SIMULEVAL : An Evaluation Toolkit for Simultaneous Translation

Xutai Ma [1,2], Mohammad Javad Dousti[1], Changhan Wang[1], Jiatao Gu[1], Juan Pino[1]

[1]Facebook AI
[2]Johns Hopkins University

xutai_ma@jhu.edu
{juancarabina,dousti,changhan,jgu}@fb.com

Abstract

Simultaneous translation on both text and speech focuses on a real-time and low-latency scenario where the model starts translating before reading the complete source input. Evaluating simultaneous translation models is more complex than offline models because the latency is another factor to consider in addition to translation quality. The research community, despite its growing focus on novel modeling approaches to simultaneous translation, currently lacks a universal evaluation procedure. Therefore, we present SIMULEVAL, an easy-to-use and general evaluation toolkit for both simultaneous text and speech translation. A server-client scheme is introduced to create a simultaneous translation scenario, where the server sends source input and receives predictions for evaluation and the client executes customized policies. Given a policy, it automatically performs simultaneous decoding and collectively reports several popular latency metrics. We also adapt latency metrics from text simultaneous translation to the speech task. Additionally, SIMULEVAL is equipped with a visualization interface to provide better understanding of the simultaneous decoding process of a system. SIMULEVAL has already been extensively used for the IWSLT 2020 shared task on simultaneous speech translation. Code will be released upon publication. [1]

1 Introduction

Simultaneous translation, the task of generating translations before reading the entire text or speech source input, has become an increasingly popular topic for both text and speech translation (Grissom II et al., 2014; Cho and Esipova, 2016; Gu et al., 2017; Alinejad et al., 2018; Arivazhagan et al., 2019; Zheng et al., 2019; Ma et al., 2020; Ren et al., 2020). Simultaneous models are typically evaluated from quality and latency perspective. Note that the term *latency* is overloaded and sometimes refers to the actual system speed. In this paper, *latency* refers to the simultaneous ability, which is how much partial source information is needed to start the translation process.

While the translation quality is usually measured by BLEU (Papineni et al., 2002; Post, 2018), a wide variety of latency measurements have been introduced, such as Average Proportion (AP) (Cho and Esipova, 2016), Continues Wait Length (CW) (Gu et al., 2017), Average Lagging (AL) (Ma et al., 2019), Differentiable Average Lagging (DAL) (Cherry and Foster, 2019), and so on. Unfortunately, the latency evaluation processes across different works are not consistent: 1) the latency metric definitions are not precise enough with respect to text segmentation; 2) the definitions are also not precise enough with respect to the speech segmentation, for example some models are evaluated on speech segments (Ren et al., 2020) while others are evaluated on time duration (Ansari et al., 2020); 3) little prior work has released implementations of the decoding process and latency measurement. The lack of clarity and consistency of the latency evaluation process makes it challenging to compare different works and prevents tracking the scientific progress of this field.

In order to provide researchers in the community with a standard, open and easy-to-use method to evaluate simultaneous speech and text translation systems, we introduce SIMULEVAL, an open source evaluation toolkit which automatically simulates a real-time scenario and evaluates both latency

[1]The code is available at https://github.com/facebookresearch/SimulEval

Proceedings of the 2020 EMNLP (Systems Demonstrations), pages 144–150
November 16-20, 2020.

and translation quality. The design of this toolkit follows a server-client scheme, which has the advantage of creating a fully simultaneous translation environment and is suitable for shared tasks such as the IWSLT 2020 shared task on simultaneous speech translation[2] or the 1st Workshop on Automatic Simultaneous Translation at ACL 2020[3]. The server provides source input (text or audio) upon request from the client, receives predictions from the client and returns different evaluation metrics when the translation process is complete. The client contains two components, an agent and a state, where the former executes the system's policy and the latter keeps track of information necessary to execute the policy as well as generating a translation. SIMULEVAL has built-in support for quality metrics such as BLEU (Papineni et al., 2002; Post, 2018), TER (Snover et al., 2006) and METEOR (Banerjee and Lavie, 2005), and latency metrics such as AP, AL and DAL. It also support customized evaluation functions. While all latency metrics have been defined for text translation, we discuss issues and solutions when adapting them to the task of simultaneous speech translation. Additionally, SIMULEVAL users can define their own customized metrics. SIMULEVAL also provides an interface to visualize the policy of the agent. An interactive visualization interface is implemented to illustrate the simultaneous decoding process. The initial version of SIMULEVAL was used to evaluate submissions from the first shared task on simultaneous speech translation at IWSLT 2020 (Ansari et al., 2020).

