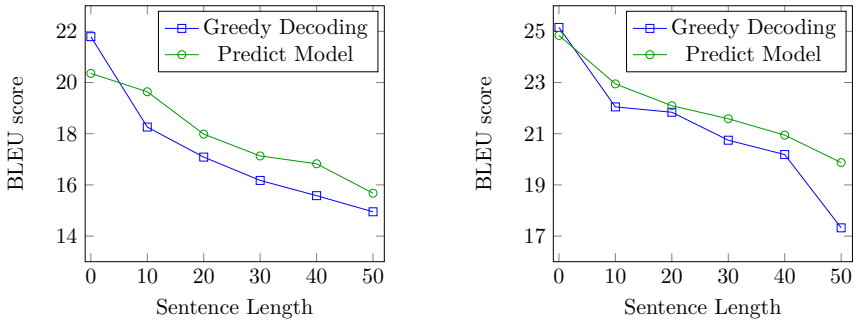


Figure 4.3: Action distribution for English to German translation in the first 25000 iterations. Each point in the graph is the average action percentage over the previous 5000 iterations.

Reinforcement Learning is used to train the AGENT using a policy gradient algorithm [41, 99] which searches for the maximum in $\mathcal{J} = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} r_t\right]$ using the gradient: $\nabla_\theta \mathcal{J} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\mathcal{R}_t\right]$ where $\mathcal{R}_t = \sum_{k=t}^{T} r_k$. The gradient for a sentence is the cumulative sum of gradients at each time step. We pre-train the ENVIRONMENT on full sentences using log-loss $\log P(\mathrm{Y}|\mathrm{X})$.



(a) EN→DE (b) DE→EN

Figure 4.4: Comparing translation quality between prediction model and greedy decoding method for various sentence lengths.

4.3 Experiments

We train and evaluate our model on English-German (EN-DE) in both directions. We use WMT 2015 for training and Newstest 2013 for validation and testing. All sentences have been tokenized and the words are segmented using byte pair encoding (BPE) [88].

Model Configuration For a fair comparison, we follow the settings that worked the best for the greedy decoding model in [41] and set the target delay d^* for the AGENT to 0.7. The ENVIRONMENT consists of two unidirectional layers with 1028 GRU units for encoder

and decoder. We train the network using AdaDelta optimizer, a batch of size 32 and a fixed learning rate of 0.0001 without decay. We use softmax policy via recurrent networks with 512 GRU units and a softmax function for the AGENT and train it using Adam optimizer [51]. The batch size for the AGENT is 10, and the learning rate is 2e-6. The word predictor is a two layer RNN Language model which consists of two layers of 1024 units, followed by a softmax layer. The batch size is 64 with a learning rate of 2e-5. The predictor has been trained on the WMT'16 dataset and tested on Newstest'16 corpora for both languages. The perplexity of our language model is reported in Table 4.1. We set $\alpha = 1$, $\beta = 0.5$ and $\lambda = 0.5$. We tried different settings for these hyperparameters during training and picked values that gave us the best Quality and Delay on the training data.

Results and Analysis Figure 4.4 shows that as the sentence length increases, prediction helps translation quality due to complex reordering and multi-clausal sentences; However, for shorter samples where the structure of the sentences are simpler, the prediction action cannot improve translation quality. Table 4.2 compares our model with the Greedy Decoding (GD) model in terms of translation quality and latency. It shows that the prediction mechanism outperforms the GD model in terms of BLEU and average proportion (AP).

The delay evaluation measure (AP) counts about the same number of READs and WRITEs. It does not account for less delay as longer sentences are produced. A better measure than AP might be needed to emphasize delay differences. Therefore we also report the average segment length (μ), which is computed as the average number of consecutive READs in each sentence. In both EN→DE and DE→EN experiments, our model constantly decreases the segment length by around 1 word which results in less latency.

In order to evaluate the effectiveness of our proposed reward function for PREDICT action, we have explored various values for its hyper-parameters (α, β, and λ). Our empirical results show that the best trade-off between quality and delay is achieved when around 20 percent of the actions are PREDICT for both EN→DE and DE→EN translation tasks (Figure 4.3). When there is no reward for PREDICT action ($\lambda = 0$), the AGENT prefers other actions, and the number of PREDICT actions turns into zero immediately after training the AGENT. If the reward for prediction is valued too highly, the ENVIRONMENT depends more on predicted words and the translation quality decreases.

Figure 4.5 depicts more detailed results on experiments with λ. Setting λ to zero makes the model to ignore the PREDICT action after the first few epochs. This results in generating a policy similar to the original model with no prediction mechanism. By setting λ to 1, motivate the agent the predict action forty percent of times. The generated policy relies heavily on the information from future time steps which negatively affect the translation quality.

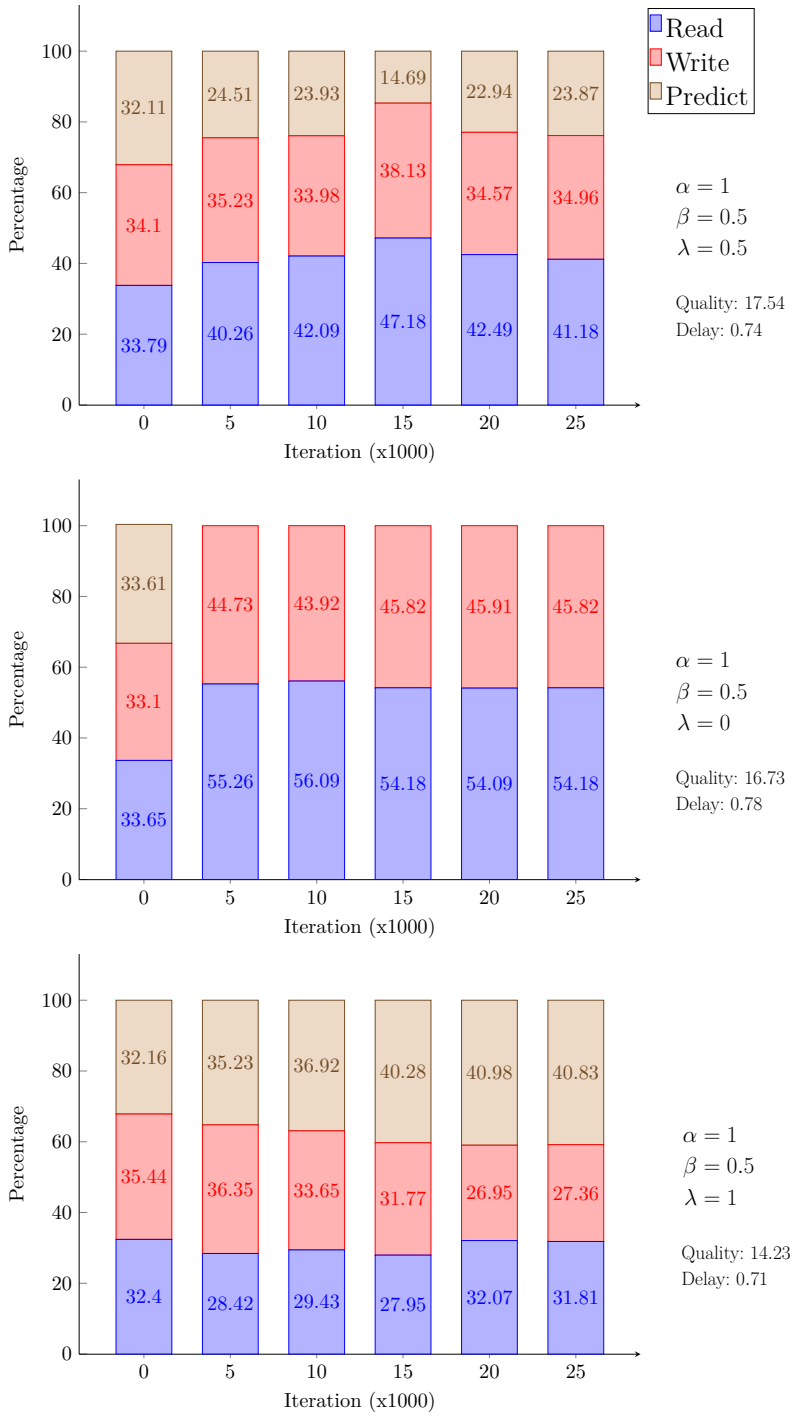


Figure 4.5: Action distribution in the first 25000 iterations for varying λ values. The numbers in each bar are the average action percentage over the previous 5000 iterations. All these graphs are for English to German translation; However, very similar behaviour was also observed for German to English translation.

	Perplexity	
	Test	Train
English	17.5917	8.1350
German	19.8532	11.1808

Table 4.1: Performance of our predictor for both English and German languages.

	EN→DE			DE→EN		
	BLEU	AP	μ	BLEU	AP	μ
GD	16.75	0.79	5.6	21.43	0.77	6
PM	**17.54**	**0.74**	**4.7**	**21.83**	**0.70**	**5.2**

Table 4.2: Comparison of translation quality (BLEU), Average Proportion (AP), and average segment length (μ) for Greedy Decoding (GD) model with Prediction Mechanism (PM) model.

4.4 Shortcomings and Drawbacks

Though the extended Environment-Agent model is fast and improves the results, there are lots of rooms that need to be improved. The main shortcomings of the model can be summarized as follows:

- **The Agent doesn't consider acoustic cues** One of the most important applications of Simultaneous NMT is to be used in speech to speech interpretation systems. Lots of critical information like pauses, tone of voices, stress, etc conveyed in the audio signals of that message[84], and all of them will missed during transcribing the signals. Consequently, the Agent does not take into account any acoustic cues.
- **Quality is not high** The translation quality of simultaneous systems are still low, there is large gap in quality of translation between simultaneous and standard translation systems.
- **The model is Complicated** The simultaneous NMT architecture without prediction action is still complicated and lots of hyper-parameters need to be tuned. Adding prediction action makes the model more complicated and increases the training time for the Agent significantly.
- **The attention mechanism is not trained for simultaneous architecture** Since we are pretraining the Environment before the Agent starts to train, the attention mechanism is tuned to work in an offline (i.e. non-simultaneous) setting. This means the attention weights are trained and work based on the assumption that they have access to the whole sentence, even during training the Agent and the test time.

4.5 Summary

We introduce a new prediction action in a trainable Agent for simultaneous neural machine translation. With prediction, the Agent can be informed about future time steps in the input

stream. Compared to a very strong baseline our results show that prediction can lower delay and improve the translation quality, especially for longer sentences and translating from an SOV (subject-object-verb) language (DE) to an SVO language (EN).

