Lab2 Report: End-to-End Speech Translation System Benchmarking

Team Jobless
Jiyang Tang (jiyangta), Ran Ju (ranj), Tinglong Zhu (tinglonz), Xinyu Lu (xinyulu2)

1 Experimental Setup

1.1 Baseline

The end-to-end speech translation architecture for our benchmark is an encoder-decoder model with multitask training objectives (Inaguma et al., 2020). During training, the encoder is trained with CTC loss (Graves et al., 2006) using source text as the ground truth, while the decoder, called *MT decoder*, is trained using label smoothing cross-entropy loss using target text as the ground truth. In addition, a separate transformer decoder sharing the same encoder connection called the *ASR decoder*, is trained using label smoothing cross-entropy loss using source text as the ground truth. Only the encoder and the MT decoder are used during inference.

To save time, we utilize a pre-trained model[1] from Hugging Face Hub for our experiments. This English-to-Dutch translation model contains a Conformer (Gulati et al., 2020) encoder with 12 layers and a Transformer (Vaswani et al., 2017) decoder with 6 layers. The encoder has a subsampling CNN layer with kernel size 31, followed by identical Conformer blocks with 4 attention heads and 2048 hidden units. The MT decoder blocks each contain 4 attention heads and 2048 hidden units. Both the encoder and the decoder use a dropout rate of 0.1 for positional encoding and attention. The beam size is set to 10 during inference.

1.2 Evaluation

BLEU (Bilingual Evaluation Understudy) measures the similarity of the machine-translated text to a set of reference texts. It is a number between zero and one, and the higher the score is, the higher the quality the translation has. Previous studies have shown that BLEU scores have a high correlation with human judgment (Papineni et al., 2002) and it has become one of the canonical metrics for speech translation among the research community. Thus, we use BLEU scores to represent the quality of our translation models.

1.3 Data

We use the test set from MuST-C (Di Gangi et al., 2019) for evaluating our models. For each language pair, MuST-C contains hundreds of hours of audio recordings from English TED talks and sentence-level transcriptions and translations. Considering the model size and inference speed, we selected 20 samples from the English-to-Dutch test set to calculate our benchmarking metrics.

1.4 Benchmarking

For benchmarking, we measure the latency in seconds, FLOPs, and BLEU scores for each experiment and visualize the results. For latency, we measure the time it takes to perform inference on every test sample and take the average of them except the first one. We rely on the DeepSpeed (Rasley et al., 2020) library for FLOPs calculation. Specifically, we measure the total FLOP count for translating all samples and take the average to mitigate the effect of varying input audio lengths. However, since the DeepSpeed FLOP profiler only works on CUDA devices, we have to run experiments to calculate FLOPs separately using an Nvidia V100 16GB GPU instead of only the CPU.

Except for FLOPs calculation, all experiments are conducted using a laptop CPU under the same conditions. We stopped as many applications and background services as possible before benchmarking. The hardware specification is shown below:

1. Laptop: Dell G15 5511
2. CPU
   (a) 11th Gen Intel i7-11800H, 2.30GHz
   (b) Speed 3.35 GHz

[1] https://huggingface.co/espnet/brianyan918_mustc-v2_en-de_st_conformer_asrinit_v2_raw_en_de_bpe_tc4000_sp

(c) 8 cores, 16 logical processors
(d) L1 cache: 640 KB
(e) L2 cache: 10.0 MB
(f) L3 cache: 24.0 MB

3. Memory: 32GB, dual-channel DDR4
4. OS: Windows 10 Education 22H2, OS Build 19045.3570
5. Python environment:
   (a) Python==3.10
   (b) PyTorch==1.12.1
   (c) ESPnet==202308
6. Test data is loaded into memory beforehand.

All experiments are conducted using ESPnet (Watanabe et al., 2018; Inaguma et al., 2020). Since each sample has different durations and output lengths, there will exist some variance between latencies from different samples.

2 Varying Input Size

The first way we vary the input size is to change the hop length. The larger the hop length, the smaller the input size. The hop length for each experiment is shown in Table 1.

The experiment results are shown in Figure 1. From the BLEU score vs FLOPs plot we observe that as we decrease the input size, the FLOPs decrease as expected. As the FLOPs decrease, we don't see a clear trend of increasing or decreasing BLEU scores. We think the reason is that we didn't re-train the model after changing the hop length. Since the model is trained using a hop length of 160, inference with different hop lengths will affect and bring a lot of uncertainty to the BLEU score performance.