In the remainder of the paper, we first formally define the task of simultaneous translation. Next, latency metrics and their adaptation to the speech task are introduced. After that, we provide a high-level overview of the client-server design of SIMULEVAL. Finally, usage instructions and a case study are provided before concluding.

2 Task Formalization

An evaluation corpus for a translation task contains one or several instances, each of which consists of a source sequence $\boldsymbol{X} = [x_1, ..., x_{|\boldsymbol{X}|}]$ and a reference sequence $\boldsymbol{Y}^* = [y_1^*, ..., y_{|\boldsymbol{Y}|}^*]$. The system to be evaluated takes $\boldsymbol{X}$ as input, and generates $\boldsymbol{Y} = [y_1, ..., y_{|\boldsymbol{Y}|}]$. We denote the elements of the $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Y}^*$ segments. For text translation, each x_j is an individual word while for speech translation, x_j is a raw audio segment of duration T_j. In the simultaneous translation task, a system starts generating a hypothesis with partial input only. Then it either reads a new source segment, or writes a new target segment. Assuming $\boldsymbol{X}_{1:j} = [x_1, ..., x_j], j < |\boldsymbol{X}|$ has been read when generating y_i, we define the delay of y_i as

$$d_i = \begin{cases} j, & \text{when input is text} \\ \sum_{k=1}^{j} T_k, & \text{when input is speech} \end{cases} \quad (1)$$

Similar to an offline model, the quality is measured by comparing the hypothesis $\boldsymbol{Y}$ to the reference $\boldsymbol{Y}^*$ after the translation process is complete. On the other hand, the latency measurement involves considering partial hypotheses. The latency metrics are calculated from a function which takes a sequence of delays $\boldsymbol{D} = [d_1, ..., d_{|\boldsymbol{Y}|}]$ as input.

3 Latency Evaluation

3.1 Existing Text Latency Metrics

First, we review three latency metrics previously introduced for the text translation task.

Average Proportion (AP) (Cho and Esipova, 2016), defined in Eq. (2), measures the average of proportion of source input read when generating a target prediction.

$$\text{AP} = \frac{1}{|\boldsymbol{X}||\boldsymbol{Y}|} \sum_{i=1}^{|\boldsymbol{Y}|} d_i \quad (2)$$

Despite AP's simplicity, several concerns have been raised. Specifically, AP is not length invariant, i.e. the value of the metric depends on the input and output lengths. For instance, AP for a wait-3 model (Ma et al., 2019) is 0.72 when $|\boldsymbol{X}| = |\boldsymbol{Y}| = 10$ but 0.52 when $|\boldsymbol{X}| = |\boldsymbol{Y}| = 100$. Moreover, AP is not evenly distributed on the $[0, 1]$ interval, i.e., values below 0.5 represent models that have lower latency than an ideal policy, and an improvement of 0.1 from 0.7 to 0.6 is much more difficult to obtain than the same absolute improvement from 0.9 to 0.8 (Ma et al., 2019).

Average Lagging (AL) first defines an ideal policy, which is equivalent to a wait-0 policy that has the same prediction as the system to be evaluated. Ma et al. (2019) define AL as

$$\text{AL} = \frac{1}{\tau(|\boldsymbol{X}|)} \sum_{i=1}^{\tau(|\boldsymbol{X}|)} d_i - \frac{(i-1)}{\gamma} \quad (3)$$

[2] http://iwslt2020.ira.uka.de/doku.php?id=simultaneous_translation

[3] https://autosimtrans.github.io/

where $\tau(|\boldsymbol{X}|) = \min\{i|d_i = |\boldsymbol{X}|\}$ is the index of the target token when the policy first reaches the end of the source sentence and $\gamma = |\boldsymbol{Y}|/|\boldsymbol{X}|$. $(i-1)/\gamma$ term is the ideal policy for the system to compare with. AL has good properties such as being length-invariant and intuitive. Its value directly describes the lagging behind the ideal policy.