Chapter 5

Translation-based Supervision for Policy Generation in SiMT

5.1 Supervised Approach

In simultaneous machine translation, we aim to receive the input sequence $\mathrm{X} = \{x_1, \ldots, x_I\}$ incrementally and to transform it to the translated tokens $\mathrm{Y} = \{y_1, \ldots, y_J\}$ as accurately and as fast as possible (by producing the output while reading the input). Our supervised framework contains two main components: The INTERPRETER which takes subsets of the input sequence $\mathrm{X}_i = \{x_1, \ldots, x_i\}$ and generates *partial* translations $\mathrm{Y}^i = \{y_1^i, \ldots, y_{m_i}^i\}$[1]; And the AGENT which decides whether to send the next input subset X_{i+1} to the INTERPRETER or not, based on the currently generated output tokens (and possibly other useful information) which represents the state of the INTERPRETER.

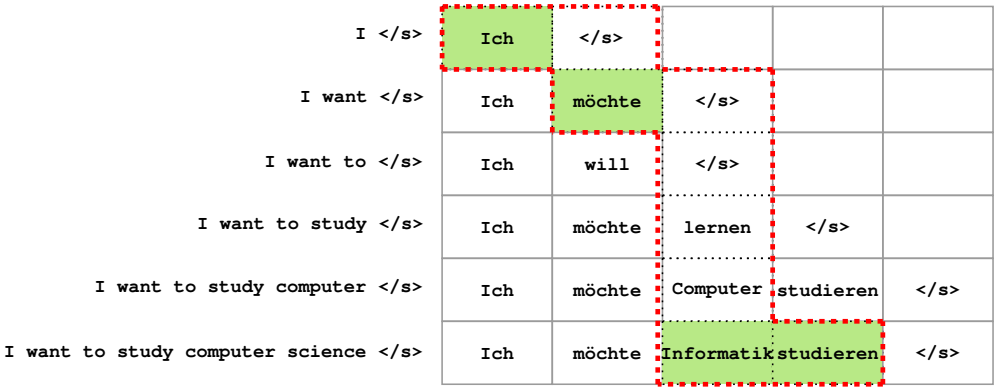
5.1.1 Reference Action Sequences

The central idea behind our method is our novel definition of an *optimal segment* in simultaneous translation. We define an *optimal segment* as a segment of the input sequence which leads the INTERPRETER to produce the exact same target words in both simultaneous (partial) translation and full-sentence translation. The initial optimal segment is a prefix of the input and each subsequent optimal segment is a further slice of the input. A reference action sequence (or an oracle action sequence) is a sequence that splits the input sentence into optimal segments.

To achieve this goal we build a *Partial Translations Table* (PTT), each row of which corresponds to the translation of a prefix of the source sentence and each column represents a translated word. More explicitly, the first row contains the translation of the first input token followed by the end-of-sentence token `</s>` and each next row will incrementally consider one more input token than the immediate row before. By definition, the last row will be equivalent to the full-sentence translation. Figure 5.1(a) shows an example of a generated Partial Translations Table.

[1]The last partial translation equals to the full-sentence translation. i.e. $\mathrm{Y}^I = \mathrm{Y}$

(a) Example of a created Partial Translations Table



(b) The created optimal segments in the input stream based on the Partial Translations Table above. The reference action sequence of READs ($\mathfrak{R}$) and WRITEs ($\mathfrak{W}$) can be created based on the occupied cells in each column.

$\mathfrak{R}$	I		want		to	study	computer	science
$\mathfrak{W}$		Ich		möchte				Informatik	studieren

Figure 5.1: Example of partial translations table. Our oracle action trajectory is indicated by the red dash line and the green cells are the words that our policy choose to WRITE. The subset of input words at each row is extended with `</s>` token to improve the quality of partial translations.

We construct the reference action sequence of optimal segments using the PTT. Starting from the leftmost word in the first row, we compare the content of each column with the word in its own column at the last row. If they are the same we add WRITE action to the oracle sequence and move to the right cell. Otherwise, we will add READ and move down to the lower cell.

Figure 5.1(b) demonstrates the optimal segments created using PTT for the example input sentence "`I want to study computer science`". Using this stream, the first element of the action sequence will be $\mathfrak{R}$ since the READ row is occupied in the first column. The next element will be $\mathfrak{W}$ since the WRITE row is occupied in the second column. If we continue to look at each column and generate a READ or WRITE action we will end up with the reference action sequence of "$\mathfrak{R}$ $\mathfrak{W}$ $\mathfrak{R}$ $\mathfrak{W}$ $\mathfrak{R}$ $\mathfrak{R}$ $\mathfrak{R}$ $\mathfrak{R}$ $\mathfrak{W}$ $\mathfrak{W}$".

We now formally define the reference action sequence generation algorithm. Let i be the number of READs and j be the number of WRITEs at a given time step $t = i + j$. At time step $t + 1$, if the token y_j in the full-sentence translation Y, equals to the j'th token in the partial translation Y^i, it means that the current subset of the input words X_i is sufficient to generate the j'th word in the output and our agent should choose to WRITE. Otherwise, we will add a READ to our action sequence. Algorithm 9 defines the extraction process.

Our proposed algorithm for generating the reference action sequence is completely agnostic about the underlying partial translation generation component. We can choose offline (non-simultaneous) or simultaneous translation models to create the Partial Translations Table.

Algorithm 9 Generating action sequences

Init $i \leftarrow 1$, $j \leftarrow 1$, actions $\leftarrow$ READ
$\mathrm{Y} = \mathrm{Translate}(\mathrm{X}_I)$
while $j < J$ **do**
  $\mathrm{Y}^i = \mathrm{Translate}(\mathrm{X}_i)$
  if $j < \mathrm{len}(Y^i)$ and $y^i_j = y_j$ **then**
    actions $\leftarrow$ actions +WRITE
    $j \leftarrow j + 1$
  else
    actions $\leftarrow$ actions +READ
    if $i < I$ **then**
      $i \leftarrow i + 1$
return actions

5.1.2 Supervised Training

For any input sentence X and its corresponding partial translations $\mathrm{Y}^1, \ldots, \mathrm{Y}^J$, we can generate an oracle action sequence $\mathrm{A} = \{a_1, \ldots, a_T\}$, where $T = I + J$. This action sequence will be used as the ground-truth for training our action generation policy in a supervised framework. At time step t, the policy observes the current state of the INTERPRETER o_t by receiving x_i and y_j alongside a history of actions η_h from previous time steps, where $\eta_h = \{a_{t-1}, \ldots, a_{t-h}\}$. We train a recurrent neural network (RNN) to maximize the probability of the current action a_t given all the previous observations and actions:

$$\max P(a_t | a < t, o \leq t; \theta)$$

where θ is the set of parameters for our RNN.

At each time step, we pass the source token, the target token, and all the actions in η_h through separate embedding layers. The action embedding layer is shared among actions. The source and target embeddings will be concatenated and go through a linear layer. A separate linear layer receives concatenated action embeddings to extract features. the concatenation of the outputs of the two linear transformation layers go through 4 LSTM layers to predict the next action. Figure 5.2 depicts the structure of our Agent. While the agent is modeled as an LSTM, the underlying translation system we use is a Transformer NMT model.

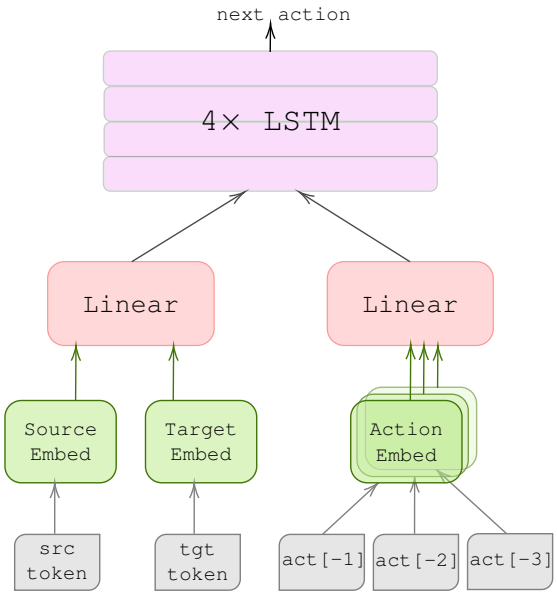


Figure 5.2: Architecture of our Agent. Passing the last 3 generated tokens alongside the source and target tokens helps the model prevent long consecutive sequences of READs or WRITEs.

5.1.3 Improving Robustness

Minimizing the discrepancy between training and inference, also known as exposure bias [81], is an essential step in successfully training a supervised agent. Our Agent has to learn how to generalize when facing unseen partial translations and make sure the errors do not compound with each mistake in its trajectory.

The Interpreter Since we do not change the translation component, if we consider the previously generated tokens instead of the ground truth translations, we can guarantee that training and inference are using the same procedure in creating the Partial Translations Table. Although this would make the training process slower, it provides more accurate results.

The Agent To prevent landing in unfamiliar areas of the prediction space with certain Agent mistakes we augment our training data of action sequences with additional examples that are introduce distortions in the action sequence [9]. For each input sentence X we generate its oracle action sequence A and the set of observations O $= \{o_1, \ldots, o_T\}$ with $o_t = (x_i, y_j, \eta_h)$ to be used for training the Agent. Then we randomly choose a time step t and check the training example (o_t, a_t) to see if it has the following conditions:

- $a_t \notin \{a_1, a_T\}$.
- $x_i \neq$ `</s>`.
- $y_j \neq$ `</s>`.

If all of the conditions are true, then we swap the action a_t from READ to WRITE or vice versa, and we generate a new oracle action sequence for the rest of the sentence. Figure 5.3 shows how we can recover from a wrong WRITE and READ actions. The observation o_t is updated according to the newly generated oracle.

Beam search vs. Greedy decoding Following previous work, to boost the simultaneous nature of the model, we use greedy decoding while performing the simultaneous decoding in the INTERPRETER.

The simultaneous decoder (aka the INTERPRETER) can use beam search to get more accurate results at expense of the translation speed. However, this modification does not substantially change the comparison with baseline methods, so we leave this for future work.

Since the Partial Translation Table can be generated in an offline manner (and can be considered a pre-processing step), we use beam search decoding when creating these partial translations without any negative effect on inference for simultaneous translation at decoding time.

5.2 Experimental Setup

5.2.1 Dataset

We use IWSLT14 [19] and WMT15[2] German to English and IWSLT15 [58] Vietnamese to English translation tasks to examine the effectiveness of our approach.

Following [35],we tokenize and lower-case the German to English data and BPE [88] sub-word tokenize both sides.