The BLEU score vs latency graph looks similar to the BLEU score vs FLOPs one. The latency decreases as we decrease the input size. The BLEU score generally decreases as we change the input size because of the different input sizes used to train and infer the model, but there is no specific trend observed.

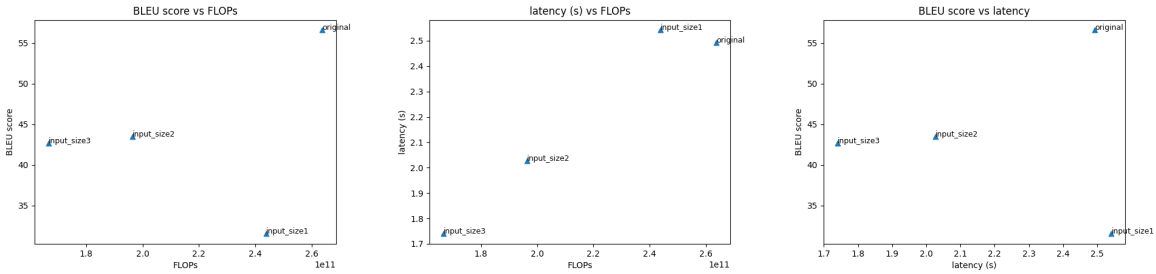
For the relationship between FLOPs and latency, we find that generally as the FLOPs increase, the latency increases as well. However, for the original and input_size1 experiments, the relationship is the opposite. From original to input_size1, we increase the hop length by 40, which is slightly smaller than the difference between input_size2 and input_size3. We hypothesize the reason is that it is related to the way the hardware deals with memory optimization.

The second way that we vary input size is to first decrease the sample rate from 16k to 8k and then increase the hop length to decrease the input size, and the hop length hyperparameters are in Table 1. The experiment results are shown in Figure 2. By examining the graph depicting the relationship between BLEU score and FLOPs, it becomes apparent that reducing the input size leads to the expected decrease in FLOPs. However, as FLOPs decrease, there isn't a discernible pattern of either an increase or a decrease in the BLEU score. The graph illustrating the relationship between BLEU score and latency closely resembles the one comparing BLEU score to FLOPs. As the input size decreases, latency also decreases. The BLEU score tends to decrease when adjusting the input size due to disparities between the training and inference input sizes, although there isn't a clear, consistent trend observed. When examining the connection between FLOPs and latency, we generally observe that as FLOPs increase, latency also tends to increase.

The difference between the two methods is that the first one only changes the hop length while the second one decreases the sample rate before increasing the hop length. We choose these two methods to understand how changing hop length and sample rate affects the latency and BLEU score. In terms of the BLEU score, we observe that if we decrease the sample rate by half, the BLEU score drops dramatically, except input_size2, the BLEU scores for all other models drop to less than 10. We think the reason is that the dropped samples contain important context information for the model to understand. In terms of latency, we find that if we decrease the sample rate first, the latency will decrease significantly. It is as expected due to there is less computation. In all, we think decreasing the sample rate is not a good strategy for decreasing the input size since the BLEU score drops too much. In comparison, changing hop length directly has a better trade-off between performance and latency.

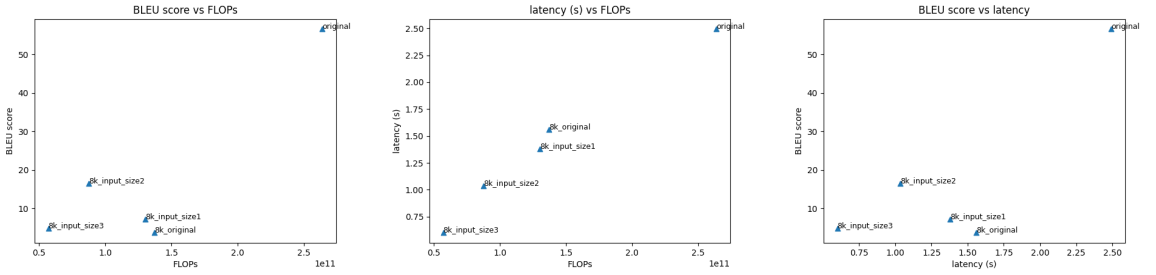
3 Varying Model Size

Modifying a pre-trained model's architecture can drastically alter its performance. Table 3 shows the experiments we conducted to adjust model parameters.