Differentiable Average Lagging (DAL) introduces a minimum delay of $1/\gamma$ after each operation. Unlike AL, it considers the tokens when $i > \tau(|\boldsymbol{X}|)$ (Cherry and Foster, 2019). It is defined in Eq. (4):

$$\text{DAL} = \frac{1}{|\boldsymbol{Y}|}\sum_{i=1}^{|\boldsymbol{Y}|} d'_i - \frac{i-1}{\gamma}, \tag{4}$$

where

$$d'_i = \begin{cases} d_i & i = 0 \\ \max(d_i, d'_{i-1} + \gamma) & i > 0 \end{cases}. \tag{5}$$

A minimum delay prevent DAL recovering from lagging once it has been incurred.

3.2 Adapting Metrics to the Speech Task

In this section, we adapt the three latency metrics introduced in Section 3.1 to the simultaneous speech translation task.

Average Proportion is straightforward to adapt to the speech task and as follows:

$$\text{AP}_{\text{speech}} = \frac{1}{|\boldsymbol{Y}|\sum_{j=1}^{|\boldsymbol{X}|} T_j}\sum_{i=1}^{|\boldsymbol{Y}|} d_i \tag{6}$$

Average Lagging is adapted as follows:

$$\text{AL}_{\text{speech}} = \frac{1}{\tau'(|\boldsymbol{X}|)}\sum_{i=1}^{\tau'(|\boldsymbol{X}|)} d_i - d^*_i, \tag{7}$$

where $\tau'(|\boldsymbol{X}|) = \min\{i|d_i = \sum_{j=1}^{|\boldsymbol{X}|} T_j\}$ and d^*_i are the delays of an ideal policy, of which the straightforward adaption is $d^*_i = (i-1) \times \sum_{j=1}^{|\boldsymbol{X}|} T_j \,/|\boldsymbol{Y}|$. However such adaptation is not robust for models that tend to stop hypothesis generation too early and generate translations that are too short. This is more likely to happen in simultaneous speech translation where a model can generate the end of sentence token too early, for example when there is a long pause even though the entire source input has not been consumed. Fig. 1 illustrate this phenomenon. The red line in Fig. 1 corresponds to the ideal policy defined in (Ma et al., 2019). We can see that when the model stops generating the translation, the lagging behind the ideal policy is negative. This is because the model stops reading any input after completing hypothesis generation. This kind of model can obtain relatively good latency-quality trade-offs as measured by AL (and BLEU), which does not reflect the reality. We thus define

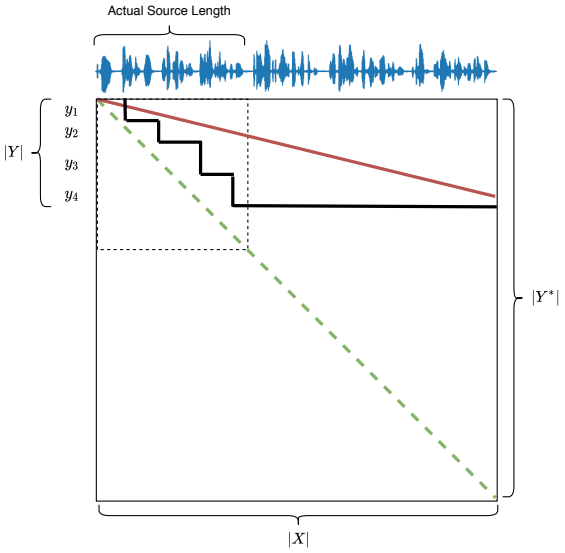

Figure 1: An example of original AL failed on early stop translation. Red (solid straight) line shows the ideal policy in (Ma et al., 2019). Green (dotted straight) line depicts the modified ideal policy in this paper. Black (solid zigzag) line demonstrates the alignment between source and target.

$$d^*_i = (i-1) \cdot \sum_{j=1}^{|\boldsymbol{X}|} T_j \,/|\boldsymbol{Y}^*| \tag{8}$$

to prevent this issue, i.e., it is assumed that the ideal policy generates the reference rather than the system hypothesis. The newly defined ideal policy is represented by the green line in Fig. 1.