For IWSLT14 data, we choose 10K separate BPE merge operations resulting in approximately 8.8K German and 6.6K English sub-word types. We train our models on 160K sentences and keep 7K of the train data as the validation set. We test our models on a concatenation of dev2010 and tst2010 to tst2013 (a total of 6750 sentence pairs).

For WMT15, we choose BPE merge operations such that we achieve a joint BPE vocabulary of size 32K types. We will randomly choose 20 percent of the sentence pairs for training the Agent[3]. We will also use the same subset for distorting the samples and together we will end up having 1.5M training examples. We use newstest2013 with 3000 sentence pairs as the validation set and test on newstest2015 with 2169 sentences.

For IWSLT15 Vietnamese to English, we use the tokenized corpus prepared by [58] which contains 17K English and 7.7K Vietnamese types. The data contains 133K training sentence pairs. We use tst2012 (1553 sentence pairs) for validation and tst2013 (1268 sentence pairs) to test our models.

[2]http://www.statmt.org/wmt15/

[3]Making use of more data did not affect our results.

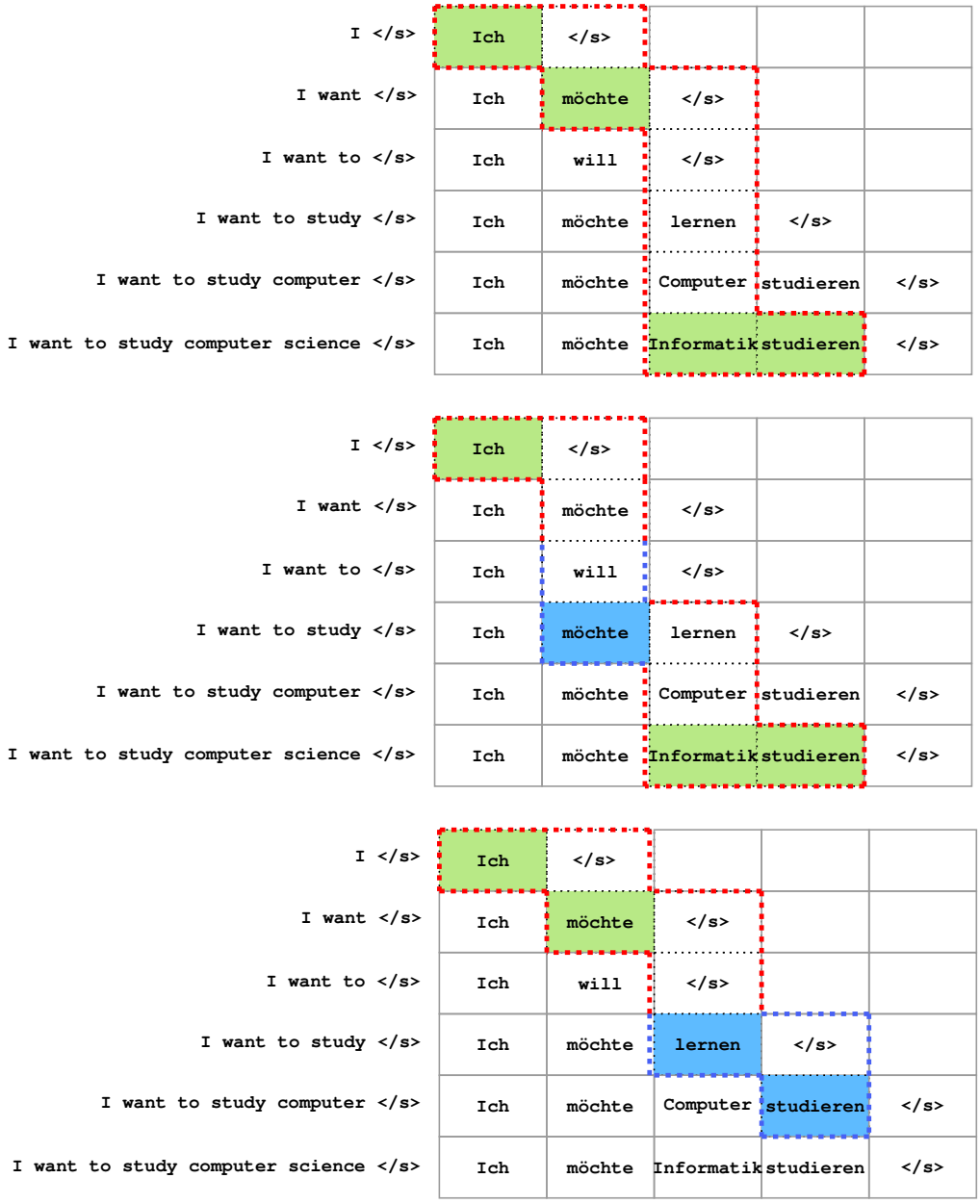


Figure 5.3: The first table shows an example of our oracle action sequence. In the middle table, we switch the second WRITE action to a READ action. We will continue reading until we observe the same word, then we can come back to the original path. In this scenario, the final translation does not change. In the last table, we distort the 5th READ into a WRITE action. Here, the generated translation is slightly changed as we are writing the wrong word.

	DE→EN		VI→EN	
	BLEU	AL	BLEU	AL
wait-∞ + eval-wait-5	27.92	4.48	19.93	**3.5**
wait-∞ + oracle policy	**34.25**	**4.14**	**28.29**	4.6
multi-path + eval-wait-5	30.50	4.67	20.18	**2.33**
multi-path + eval-wait-5 + oracle policy	**33.04**	**3.75**	**25.25**	4.35
wait-∞ + [101]	31.18	4.50	20.71	3.31

Table 5.1: Comparison between different oracles on IWSLT14 DE → EN and IWSLT15 VI → EN datasets.

5.2.2 Evaluation

We evaluate the translation quality of the translated sentences using tokenized word-level BLEU score [75][4]. We use Average Lagging (AL) [59] to measure the decoding latency for our models. AL measures the average number of words we are lagging behind the ideal policy with no delay.

Our goal in this chapter is to minimize the decoding latency (get the lowest average lagging possible) while trying to balance the loss in the translation quality as measured by BLEU score. Among the two measures, BLEU normally gets lower in production settings, as the data is not as clean and well prepared as the benchmark data. On the other hand, improving average lagging is a more reliable method to improve the user experience (in simultaneous translation).

5.2.3 Model Configuration

We use fairly standard Transformer-based NMT system as implemented in Fairseq [73] for all of our experiments[5]. We augment the implementation to perform simultaneous translation and we incorporate our Agent trained to produce read and write actions with the base NMT system and decoder. Our INTERPRETER can be additionally configured to replicate the model proposed in [35].

Table 5.2 summarizes all the hyper-paraameters we used during training process. Our agent consists of 4 unidirectional LSTM layers [45] with 512 units in each layer. We use a history of the last 3 previous action tokens (i.e. $h = 3$). Each embedding and linear layer generates a vector of dimension 512.

We train our Agent using Adam [51] optimizer. The initial learning rate is set to 0.0008 and we use a fixed learning rate scheduler with a shrink factor of 0.95.

[4]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

[5]The implementation will be available on Github.

Hyperparameter	Value
embed dim	512
num layers	4
residuals	True
Learning rate	8e-4
LR scheduler	fixed
LR shrink	0.95
force aneal	4
dropout	0.4
clip norm	5
optimizer	Adam
update freq	4
loss	label smoothed cross entropy
label smoothing	0.1

Table 5.2: The hyperparameters of the agent's training process.

After training for 50 epochs, our agent that is trained to produce optimal segments obtains an average accuracy of 92.3% on the training data and 83% on the dev set when averaged over 4 experiments on the IWSLT datasets.

5.3 Results and Analysis

We compare the performance of our system against two baselines:

- **Wait-∞** also known as Wait-Until-End, where the full sentences are read during training before generating any translations . This is an extreme case of *Wait-k* strategy [59] in which the model reads the first k words of the input and then performs consecutive WRITE/READ actions afterwards.

- **Multi-path** model proposed by [35]. The multi-path model jointly trains the translation component on decoding strategies with various latencies, which makes this model effective on a wide range of delay values.

For both of these settings, we will use the *eval-wait-k* [59] policy, where each written word is exactly k words behind the source side. We compare the performance of our model against the state-of-the-art (SotA) in section 5.3.2.

5.3.1 Performance of Oracle Policy

Table 5.1 compares the performance of our policy for DE $\rightarrow$ EN and VI $\rightarrow$ EN language pairs on IWSLT dataset. *wait-∞ + eval-wait-5* corresponds to the model in which we decode an offline translation component using wait-5 policy. In *wait-∞ + oracle policy*, we use an offline Interpreter to generate partial translations in PTT, and then we use our algorithmic oracle to generate policies. The policy in *multi-path + eval-wait-5* is generated by decoding multi-path model with wait-5 policy. *multi-path + eval-wait-5 + oracle policy*

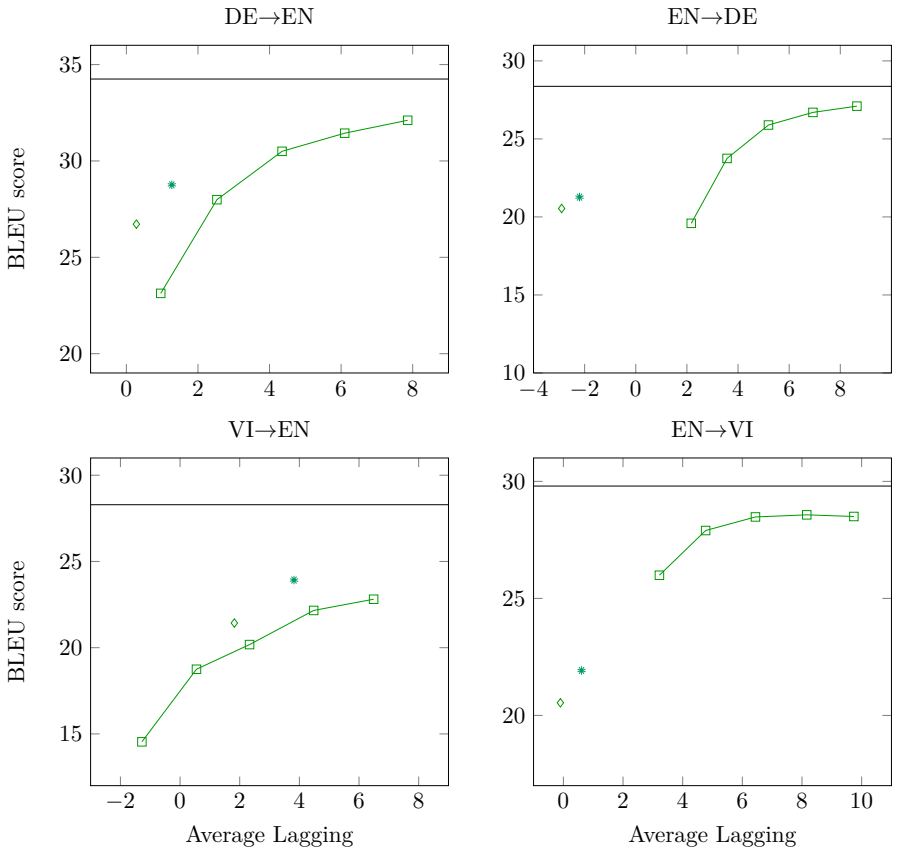


Figure 5.4: Translation quality vs delay of our oracle policy compared to multi-path policy. Markers: —— represents wait-∞ model. —□— corresponds to the multi-path model. each point in the curve is generated using eval-wait-k policy for k $\in \{1, 3, 5, 7, 9\}$. ✳ points to the wait-∞ + our trained policy. ◊ represents the multi-path + eval-wait-5 + our trained policy.

is the model that we first use the multi-path model with wait-5 decoding path to generate partial translation and then we use our algorithm to generate oracle policy. Our choice of *eval-wait-5* for decoding the *multi-path* model is based on the fact that the *multi-path* model generates more accurate translations via *eval-wait-5* compared to offline decoding [35]. The numbers in the last row of Table 5.1 are generated by using an offline INTERPRETER and the oracle in [101].