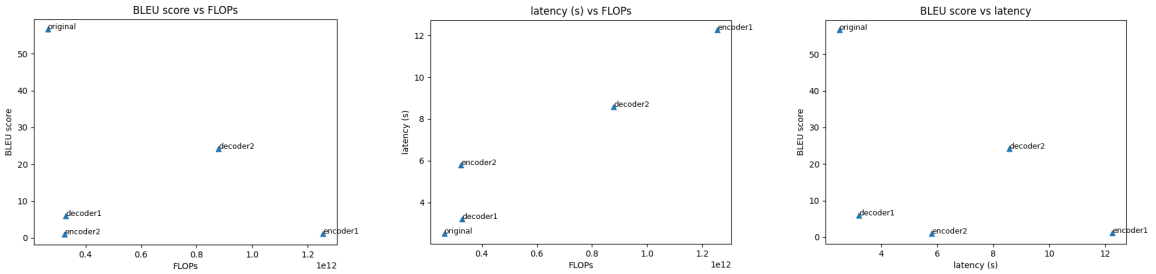
(a) FLOPs vs. BLEU (b) FLOPs vs. Latency (c) Latency vs. BLEU

Figure 1: Comparative Plots For Different Input Size (Method 1)



(a) FLOPs vs. BLEU (b) FLOPs vs. Latency (c) Latency vs. BLEU

Figure 2: Comparative Plots For Different Input Size (Method 2)



(a) FLOPs vs. BLEU (b) FLOPs vs. Latency (c) Latency vs. BLEU

Figure 3: Comparative Plots For Different Model Size

Model	Hop length
original	160
input_size1	200
input_size2	250
input_size3	300

Table 1: Experiments of Varying Input Size

1. **Reducing Layer Depth and Widths**

   To reduce the layer depth, we reduced the number of blocks with decoder1 and encoder1. Specifically, we keep the first N layers and discard subsequent ones. In addition, we reduce the width of the entire model reducing the output size of both the encoder and the decoder in encoder2 experiment. Decreasing the model size reduces the capability of the model to recognize complex patterns while drastically cutting down the computational cost and parameter count.

2. **Reducing Number of Attention Heads**

   In decoder2 and encoder2. we adjusted attention_heads and linear_units. In this way, we lessen the model's capacity to focus on various parts of the input sequence in parallel while narrowing the model width.

   In these ways, we reduce the parameters and maintain the already learned features as much as possible at the same time. All the methods aim to minimize the computational burden.

Model	*num_blocks*	*attention_heads*	*linear_units*	*output_size*	**Total** parameter count
Original (Encoder)	12	4	2048	256	57,592,032
Original (Decoder)	6	4	2048	–	57,592,032
decoder1	4	–	–	–	54,434,528
decoder2	–	2	1024	–	54,440,160
encoder1	8	–	–	–	47,033,568
encoder2	–	2	1024	128	18,918,368

Table 2: Parameter settings for different models compared to original ESPNet parameters. – indicates that the parameter retains its original value.

The results are shown in Figure 3. From the result, we could see that the model with the lowest FLOPs(decoder1) does not have the lowest BLEU score. decoder2 shows a relatively high BLEU score with intermediate FLOPs, indicating a good trade-off between computational complexity and model performance. decoder1 has a relatively low latency and moderate BLEU score which could be advantageous in real-time applications where lower latency is critical. encoder1 and encoder2 seem to be ineffective.

Our explanations are:

- When reducing model size without retraining, the pre-existing weights might not be optimal for the new architecture. It could lead to nonsensical outputs from the decoder.
- The decoder relies on its previous outputs so without proper adaptation to the reduced size architecture, it could generate "garbage" outputs, repeating or elongating translations.
- To mitigate the problems, we should reduce the model's size coupled with retraining, fine-tuning, layer pruning, or employing knowledge distillation techniques.