Differentiable Average Lagging for the speech task still uses Eq. (4) and Eq. (5) with a new γ defined as

$$\gamma_{\text{speech}} = |\boldsymbol{Y}|/\sum_{j=1}^{|\boldsymbol{X}|} T_j \tag{9}$$

4 Architecture

SIMULEVAL simulates a real-time scenario by setting up a server and a client. The server and client

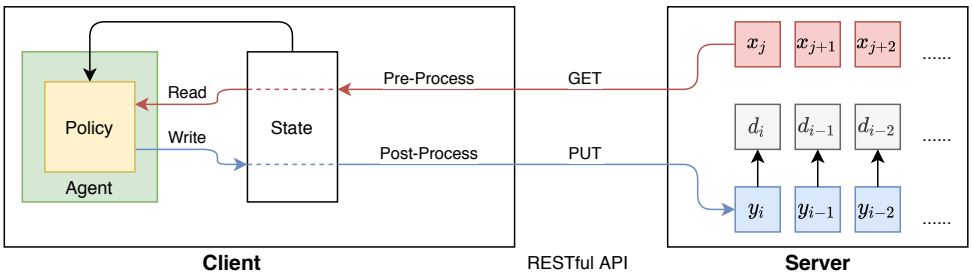

Figure 2: The architecture of SIMULEVAL. The client executes the policy and the server operates the evaluation.

can be run separately or jointly, and are connected through RESTful APIs. An overview is shown in Fig. 2.

4.1 Server

The server has primarily four functions. First, read source and reference files. Second, send source segments to the client upon a `READ` action. Third, receive predicted segments from the client upon a `WRITE` action, and record the corresponding delays. Fourth, run the evaluation on instances.

The evaluation process by the server on one instance is shown in Algorithm 1. Note that in line 18 in Algorithm 1, the server only runs sentence-level metrics. The server will collect $\boldsymbol{Y}$, $\boldsymbol{D}$ and $\boldsymbol{T}$ for every instance in the evaluation corpus, and calculate corpus-level metrics after all hypotheses are complete.

Algorithm 1 Server side algorithm

Input: $\boldsymbol{X} = [x_1, ..., x_{|\boldsymbol{X}|}], \boldsymbol{Y}^* = [y^*_1, ..., y^*_{|\boldsymbol{Y}^*|}]$
Input: $\boldsymbol{Y} = [], \boldsymbol{D} = []$
Input: $i = 0, j = 0, y_0 = \text{BOS}, d_0 = 0$
1: **while** $y_i \neq \text{EOS}$ **do**
2: r = await_request_from_client()
3: **if** r.action == READ **then**
4: **if** $j < |\boldsymbol{X}|$ **then**
5: $j = j + 1$
6: send_segment_to_client(x_j)
7: **else**
8: send_segment_to_client(EOS)
9: **else**
10: $i = i + 1$
11: $y_i = r$.segment
12: $\boldsymbol{Y} = \boldsymbol{Y} + [y_i]$
13: **if** data type is speech **then**
14: $d_i = d_{i-1} + T_j$
15: **else**
16: $d_i = j$
17: $\boldsymbol{D} = \boldsymbol{D} + [d_i]$
18: **return** evaluate($\boldsymbol{Y}, \boldsymbol{Y}^*, \boldsymbol{D}, \boldsymbol{T}$)

4.2 Client

The client contains two components — an agent and a state. The agent is a user-defined class that operates the policy and generates hypotheses for simultaneous translation, the latter provides functions such as pre-processing, post-processing and memorizing context. The purpose of this design is to make the user free from complicated setups, and focus on the policy. The client side algorithm is shown in Algorithm 2.

Algorithm 2 Client side algorithm

Input: $\boldsymbol{X} = [], i = 0, j = 0, y_0 = \text{BOS}$, State, Agent
1: **while** $y_i \neq \text{EOS}$ **do**
2: action = Agent.policy(State)
3: **if** action == READ **then**
4: x = request_segment_from_server()
5: **if** x is not EOS **then**
6: $j = j + 1$
7: x_j = State.preprocess(x)
8: States.update_source(x_j)
9: continue
10: $i = i + 1$
11: y_i = Agent.predict(State)
12: y_i = State.postprocess(y_i)
13: States.update_target(y_i)
14: send_segment_to_server(y_i)