Our oracle policy vs. static policies

On both offline and multi-path settings, our oracle policy outperforms *eval-wait-5* policy both in terms of translation quality and latency on DE $\rightarrow$ EN language direction. Our policy on VI $\rightarrow$ EN experiment is slightly more delayed, but the translation quality is considerably more accurate. The performance of the *eval-wait-5* policy improves when we replace the offline INTERPRETER with the multi-path model. However, the multi-path model does not outperform *eval-wait-5* combined with our oracle policy.

The delay of our oracle policy decreases when we change the underlying translation component to a multi-path translation model so using a base translation model trained to handle shorter segments can be combined with an agent trained on our reference actions to improve the average lagging.

Our oracle policy vs dynamic policies

[101, 102, 9] propose algorithmic methods for generating oracle action sequences. The oracle in [102] introduces a new delay token in the target vocabulary which makes their INTERPRETER incompatible to our agent.

[9] use alignment-based segments to jointly train their policy and translation components. Only using alignments extracted from fast-align [34] on an offline translation component gives us very low BLEU scores.

The oracle in [101] is the closest model to our work. Unlike our policy, they compare each word in partial translations to the *target words* to find the optimal action sequence. The numbers in Table 5.1 correspond to the closest results we could get by searching for the optimal hyperparameters. Our oracle policy outperforms their oracle policy in both language pairs.

5.3.2 Trained Agent Performance

Figure 5.4 shows the results of our trained policy in comparison with the policy trained with multi-path method on DE $\leftrightarrow$ EN and VI $\leftrightarrow$ EN language pairs. We will apply our policy on two different settings: (1) When we use a translation model trained on full sentences to fill up the partial translations table (offline + our policy) and (2) generating translations in PTT via multi-path model decoded with eval-wait-5 trajectory (multi-path + our policy). In both settings, we are using a beam of size 5 for generating each partial translation.

	With distortion		Without distortion	
	BLEU	AL	BLEU	AL
EN→DE	20.54	-2.9	20.99	0.4
DE→EN	26.72	0.28	29.45	4.3
EN→VI	20.54	-0.1	26.6	6.5
VI→EN	21.43	1.82	21.38	2.4

Table 5.3: The effect of training our Agent with and without distorted samples.

One explanation for the negative values of the AL in Figure 5.4 is the implicit prediction in the translation component, as we discussed in chapter 4. Training the Interpreter via partial sentences enable the model to generate translations sooner with fewer information in the source language. This way, the hidden states of the encoder can capture some future information from the source sentence, and the decoder can learn the the most probable word for the next time step based on the structure of the target sentence. So the distinction between source-side and target-side prediction is not clear and requires further analysis.

First, we investigate how the capabilities of the INTERPRETER can affect the quality of the generated policy. By comparing the offline INTERPRETER (marked with star) and multi-path INTERPRETER (marked with diamond) we can see that the offline model can achieve a higher translation quality with a more delayed policy. This is in align with our experimental results with their oracles (Section 5.3.1) where using multi-path INTERPRETER gave us less delayed policy by sacrificing translation quality.

By comparing the multi-path model (marked with square) with our trained policy we can see that in both DE → EN and VI → EN language pairs our policy outperforms the multi-path model in both settings. This is because our policy gives the INTERPRETER the freedom to translate quickly at the beginning and have consecutive reads later in the sentence, which consequently results in higher translation quality and lower latency.

On multi-path settings, for DE → EN language pair, our trained policy has a much lower latency while at the same time it is considerably more accurate in terms of translation scores compared to the *eval-wait-1* policy. Similarly, for VI → EN experiment, the translation quality of our policy is close to *eval-wait-7* with a delay less than *eval-wait-5* policy. This implies that we can boost the performance of the previously proposed methods by combining their translation system with an agent trained using our proposed oracle policy.

Translating from English

In EN → DE and EN → VI our agent generates translation with much lower delays with slightly lower translation scores. While the performance of our model is quite consistent across all four language directions, our trained agent performs relatively poorer when translating from English. Figure 5.5 gives a better picture on the performance of our model across various language directions. Although our model generates translations that are very

close to full-sentence translations, the BLEU score comparison of our generated translations with full-sentence translations indicate that the translation quality of $\sim$ 78 for translating *to* English drops to translation quality of $\sim$ 67 for translating *from* English.

This happens because translation in this direction can be monotonic without losing much in terms of translation scores. In this scenario, our model has a higher chance of making mistakes as the length of the sentence gets longer; while the multi-path model while follows the static policy of *eval-wait-k* obtains slightly higher translation scores.

Another reason that contributes is that our model tends to predict `</s>` token in the target language sooner than expected and generates shorter sentences. This situation can be moderated by making the model to wait longer to see if any information in the source remained untranslated. We also assume that by making the model more robust in generating `</s>` can help us improve the results when translating from English.

5.3.3 Effect of adding distorted samples

Table 5.3 compares the performance of our agent when we train it with or without distorted samples. While adding distorted samples lowers the BLEU score in most language directions, the latency of our system drops significantly. This help our model to perform above the Multi-path curve in the trade-off between translation quality and delay. One potential reason is that after adding distorted samples, the model learns to finish translation process early. This approach is highly effective in longer sentences where translated words at the end of the sentences are not very accurate. But in shorter sentences this strategy may decrease translation quality.

5.3.4 Performance on WMT15 Dataset

In order to compare the performance of our trained agent with other state-of-the-art methods, we will conduct experiments on the WMT15 DE→EN dataset. Our oracle policy is able to generate action sequences with AL of around 6, while our translation quality is as good as the offline model. On datasets with longer sentences like WMT, achieving oracle-level accuracy is a harder task. However, compared to a static policy like *eval-wait-k*, our Agent generates action sequences that perform similar to *eval-wait-3*. Although the policies in MoChA [22], MILK [6] and their recent version of MMA-H and MMA-IL [61] generate more accurate translations, the delay of their systems is considerably higher than our approach.

The BLEU scores of the previously proposed supervised learning approach in [101] is much worse than our model when we consider AL values that are similar to ours. We obtain +2 points higher BLEU score for translation quality with almost the same delay as their model.

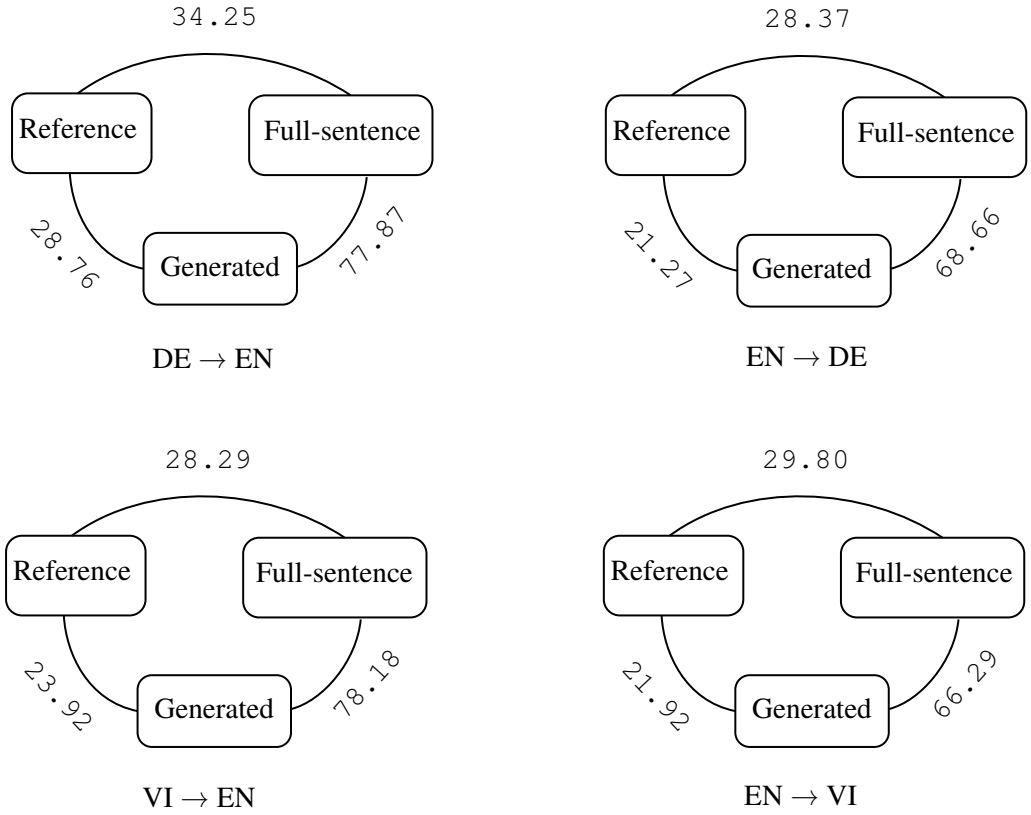


Figure 5.5: Comparing translation quality of different translations on EN ↔ DE and EN ↔ VI language pairs. The numbers are BLEU scores of connected boxes. *reference* is referring to the reference translations. *Full-sentence* denotes the full-sentence translations generated by the interpreter, and *Generated* is the output of our trained agent.

$\mathfrak{R}$	aber	wo	sind	wir	nützlich	?
$\mathfrak{W}$	but	where	are	we	useful	?

Table 5.4: An example translation from our model where word-by-word translation generates good translations and waiting for 5 words is redundant.