4 Putting it together

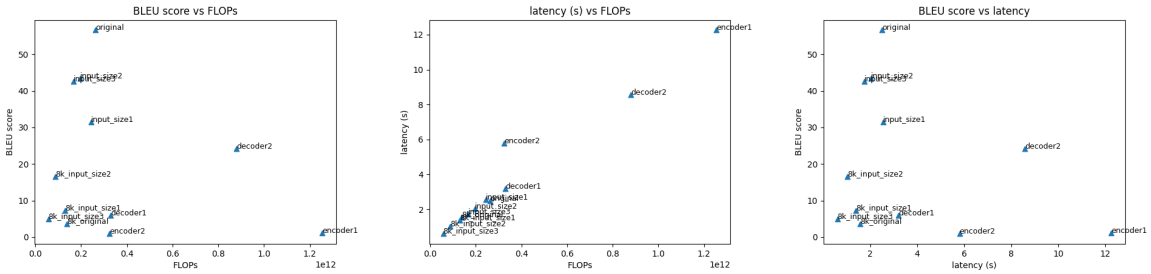
Model	BLEU scores	Latency (s)	FLOPs
Original	**56.6**	2.5	263,773,099,929
encoder1	1.2	12.3	1,254,943,585,484
encoder2	0.9	5.8	324,969,515,078
decoder1	6.0	3.2	329,232,078,169
decoder2	24.2	8.6	878,961,158,694
input_size1	31.5	2.5	243,909,508,076
input_size2	43.5	2.0	196,505,325,796
input_size3	42.6	1.7	166,594,033,230
8k_input_size1	7.3	1.4	130,435,959,142
8k_input_size2	16.5	1.0	87,710,174,390
8k_input_size3	4.9	**0.6**	**57,216,657,598**

Table 3: Benchmarking results for all experiments.

Here, we did not conduct experiments that combine different input sizes and model sizes. Reducing the model sizes without fine-tuning and retraining will lead to huge and unacceptable performance degradation (e.g. senseless sentences, repeating a single token/words until reaching the maximum length, etc.). If we reduce the model size and input size at the same time, the results will get worse. Thus, we did not combine different input sizes and model sizes in this section, but we did put the results corresponding to different input sizes and model sizes together for analysis (Table 4).

From the scatter plots below, we can see that reducing the input sizes (by increasing the hop length of MFCC) can help dramatically improve the latency and FLOPs while keeping a merely acceptable performance degradation. One possible explanation for this could be speech signals are periodic signals, the semantic information contained in a period may be the same. Since increasing the hop length will not lead to a huge information loss, the reduction of input sizes also has little risk of repeating the outputs until reaching the maximum output length. Thus, increasing the hop length of MFCC features may not lead to a huge information loss but can help reduce the latency since the number of frames the encoder has to deal with is only half of that of the original. But directly downsampling the waveform will cause dramatic performance degradation; this might be due to the losing information from part of the frames in the raw waves. In comparison, increasing hop length can reduce input size while keeping an acceptable information loss since it gets information from all frames in the raw waves.

However, when we tried to decrease the model size, the performance dropped dramatically (no matter whether we reduced the size of the encoder or the decoder). As we only keep the first few layers of the encoder/decoder, we will lose the se-



(a) FLOPs vs. BLEU (all together) (b) FLOPs vs. Latency (all together) (c) Latency vs. BLEU (all together)

Figure 4: Comparative Plots (All together)

mantic insights from the latter layers, which are important for translation. In addition, as mentioned above, losing semantic information will cause the model to repeatedly generate the same tokens/words, which will not only decrease its performance but will also greatly increase its inference latency.

In short, if we want to improve the model inference latency without having serious performance degradation and without retraining the model, increasing MFCC's hop length to 300 would be a good solution.

References

Mattia A. Di Gangi, Roldano Cattoni, Luisa Bentivogli, Matteo Negri, and Marco Turchi. 2019. 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. Association for Computational Linguistics.

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

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. 2020. Conformer: Convolution-augmented transformer for speech recognition. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pages 5036–5040. ISCA.

Hirofumi Inaguma, Shun Kiyono, Kevin Duh, Shigeki Karita, Nelson Enrique Yalta Soplin, Tomoki Hayashi, and Shinji Watanabe. 2020. Espnet-st: All-in-one speech translation toolkit. *arXiv preprint arXiv:2004.10234*.

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*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA. Association for Computing Machinery.

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

Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. 2018. Espnet: End-to-end speech processing toolkit. In *Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 September 2018*, pages 2207–2211. ISCA.