5 Usage Instructions

5.1 User-Defined Agent

A user-defined agent class is required for evaluation, along with the user's model specific arguments. The user is able to add customized arguments and initialize the model. Two functions must be defined in order to successfully run online decoding. The first one is "policy", which takes the state as input and returns a decision on whether to perform a *read* or *write* action. The other function is "predict" which will be called when the "policy" returns a *write* action and return a new target prediction given the state. An example of a text wait-k model is shown below.

```python
from simuleval.agents import TextAgent
from simuleval import READ_ACTION, WRITE_ACTION,
↪  DEFAULT_EOS
# User defined model code
from user_library import init_model

class WaitKTextAgent(TextAgent):
    def __init__(self, args):
        super().__init__(args)
        # Initialization
        self.waitk = args.waitk
        self.model = init_model(args.model)

    @staticmethod
    def add_args(parser):
        # Customized arguments
        parser.add_argument(
            "--waitk", type=int,
            help="Lagging between source and
            ↪  target")
        parser.add_argument(
            "--model", help="model specifics")

    def preprocess(self, state):
        # preprocess code
        return state

    def postprocess(self, state):
        # postprocess code
        return state

    def policy(self, state):
        # Make a decision here
        if (
            len(state.source) - len(state.target)
            < self.waitk
            and not state.finish_read()
        ):
            return READ
        else:
            return WRITE

    def predict(self, state):
        # Predict a token here
        # Called when self.policy() returns
        ↪  WRITE_ACTION
        return model.predict(state)
```

Listing 1: An example of user defined agent class for a text wait-k model.

Additionally, the user can define pre-processing or post-processing methods to handle different types of input. For example, for a speech translation model, the pre-processing method can be a feature extraction function that converts speech samples to filterbank features while for text translation, the pre-processing can be tokenization or subword splitting. Post-processing can implement functions such as merging subwords and detokenization.

5.2 User-Defined Client

A typical user will only need to implement an agent and will rely on the out-of-the-box client implementation of Algorithm 2. However, sometimes, a user may want to customize the client, for example if they want to use a different programming language than Python or make the implementation of Algorithm 2 more efficient. In that case, they can take advantage of the RESTful APIs between the client and the server described in Table 1. Users can easily plug in these APIs into their own client implementations.

5.3 Evaluation

With a well-defined agent class, SIMULEVAL is able to start the evaluation automatically. Assuming the agent class is stored in `text_waitk_agent.py`, the evaluation can be run in one single command or separate commands:

```
simuleval \
  --output $OUT_DIR \
  --source $SOURCE \
  --reference $TARGET \
  --agent text_waitk_agent.py \
  --waitk 3 \
  --model $MODEL_PARAMS
```

Listing 2: Evaluation command (joint)

```
simuleval server \
  --output $OUT_DIR \
  --port 5000 \
  --source $SOURCE \
  --reference $TARGET &

simuleval client \
  --port 5000 \
  --agent text_waitk_agent.py \
  --waitk 3 \
  --model $MODEL_PARAMS
```

Listing 3: Evaluation command (Separate)

After all hypotheses are generated, the intermediate results and corpus level evaluation metrics will be saved in the output directory. SIMULEVAL also supports resuming an evaluation if the process has been interrupted.

5.4 Visualization

SIMULEVAL provides a web user interface (UI) for visualizing the online decoding process. Fig. 3 shows an interactive example on simultaneous speech translation. A user can move the cursor to find the corresponding translation at a certain point. The visualization server can be simply started by

```
simuleval server --visual --log-dir $OUT_DIR
```

The default port is 7777 and the web UI can be accessed at `http://ip-of-server:7777`.

5.5 Case Study: IWSLT 2020

In order to avoid inconsistencies in how latency metrics are computed and to ensure fair comparisons between results presented in research papers, we encourage the research community to use SIMULEVAL when reporting latency in the future.

Function	Endpoint	Params	Response / Body
Get next source segment x_j	/src	{sent_id: sent_idx}	x_j (text)
Get next T_j ms speech segment x_j	/src	{sent_id: sent_idx, segment_size: T_j}	x_j (samples)
Send a predicted y_i	/hypo	{sent_id: sent_idx}	y_i

Table 1: A subset of the RESTful APIs for the SIMULEVAL server.