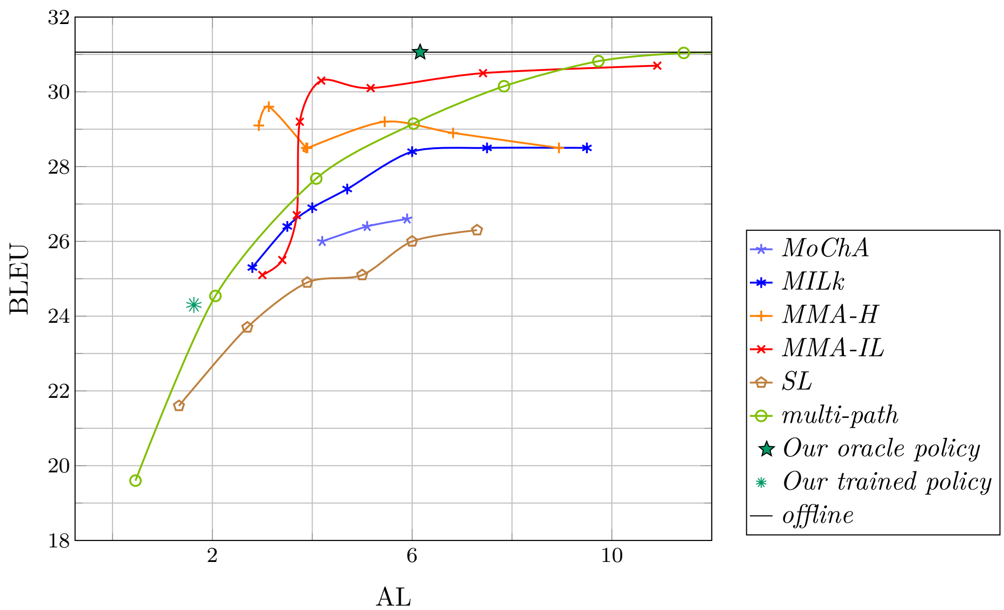


Figure 5.6: Comparison against SoTA results on WMT15 DE → EN language pair. SL is the supervised learning approach proposed by [101]. "*Our oracle policy*" generates the oracle action sequences for partial translations generated by wait-∞ translation model, and "*Our trained policy*" is our Agent trained to learn that oracle.

(a) Generated actions and translations via our supervised agent

𝔑	wir		mussten		uns	wegen	der	[an]	[wäl]	te	[us]	w.	sorgen	machen	.									
𝔚		we		had												to	worry	about	[law]	yers	and	so	on	.

(b) Generated actions and translations via Multi-path agent

𝔑	wir	mussten	uns	wegen	der		[an]		[wäl]		te		[us]		w.		sorgen		machen			.			
𝔚						we		had		to		look		forward		because		of			the		[law]	yers	...

Table 5.5: The performance comparison of generated actions from our model (a) and multi-path model (b) on the generated and reference translations. Each column shows a single READ (𝔑) or WRITE (𝔚). Subword tokens ending in @@ are shown inside brackets.

5.3.5 Qualitative Analysis

Waiting for long consecutive words in the source is not always a good strategy in translating from SOV to SVO languages. Table 5.4 shows an example where our policy can generate each word as soon as it receives them. Such examples explain why our model performs better than *wait-k* models which are forced to wait for *k* words even when there is sufficient information to start the translation.

Table 5.5 compares the output of our model for an example German sentence to that of the multi-path model decoded with eval-wait-5 policy. As we mentioned earlier, our model does not wait too long to translate the tokens for which it has enough information. Please compare the translation time of "`wir`" and "`mussten`" in Table 5.5.(a) and Table 5.5.(b).

It is apparent that eval-wait-5 must wait for 5 tokens to translate the tokens that it has already had enough information about them for a while.

In addition, our model does not translate tokens based on immature information. For example, in table 5.5, our model waits until it receives "`sorgen machen`" which is essential for translating "`to worry`" as opposed to the eval-wait-5 design which forces the model to produce the generic verb "`to look forward`" which causes the model to lose information about the verb. While forcing this step lowers the latency, the translated sentence becomes highly inaccurate.

5.4 Summary

We present a novel idea for generating optimal segments in simultaneous translation by comparing the output of a simultaneous system for translation with an offline translation model. This provides us with oracle action sequences that we can use to train an Agent used to produce read and write actions for simultaneous translation. Our experimental results show that by using an offline translation component, our agent can generate better policies in terms of translation quality and delay compared to our baselines. We also show that our agent can be trained by previously proposed translation components and generate better policies compared to what they have reported before.

Chapter 6

Effectively pretraining a speech translation decoder with Machine Translation data

Automatic Speech Translation (AST) aims to directly translate audio signals in the source language into the text words in the target language. For many years, the pipeline of transcribing speech with ASR and then translating with the MT component was a standard method to address the speech translation problem. Having access to lots of data in many language pairs, the cascaded model for speech translation can benefit from well-trained ASR and MT components and generate high-quality translations.

In recent years, it has shown that we can remove the transcription step and build an end-to-end model that is strong enough to compete with the cascaded model [79]. Such models not only have lower inference latency, but they also do not suffer from the problem of errors that propagate from one component to the next. However, the scarcity of available resources is the main challenge in this task, and a variety of methods are proposed to address this problem. One of the most effective approaches to increase the performance of AST systems is to pretrain the encoder using an ASR model [13]. While pretraining the encoder by an ASR model even in different languages shows promising results [14], using a pretrained MT decoder is not beneficial [16, 13] or slightly improve the result [90] and even in some cases may worsen the results [10].

One explanation for this phenomenon is that the decoder works well only if its input comes from an encoder that it was trained with [54]. To solve the problem of invariant encoder representations, we make use of an adversarial regularizer in our loss function to bring the output of the ASR encoder closer to the input of MT decoder. We show that this modification can improve the BLEU score by +2.0 BLEU points.

6.1 Models

Similar to conventional MT models, the speech translation task generates translated words in the target language, representing as $\mathrm{Y} = (y_1, \ldots, y_J)$, given the sequence of source speech features $\mathrm{X} = (x_1, \ldots, x_I)$. The translation model then minimizes the Cross-Entropy loss $\mathrm{L}_{CE} = \Delta(\mathrm{Y}, \mathrm{Y}^*)$, where Y^* is the reference translation, and Δ is the sum of character-level Cross-Entropy loss.

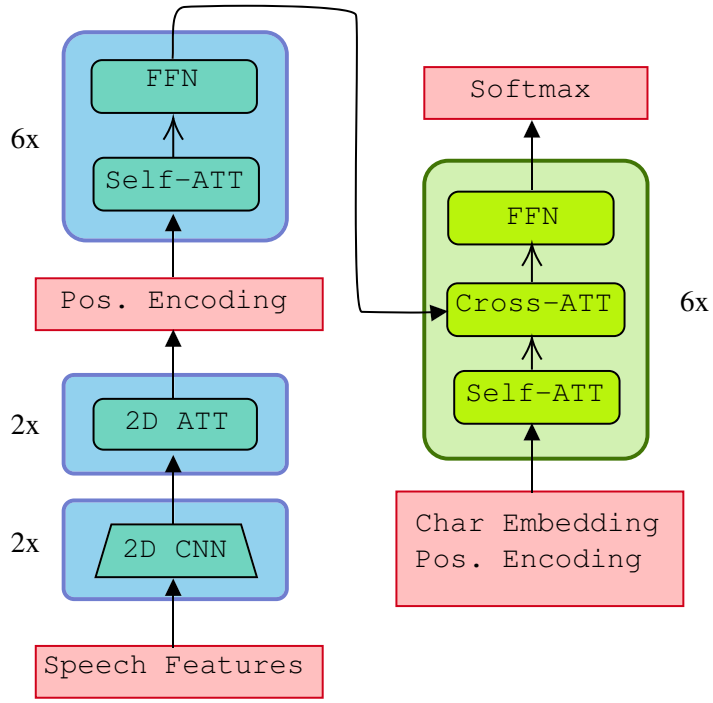


Figure 6.1: The illustration of our AST model with S-transformer [37]. We process the speech signals in two dimensions and combine them with positional encoding before forwarding them to the standard transformer encoder.

We use character-level encoding and decoding using Transformer [96] as the basic architecture of all our models. Our machine translation component follows the architecture of standard transformer model proposed Vaswani et al. [96].

6.1.1 End-to-End Speech Translation

An illustration of our architecture for the AST and ASR components is depicted in figure 6.1. We use similar architecture to [31] with an S-Transformer proposed by Di Gangi et al. [37]. The main difference between transformer and S-Transformer is the way it encodes the input features. S-Transformer encodes the audio features by passing them into two stacked layers of Convolutional Neural Nets (CNN) to extract local 2D-invariant features of the input. Then, it uses a 2D self-attention layer to compute the attention matrix using the second CNN's output.

Aside from the feature extraction, S-Transformer motivate the encoder's self attention to attend to short-range dependencies. Given the input sequence X, the standard transformer model [96] maps the input into three vectors Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$), and computes attention context vectors for each head $\mathbf{Z}$ as a weighted average of the values $\mathbf{V}$:

$$\mathbf{Q} = \mathrm{X}W^{\mathbf{Q}}, \quad \mathbf{K} = \mathrm{X}W^{\mathbf{K}}, \quad \mathbf{V} = \mathrm{X}W^{\mathbf{V}},$$

$$\mathbf{Z} = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d_{\text{model}}}).\mathbf{V}$$

Where d_{model} is a constant scaling factor. The S-Transformer alter this formula to penalize long-distance relationships:

$$\mathbf{Z} = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d_{\text{model}}} - \pi(\mathbf{D})).\mathbf{V}$$

Where $\mathbf{D}$ is a matrix containing distances between each two positions $d_{i,j} = |i - j|$, and π is a function that penalizes the long distance. Di gangi et al. [37] introduce a new logarithmic function as a distance penalty:

$$\pi_{\log}(d) = \begin{cases} 0 & \text{if } d = 0 \\ \ln(d) & \text{otherwise} \end{cases}$$

This function highlights the information around each position and as slowly fade the information in distant positions.

6.1.2 Aligning Encoder Representations

The conventional method for training an AST model is to pretrain ASR and NMT models separately and then transferring parameters of the encoder from ASR and the decoder from MT to the AST model, before starting to train via speech translation data.

Since we are training the encoder representations of the ASR model and the decoder parameters of the NMT system to work with their own encoder and decoder, pretraining the parameters of the AST model with a speech encoder from ASR and a text decoder from NMT is not ideal. Therefore, we propose to use adversarial training to bring NMT encoder and ASR encoder representations closer together.