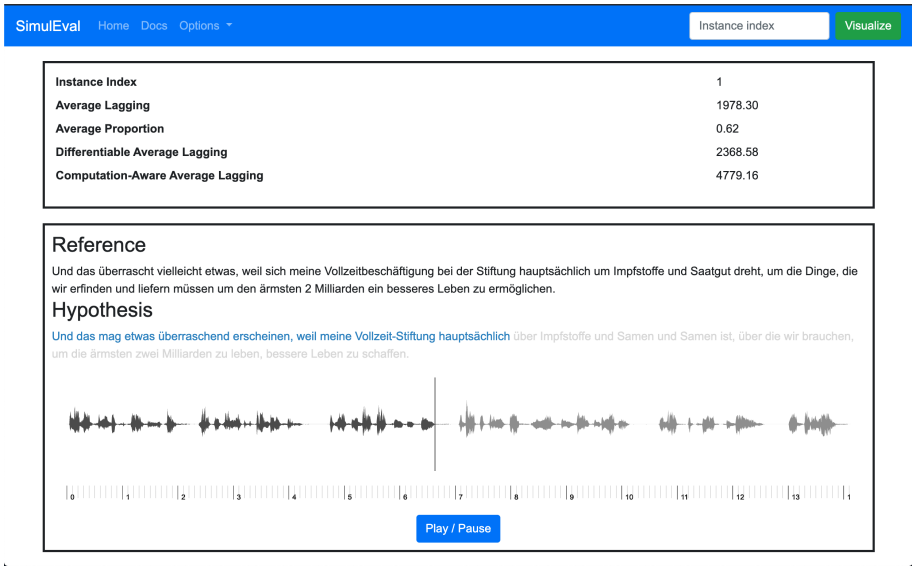

Figure 3: Visualization interface of SIMULEVAL.

In addition, an earlier version of SIMULEVAL was used in the context of the first simultaneous speech translation shared task at IWSLT (Ansari et al., 2020), where it is of paramount importance to have the same evaluation conditions for all submissions. In order to preserve the integrity of the evaluation process, the test set, including the source side, could not be released to participants. This motivated the client-server design, where participants defined their own agent file and submitted their system in a Docker (Merkel, 2014) environment. The organizers of the task were then able to run SIMULEVAL and score each submission in a consistent way, even for systems implemented in different frameworks.

6 Conclusion

In this paper, we introduced SIMULEVAL, a general and easy-to-use evaluation toolkit for simultaneous speech and text translation. It simulates a real-time scenario with a server-client scheme and automatically evaluates simultaneous translation given a user-defined agent, both for text and speech. Furthermore, it provides a visualization interface for the user to track the online decoding process. We introduced example use cases of the toolkit and showed that its general design allows evaluation on different frameworks. We encourage future research on simultaneous speech and text translation to make use of this toolkit in order to obtain an accurate and standard comparison of the latency between different systems.

References

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. 2018. Prediction improves simultaneous neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3022–3027.

Ebrahim Ansari, amittai axelrod, Nguyen Bach, Ondřej Bojar, Roldano Cattoni, Fahim Dalvi, Nadir Durrani, Marcello Federico, Christian Federmann, Jiatao Gu, Fei Huang, Kevin Knight, Xutai Ma, Ajay Nagesh, Matteo Negri, Jan Niehues, Juan Pino, Elizabeth Salesky, Xing Shi, Sebastian Stüker, Marco Turchi, Alexander Waibel, and Changhan Wang. 2020. FINDINGS OF THE IWSLT 2020 EVALUATION CAMPAIGN. In *Proceedings of the 17th International Conference on Spoken Language Translation*, pages 1–34, Online. Association for Computational Linguistics.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. 2019. 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. Association for Computational Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

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

Kyunghyun Cho and Masha Esipova. 2016. Can neural machine translation do simultaneous translation? *arXiv preprint arXiv:1606.02012*.

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

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor OK Li. 2017. Learning to translate in real-time with neural machine translation. In *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017*, pages 1053–1062. Association for Computational Linguistics (ACL).

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. 2019. STACL: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036, Florence, Italy. Association for Computational Linguistics.

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

Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Yi Ren, Jinglin Liu, Xu Tan, Chen Zhang, Tao QIN, Zhou Zhao, and Tie-Yan Liu. 2020. 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. Association for Computational Linguistics.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200. Cambridge, MA.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019. 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. Association for Computational Linguistics.