An overview of our model is depicted in Figure 6.2. Instead of separately pretraining the ASR and NMT, we propose to update their parameters simultaneously. In order to add explicit incentives to learn multi-modal representations in the encoder, we will train our NMT and ASR models on both Cross-Entropy loss and a new regularization loss. The final training objective for each task can be formulated as:

$$\text{Loss} = \text{L}_{CE} + \alpha\ \text{L}_{DISC}$$

where L_{CE} is the Cross-Entropy loss, L_{DISC} is the newly added regularization term, and α is the constant parameter to control the effect of our regularizer. Since L_{DISC} is a smaller number compared to L_{CE}, we set α to 5 in all our experiments to make the regularizer loss more perceptible during backward propagation. We are also sharing the parameters of the transformer layers in the encoder between AST and MT models. In the following section, we describe the regularizer.

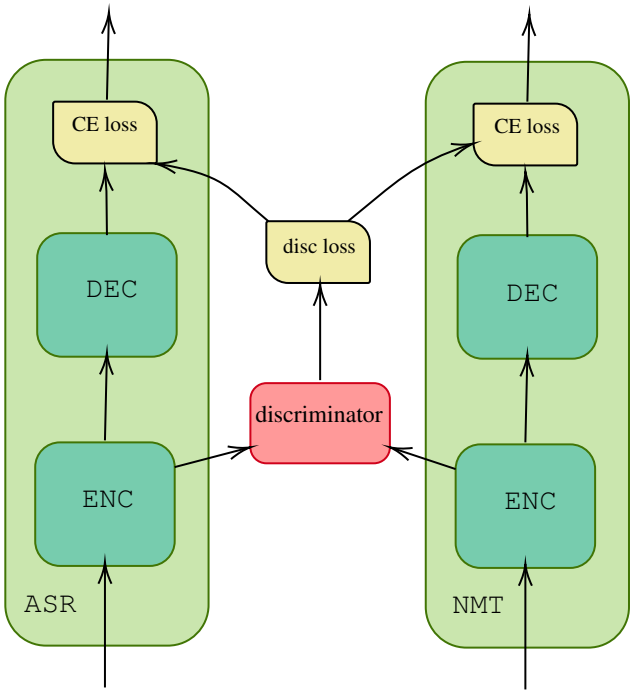


Figure 6.2: The proposed pretraining method using an adversarial loss.

6.1.3 Adversarial Regularizer

Given the embeddings of inputs x_i in each modalities (speech features for ASR or character embeddings for NMT), the encoder computes the encoder representations Z_{x_i}. By passing Z_{x_i} to the discriminator, we can train its network by minimizing the loss function $\text{Loss}_D = -\mathbb{E}_{x_i}[\log P_D(\mathcal{M}|Z_{x_i})]$, where $\mathcal{M}$ is the modality of x_i, with $\mathcal{M} \in \{\text{ASR}, \text{NMT}\}$, and P_D is the probability of choosing the right modality given the output of encoder.

The encoder of NMT or ASR will be trained in order to deceive the discriminator by minimizing the loss:

$$\text{L}_{DISC} = -\mathbb{E}_{x_i}[\log P_D(-\mathcal{M}|Z_{x_i})]$$

where $-\mathcal{M} = \text{ASR}$ if $\mathcal{M} = \text{NMT}$ and vice versa. The final expanded Loss function for each modalities can be formulated as:

$$\text{Loss} = -\mathbb{E}_{x_i}[\log P_{\mathcal{M}}(y_i|y_{<i}, X)] + \alpha\ (-\mathbb{E}_{x_i}[\log P_D(\mathcal{M}|Z_{x_i})])$$

Where y_i is the ground truth label for the input x_i, and $P_{\mathcal{M}}$ is the probability of choosing the right label by modality $\mathcal{M}$. By incorporating this regularizer, we ensure that the encoder

representations from different modalities (speech and text) become indistinguishable during training.

Our discriminator consists of a three-layer feed-forward network with 1024 hidden units, followed by a Leaky-ReLU activation function [54].

6.2 Experiments

6.2.1 Dataset

To evaluate our AST systems, we conducted our experiments on two datasets. For the English-German language pair, we use the MuST-C corpus [30], which consists of 408 hours of speech data aligned with 234K translated sentences. For the English-French language pair, we use the full training set of Translation Augmented Librispeech (Libri-Trans) corpus [52] with 230 hours of speech aligned with 131K french sentences.

We use LibriSpeech corpus [74] with 960h of English speeches in order to train our ASR system. Since the test and dev sets of Libri-Trans corpus is part of the ASR LibriSpeech dataset, we remove all utterances from ASR LibriSpeech that share the same (chapter-id, reader-id) pairs with the test and dev sets in the Libri-Trans corpus. For En-De MT training, we use the combination of TED and Opensubtitle2018 corpora [1] [2] which contains more than 18M sentences pairs after filtering noisy pairs. The MT training of the English-French language pair uses the En-Fr portion of the WMT14 competition [18].

6.2.2 Preprocessing and Evaluation

For each speech utterance, we extract 40 Mel-filterbank energy features with a step size of 10 ms and a window size of 25ms. For features extracted from MuSt-C and ASR LibriSpeech, we apply mean and variance normalization for each speaker.

We keep all the texts in our experiments true-case and tokenize them using Moses tokenizer[3]. We remove the punctuation from all English texts (both from the target side of ASR and the source side of MT).

For translation tasks (AST and MT), we report BLEU score [75] on tokenized sentences[4]. We evaluate our ASR systems using Word Error Rate (WER)[5].

6.2.3 Model Settings

For both En-De and En-Fr tasks, we followed the architecture in [31]. We use six Transformer layers of size 512 in the encoder and decoder with eight attention heads. The size of feed-

[1] http://www.opensubtitles.org/

[2] http://opus.nlpl.eu/OpenSubtitles-v2018.php

[3] http://www.statmt.org/moses/

[4] https://www.nltk.org/_modules/nltk/translate/bleu_score.html

[5] https://github.com/belambert/asr-evaluation

	En-De		En-Fr	
	#param	#hours	#param	#hours
cascaded NMT	45M	27	45M	13
cascaded ASR	31M	22.5	31M	18.2
AST	31M	34	31M	18

Table 6.1: The number of parameters and run-time of our models on MuSt-C dataset (En-De) and Libri-Trans dataset (En-Fr).

forward mechanism is 1024. The embedding layer in the encoder for the AST task contains two layers of 2D CNNs [55] followed by a ReLU activation function. Each CNN layer has 16 output channels, with a stride of (2, 2). We run all our models on two GeForce GTX 1080 GPUs with 12GB RAM each. The total number of parameters and run-time of our models in Table 6.1.

6.2.4 Training Settings

In all our models, we use the Adam optimizer [51] with an initial learning rate of 0.00005. During the first 6000 warm-up updates, we increase it linearly to 0.003, then decrease it with inverse square root decay [96]. The number of warm-up updates in our MT systems is 8000. The hyper-pararmeters of the ASR, NMT and Discriminator components are summarized in table 6.2.

6.3 Results

In this section, we analyze the effect of our regularizer on two different settings: (A) When we only have access to AST data (section 6.3.1) and (B) When we can benefit from External data (section 6.3.2). For each setting, we run experiments on four different models:

1. The cascaded model
2. AST model with pretrained ASR encoder
3. AST model with pretrained ASR encoder and MT decoder
4. Our proposed model with adversarial loss.

6.3.1 Using Only AST Data

Table 6.3 shows the performance of AST models for En-De and En-Fr language pairs. When the cascaded model is restricted to use small AST datasets merely, the model will not be strong enough to beat an AST model with a pretrained encoder and decoder. We should also note that unlike [14, 10], where transferring decoder parameters were not effective, in all our AST models, we could only beat the cascaded model by pretraining the decoder.

Hyperparameter	Model	Value
dropout		0.1
warmup update	A,D	6000
warmup update	N	8000
warmup init lr		5e-5
criterion		cross entropy
LR	D	1e-4
LR	A,N	3e-3
LR scheduler		inverse sqrt
optimizer		Adam
clip norm		20
num layers	D	3
hidden dim	D	128
encoder embed dim	A,N	512
encoder convolutions	A	[(64, 3, 3)] * 2
encoder layers	A,N	6
encoder ffn embed dim	A,N	1024
encoder attention heads	A,N	8
decoder embed dim	A,N	512
decoder layers	A,N	6
decoder out embed dim	A,N	512
decoder output dim	A,N	512
decoder ffn embed dim	A,N	1024
decoder attention heads	A,N	8

Table 6.2: The hyper-parameters of the ASR (A), NMT (N), and Discriminator (D). If the name of the model (A, M, or D) is not specified in front of the parameter, then it will be used in all three models.

The last row in the table gives the AST model results, which uses adversarial regularizer during the pretrain step. As we can see, training the NMT and the ASR models simultaneously can help pretrained components be compatible with each other and improve the final performance by 1.2 and 1.7 BLEU scores for En-De and En-Fr language pairs respectively.

6.3.2 Using Both AST and External Data

Limiting the training data for the speech translation models to AST datasets is not a realistic assumption for many language pairs, and in practice, the cascaded model can greatly benefit from the large amounts of NMT and ASR corpora.

Table 6.4 summarizes the effects of adding external training data to our experiments. Adding external data can boost the performance of the cascaded model and by comparing Table 6.3 and 6.4, we can see that the additional NMT and ASR data can improve the translation quality of the cascaded model by +2 BLEU scores, while it can barely affect the AST model with pretrained encoder and the decoder. Consequently, the gap between the

Task	En-De	En-Fr
cascaded	18.76	15
AST + ASR pre	18.71	14.7
AST + ASR pre + MT pre	19.05	15.3
AST + regularizer	20.24	17.01

Table 6.3: Results of AST models trained only with AST data. The performance is measured with BLEU score on MuST-C test set.

Task	En-De	En-Fr
cascaded	21.06	19.21
AST + ASR pre	19.01	16.13
AST + ASR pre + MT pre	19.12	16.27
AST + regularizer	20.81	17.7

Table 6.4: BLEU scores of AST models, trained with both AST and external ASR and MT data.

AST model and the cascaded system increases by around +3 BLEU scores for En-Fr and +2 BLEU scores for the En-De language pair.

As we can see in the last row of Table 6.4, adding our proposed pretraining step can help the model perform better during training, and compared to the conventional pretraining step, we can see an increase of more than 1 BLEU point in each language pair. Although the cascaded model by having access to all the pretrained parameters (the encoder and decoder of both NMT and ASR) still has better translation quality, we can bring the performance of an end-to-end model closer to it by adding the new regularizer. It is also important to note that since we are not changing the final structure of the AST model, most of the other techniques for further improving the translation quality, such as data augmentation, which was examined in previous studies [65, 76] can also be applied. But we won't study them in this chapter.

6.4 Related Work

The cascaded pipeline of transcribing speech signals and then translating them using an MT component [69, 23] was for many years the standard design of speech translation systems [47]. The idea of having an end-to-end structure for this task showed promising results in the works of [1, 33, 17, 4, 2, 15]. After the success of [98] in creating a powerful model for ST systems, more recent studies focused on exploring their power, and one of the main approaches to boost the performance of such models is to make use of available data from other tasks, such as ASR and NMT. [98, 3, 90] show that multitask learning can be effective and [49, 79, 76, 65] investigate various data augmentation techniques. The impact of pretraining the encoder with ASR model is also studied in [16, 13, 14]. In experiments of [10, 14] the performance gain of pretraining the decoder with an MT model was marginal.

[50] addresses the ASR encoder and MT decoder gap problem by proposing a "Transcoder" and use smooth-L1 loss to bring ASR hidden representation close to MT encoder hidden representation.

The idea of modifying loss function in AST models was also discussed in [90]. Their formulation of the additional loss is different from ours, and they use their additional loss function in a different NMT architecture from ours.

The idea of adding adversarial regularizer was discussed in other tasks such as unsupervised MT [54] or zero-shot translation [78]. The closest research to our work is [5], which uses a similar adversarial network to bring encoder representations closer together. However, they apply their model to the zero-shot machine translation task, with a different architecture. They also apply their regularizer to the representations of the different languages with the same modalities.

6.5 Summary

In this chapter, we studied the impact of pretraining the decoder of an end-to-end speech translation system using a machine translation model and proposed a method to make the pretraining step more effective. We showed that we can align the latent representations of speech and text modalities by using adversarial loss and make the ASR encoder more compatible with the MT decoder. Our experimental results demonstrate that we can improve the performance by around 1.5 BLEU points on two language pairs compared to conventional pretraining methods.

Chapter 7

Conclusion

In this thesis, we have presented three contributions to the simultaneous translation task. Although the projects are distinct, their final goal, which is to create a simultaneous translation model as quickly as possible, links them to each other. We will summarize this thesis in the following three aspects:

- In our first project, we have investigated the effects of adding prediction mechanism to the simultaneous translation task. We added a new Prediction action to the set of READ and WRITE actions in a Reinforcement Learning framework. We have also introduced a new reward function for the training process to motivate the agent to choose the prediction action. Our experiments with our new agent show an improvement of around 1 BLEU point in the translation quality with lower delay. The effect of the prediction action is noticeable when the length of the sentences gets longer due to complex reordering and multi-clausal sentences.

- We have also proposed a novel supervised learning approach for training an agent that can detect the minimum number of reads required for generating each target token by comparing simultaneous translations against full-sentence translations during training to generate oracle action sequences. These oracle sequences can then be used to train a supervised model for action generation at inference time. Our approach provides an alternative to current heuristic methods in simultaneous translation by introducing a new training objective, which is easier to train than previous attempts at training the agent using reinforcement learning techniques for this task. Our experiments showed the translation quality of our oracle policy is around 7 BLEU points higher than our baseline when we use an offline translation component on two different language pairs. We demonstrated that using our oracle as a training objective for the agent can result in policies that are adaptive to the source and target sentence structures with noticeably higher BLEU scores and lower delay. In experiments with our trained agent, we generated oracles for translation component that was trained to work with wait-k policy, which then we use those oracles for training a new agent. These experiments showed our agent could generate more accurate translations with less latency. This implies that previously proposed methods for improving the performance of transla-

tion components in simultaneous translation settings are more accurate and agile than what was reported before if we use a more suitable policy.

- Our final project investigates a method for improving the performance of end-to-end speech translation models via available datasets for machine translation task. The main contribution of this project is introducing a new adversarial regularizer that allows us to pretrain ASR and MT models simultaneously and bring their encoder representations closer together. This leads to having ASR encoder representations that are more compatible with MT decoder. Our experimental results indicate that when we restricted to only use available data for the speech translation task, using adversarial regularizer during pretraining can improve the translation quality by around 1.5 BLEU scores on two language pairs. While adding external data from MT and ASR tasks can boost the performance of our solid cascaded baseline by +2 BLEU scores, using the adversarial loss in pretraining step can improve the translation accuracy of our system by +1 BLEU points and decrease the gap between the cascaded model and the end-to-end model. This new adversarial loss will only be applied during pretraining step, and the final structure of an end-to-end speech translation system remains unchanged. This indicates that all the techniques previously introduced to segment the speech signals for the simultaneous task can also be applied to this new speech translation system.

7.1 Future Direction

In this section, we explain four ideas that have the potential to improve the performance of simultaneous translation systems:

- **Applying prediction to other structures:** Having a simple and stable training process is essential if we want to extend the model with the prediction mechanism effectively. Recently there has been much progress in designing accurate agents with less complicated structures compared to reinforcement learning [101, 35, 6]. Each of these models can be extended to anticipate future time steps. Especially extending the supervised learning method which we studied recently (chapter 5) should be straightforward and can help the model to make more precise decisions on longer sentences.

- **Extending supervised learning method:** We have thoroughly explored the behaviour of our agent using two different translation components. It would be interesting to see whether filling up the partial translations table using recently proposed interpreters, like those described in [9, 6] or [61] can lead to generating more optimal policies or not. Similar to the idea in [59], we can retrain the translation component to become more accurate in translating phrases generated by our supervised agent. In order to build such an interpreter, we are thinking of extracting phrase pairs that our agent aligns and use them to fine-tune a translation model trained on full sentences.

Analysing the distinction between source-side and target-side prediction can also help us have better understanding of our current models. Knowing that a prediction comes from the source sentence or the target sentence can help us emphasize each of them to further improve the results.

- **Finding an alternative to the loss value for reporting the results:** While the cross-entropy loss is an effective metric for guiding the model toward oracle policy in a supervised setting, it is not a good criterion for choosing the optimal policy that can balance the trade-off between translation quality and delay. In other words, a model with a low loss can generate accurate translations by sacrificing delay. Finding a metric that can measure both translation quality and delay can be helpful in finding the optimal policy during training. Although the Hamming [42] and Levenshtein [56] distances between generated and reference action sequences are more effective compared to loss, a more accurate metric is required to measure how well our model can balance the translation quality and delay.

- **Designing a new evaluation metric:** The available datasets for the simultaneous translation task is coming from the offline task and the translations are not suited for the real-time environment. Making use of an experienced human interpreter can help us evaluate our systems more accurately. Recording the timing of such interpreter can help us measure our metrics for the delay, or even lead us to have a unified metric for translation quality and delay.

- **Applying speech segmentation:** The improvement we achieved to train speech translation model more accurately makes our trained end-to-end model a safe choice to be used in simultaneous settings. However, it is not clear how the generated policies would change if we replace the standard speech translation system with our speech translation model.

Bibliography

[1] Oliver Adams, Graham Neubig, Trevor Cohn, Steven Bird, Quoc Truong Do, and Satoshi Nakamura. Learning a lexicon and translation model from phoneme lattices. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2377–2382, Austin, Texas, November 2016. Association for Computational Linguistics.

[2] Antonios Anastasopoulos and David Chiang. A case study on using speech-to-translation alignments for language documentation. In *Proceedings of the 2nd Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 170–178, Honolulu, March 2017. Association for Computational Linguistics.

[3] Antonios Anastasopoulos and David Chiang. Tied multitask learning for neural speech translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 82–91, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[4] Antonios Anastasopoulos, David Chiang, and Long Duong. An unsupervised probability model for speech-to-translation alignment of low-resource languages. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1255–1263, Austin, Texas, November 2016. Association for Computational Linguistics.

[5] Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Roee Aharoni, Melvin Johnson, and Wolfgang Macherey. The missing ingredient in zero-shot neural machine translation. *ArXiv*, abs/1903.07091, 2019.

[6] Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. Monotonic infinite lookback attention for simultaneous machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1313–1323, Florence, Italy, July 2019. Association for Computational Linguistics.

[7] Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, and George Foster. Re-translation versus streaming for simultaneous translation. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 220–227, Online, July 2020. Association for Computational Linguistics.

[8] Naveen Arivazhagan, Colin Cherry, I Te, Wolfgang Macherey, Pallavi Baljekar, and George Foster. Re-translation strategies for long form, simultaneous, spoken language translation. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7919–7923, 2020.

[9] Philip Arthur, Trevor Cohn, and Gholamreza Haffari. Learning coupled policies for simultaneous machine translation using imitation learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2709–2719, Online, April 2021. Association for Computational Linguistics.

[10] Parnia Bahar, Tobias Bieschke, and Hermann Ney. A comparative study on end-to-end speech to text translation. *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 792–799, 2019.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[12] Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour, and Aura Jimenez. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 437–445, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[13] Sameer Bansal, Herman Kamper, Karen Livescu, Adam Lopez, and Sharon Goldwater. Low-resource speech-to-text translation. *CoRR*, abs/1803.09164, 2018.

[14] Sameer Bansal, Herman Kamper, Karen Livescu, Adam Lopez, and Sharon Goldwater. Pre-training on high-resource speech recognition improves low-resource speech-to-text translation. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 58–68. Association for Computational Linguistics, 2019.

[15] Sameer Bansal, Herman Kamper, Adam Lopez, and Sharon Goldwater. Towards speech-to-text translation without speech recognition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 474–479, Valencia, Spain, April 2017. Association for Computational Linguistics.

[16] Alexandre Berard, Laurent Besacier, Ali Kocabiyikoglu, and Olivier Pietquin. End-to-end automatic speech translation of audiobooks. pages 6224–6228, 04 2018.

[17] Alexandre Bérard, Olivier Pietquin, Laurent Besacier, and Christophe Servan. Listen and Translate: A Proof of Concept for End-to-End Speech-to-Text Translation. In *NIPS Workshop on end-to-end learning for speech and audio processing*, Barcelona, Spain, December 2016.

[18] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

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

[20] Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL 2014, June 26-27, 2014, Baltimore, Maryland, USA*, pages 362–367, 2014.

[21] Colin Cherry and George F. Foster. Thinking slow about latency evaluation for simultaneous machine translation. *ArXiv*, abs/1906.00048, 2019.

[22] Chung-Cheng Chiu and Colin Raffel. Monotonic chunkwise attention. In *International Conference on Learning Representations*, 2018.

[23] Eunah Cho, Jan Niehues, and Alex Waibel. Nmt-based segmentation and punctuation insertion for real-time spoken language translation. In *Proc. Interspeech 2017*, pages 2645–2649, 2017.

[24] Kyunghyun Cho and Masha Esipova. Can neural machine translation do simultaneous translation? *CoRR*, abs/1606.02012, 2016.

[25] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[26] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[27] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[28] Lonnie Chrisman. Learning recursive distributed representations for holistic computation. *Connection Science*, 3(4):345–366, 1991.

[29] 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.

[30] Mattia A. Di Gangi, Roldano Cattoni, Luisa Bentivogli, Matteo Negri, and Marco Turchi. MuST-C: a Multilingual Speech Translation Corpus. 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 2012–2017, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[31] Mattia A. Di Gangi, Matteo Negri, Viet Nhat Nguyen, Amirhossein Tebbifakhr, and Marco Turchi. Data augmentation for end-to-end speech translation: Fbk@iwslt '19. 2019.

[32] Paul R. Dixon, A. Finch, Chiori Hori, and H. Kashioka. Investigation on the effects of asr tuning on speech translation performance. In *IWSLT*, 2011.

[33] Long Duong, Antonios Anastasopoulos, David Chiang, Steven Bird, and Trevor Cohn. An attentional model for speech translation without transcription. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 949–959, San Diego, California, June 2016. Association for Computational Linguistics.

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

[35] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Efficient Wait-k Models for Simultaneous Machine Translation. In *Interspeech 2020 - Conference of the International Speech Communication Association*, pages 1461–1465, Shangai (Virtual Conf), China, October 2020.

[36] Christian Fügen, Alex Waibel, and Muntsin Kolss. Simultaneous translation of lectures and speeches. *Machine Translation*, 21(4):209–252, Dec 2007.

[37] Mattia A. Di Gangi, Matteo Negri, and Marco Turchi. Adapting Transformer to End-to-End Spoken Language Translation. In *Proc. Interspeech 2019*, pages 1133–1137, 2019.

[38] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery.

[39] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[40] 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 *Empirical Methods in Natural Language Processing*, 2014.

[41] Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. Learning to translate in real-time with neural machine translation. In *15th Conference of the European*

Chapter of the Association for Computational Linguistics (EACL), Valencia, Spain, April 2017.

[42] Richard W. Hamming. *Coding and Information Theory (2nd Ed.)*. Prentice-Hall, Inc., USA, 1986.

[43] He He, Alvin Grissom II, John Morgan, Jordan Boyd-Graber, and Hal Daumé III. Syntax-based rewriting for simultaneous machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 55–64, 2015.

[44] Xiaodong He, Li Deng, and Alex Acero. Why word error rate is not a good metric for speech recognizer training for the speech translation task? In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5632–5635, 2011.

[45] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[46] Eduard Hovy. Toward finely differentiated evaluation metrics for machine translation. In *Proceedings of the EAGLES Workshop on Standards and Evaluation*, pages 127–133, 1999.

[47] Hirofumi Inaguma, Kevin Duh, Tatsuya Kawahara, and Shinji Watanabe. Multilingual end-to-end speech translation. *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 570–577, 2019.

[48] Navdeep Jaitly, Quoc V Le, Oriol Vinyals, Ilya Sutskever, David Sussillo, and Samy Bengio. An online sequence-to-sequence model using partial conditioning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[49] Ye Jia, Melvin Johnson, Wolfgang Macherey, Ron Weiss, Yuan Cao, Chung-Cheng Chiu, Naveen Ari, Stella Laurenzo, and Yonghui Wu. Leveraging weakly supervised data to improve end-to-end speech-to-text translation. pages 7180–7184, 05 2019.

[50] Takatomo Kano, Sakriani Sakti, and Satoshi Nakamura. End-to-end speech translation with transcoding by multi-task learning for distant language pairs. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 28:1342–1355, January 2020.

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

[52] Ali Can Kocabiyikoglu, Laurent Besacier, and Olivier Kraif. Augmenting librispeech with French translations: A multimodal corpus for direct speech translation evaluation. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[53] Lars Konieczny and Philip Döring. Anticipation of clause-final heads. evidence from eye-tracking and srns. *Proceedings of the 4th International Conference on Cognitive Science*, pages 330–335, 2003.

[54] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations*, 2018.

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[56] Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[57] Chin-Yew Lin and Franz Josef Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[58] Minh-Thang Luong, Christopher D Manning, et al. Stanford neural machine translation systems for spoken language domains. In *Proceedings of the international workshop on spoken language translation*, pages 76–79, Da Nang, Vietnam, 2015.

[59] 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. pages 3025–3036, July 2019.

[60] Xutai Ma, Juan Pino, and Philipp Koehn. SimulMT to SimulST: Adapting simultaneous text translation to end-to-end simultaneous speech translation. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 582–587, Suzhou, China, December 2020. Association for Computational Linguistics.

[61] Xutai Ma, Juan Miguel Pino, James Cross, Liezl Puzon, and Jiatao Gu. Monotonic multihead attention. In *International Conference on Learning Representations*, 2020.

[62] Xutai Ma, Yongqiang Wang, Mohammad Javad Dousti, Philipp Koehn, and Juan Pino. Streaming simultaneous speech translation with augmented memory transformer. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7523–7527, 2021.

[63] Shigeki Matsubara, Keiichi Iwashima, Nobuo Kawaguchi, Katsuhiko Toyama, and Yasuyoshi Inagaki. Simultaneous japanese-english interpretation based on early prediction of english verb. *Proceedings of the 4th Symposium on Natural Languauge Processing(SNLP-2000)*, pages 268–273, 5 2000.

[64] Evgeny Matusov, Patrick Wilken, and Yota Georgakopoulou. Customizing neural machine translation for subtitling. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 82–93, Florence, Italy, August 2019. Association for Computational Linguistics.

[65] A. D. McCarthy, L. Puzon, and J. Pino. Skinaugment: Auto-encoding speaker conversions for automatic speech translation. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7924–7928, 2020.

[66] A. McNair, A. Jain, A. Waibel, J. Tebelskis, A. Hauptmann, and H. Saito. Janus: a speech-to-speech translation system using connectionist and symbolic processing strategies. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, pages 793–796, Los Alamitos, CA, USA, apr 1991. IEEE Computer Society.

[67] Takashi Mieno, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Speed or accuracy? a study in evaluation of simultaneous speech translation. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529 EP –, 02 2015.

[69] H. Ney. Speech translation: coupling of recognition and translation. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 1, pages 517–520 vol.1, 1999.

[70] Jan Niehues, Thai Son Nguyen, Eunah Cho, Thanh-Le Ha, Kevin Kilgour, Markus Müller, Matthias Sperber, Sebastian Stüker, and Alex Waibel. Dynamic transcription for low-latency speech translation. In *Interspeech 2016*, pages 2513–2517, 2016.

[71] Jan Niehues, Ngoc-Quan Pham, Thanh-Le Ha, Matthias Sperber, and Alex Waibel. Low-latency neural speech translation. In *Proc. Interspeech 2018*, pages 1293–1297, 2018.

[72] 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)*, pages 551–556, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[73] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[74] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[75] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[76] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. In *INTERSPEECH*, 2019.

[77] Stephan Peitz, Simon Wiesler, M. Nußbaum-Thom, and H. Ney. Spoken language translation using automatically transcribed text in training. In *IWSLT*, 2012.

[78] Ngoc-Quan Pham, Jan Niehues, Thanh-Le Ha, and Alex Waibel. Improving zero-shot translation with language-independent constraints. In *Proceedings of the Forth Conference on Statistical Machine Translation (WMT 2019)*, 2019.

[79] Juan Pino, Liezl Puzon, Jiatao Gu, Xutai Ma, Arya D. McCarthy, and Deepak Gopinath. Harnessing indirect training data for end-to-end automatic speech translation: Tricks of the trade. 2019.

[80] Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. Online and linear-time attention by enforcing monotonic alignments. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2837–2846. PMLR, 06–11 Aug 2017.

[81] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[82] Kanishka Rao, Haşim Sak, and Rohit Prabhavalkar. Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 193–199, 2017.

[83] Yi Ren, Jinglin Liu, Xu Tan, Chen Zhang, Tao Qin, Zhou Zhao, and Tie-Yan Liu. SimulSpeech: End-to-end simultaneous speech to text translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3787–3796, Online, July 2020. Association for Computational Linguistics.

[84] Debra L Russell. *Interpreting in legal contexts: Consecutive and simultaneous interpretation*. Calgary, 2000.

[85] H. Sak, M. Shannon, Kanishka Rao, and F. Beaufays. Recurrent neural aligner: An encoder-decoder neural network model for sequence to sequence mapping. In *INTERSPEECH*, 2017.

[86] Harsh Satija and Joelle Pineau. Simultaneous machine translation using deep reinforcement learning. *Abstraction in Reinforcement Learning Workshop, ICML 2016*, (33), June 2016.

[87] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11):2673–2681, November 1997.

[88] 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, Berlin, Germany, August 2016. Association for Computational Linguistics.

[89] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783, 2018.

[90] Matthias Sperber, Graham Neubig, Jan Niehues, and Alex Waibel. Attention-passing models for robust and data-efficient end-to-end speech translation. *Transactions of the Association for Computational Linguistics*, 7:313–325, March 2019.

[91] Vivek Kumar Rangarajan Sridhar, John Chen, Srinivas Bangalore, Andrej Ljolje, and Rathinavelu Chengalvarayan. Segmentation strategies for streaming speech translation. In *HLT-NAACL*, pages 230–238, 2013.

[92] F. W. M. Stentiford and M. G. Steer. *Machine Translation of Speech*, page 183–196. Chapman & Hall, Ltd., GBR, 1990.

[93] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[94] T. Takezawa, T. Morimoto, Y. Sagisaka, N. Campbell, H. Iida, F. Sugaya, Akio Yokoo, and S. Yamamoto. A japanese-to-english speech translation system: Atr-matrix. In *ICSLP*, 1998.

[95] Yulia Tsvetkov, Florian Metze, and Chris Dyer. Augmenting translation models with simulated acoustic confusions for improved spoken language translation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 616–625, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

[96] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[97] Alex Waibel and C. Fugen. Spoken language translation. *Signal Processing Magazine, IEEE*, 25:70 – 79, 06 2008.

[98] Ron J. Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. Sequence-to-sequence models can directly translate foreign speech. In *INTERSPEECH*, 2017.

[99] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

[100] Chunyang Wu, Yongqiang Wang, Yangyang Shi, Ching-Feng Yeh, and Frank Zhang. Streaming Transformer-Based Acoustic Models Using Self-Attention with Augmented Memory. In *Proc. Interspeech 2020*, pages 2132–2136, 2020.

[101] 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 the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1349–1354, Hong Kong, China, November 2019. Association for Computational Linguistics.

[102] 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, Florence, Italy, July 2019. Association for Computational Linguistics.