wav2vec-S: Adapting Pre-trained Speech Models for Streaming

Biao Fu\textsuperscript{1,3} *, Kai Fan\textsuperscript{2,1}, Minpeng Liao\textsuperscript{2}, Yidong Chen\textsuperscript{1,3}, Xiaodong Shi\textsuperscript{1,3} †, Zhongqiang Huang\textsuperscript{2}

\textsuperscript{1}School of Informatics, Xiamen University
\textsuperscript{2}Alibaba Group
\textsuperscript{3}Key Laboratory of Digital Protection and Intelligent Processing of Intangible Cultural Heritage of Fujian and Taiwan (Xiamen University), Ministry of Culture and Tourism

biaофu@stu.xmu.edu.cn, {ydchen,mandel}@xmu.edu.cn
{k.fan,minpeng.lmp,z.huang}@alibaba-inc.com

Abstract

Pre-trained speech models, such as wav2vec 2.0, have significantly advanced speech-related tasks, including speech recognition and translation. However, their applicability in streaming scenarios is limited because these models are trained on complete utterances, leading to a mismatch with incremental streaming inputs. This paper identifies three critical design aspects within the architecture of wav2vec 2.0 and proposes a novel model, wav2vec-S, which incorporates simple modifications to ensure consistent speech representations during both training and inference phases for streaming speech inputs. Furthermore, we demonstrate that wav2vec-S models can be efficiently adapted from pre-trained wav2vec 2.0 models through continued pre-training and effectively finetuned to meet various latency requirements in downstream applications. Experiments on speech recognition and translation tasks show that wav2vec-S outperforms strong baseline models and achieves a superior balance between quality and latency.

1 Introduction

The advent of pre-trained speech models, such as wav2vec (Schneider et al., 2019; Baevski et al., 2020b), HuBERT (Hsu et al., 2021), WavLM (Chen et al., 2022), and data2vec (Baevski et al., 2022, 2023), have catalyzed significant progress in speech-related tasks, including automatic speech recognition (ASR) (Baevski et al., 2020b; Hsu et al., 2021) and speech translation (ST) (Xu et al., 2021; Fang et al., 2022; Wang et al., 2022; Fang and Feng, 2023). These models learn powerful speech representations from large amounts of unlabeled speech data through self-supervised learning techniques, leading to significant performance improvements.

Despite the advancements, their potential has not yet been fully harnessed for streaming scenarios. These models, trained and optimized for processing complete utterances, yield sub-optimal performance when applied to partial speech inputs in streaming scenarios. Previous efforts to bridge this gap have often utilized knowledge distillation techniques (Cao et al., 2021; Shim et al., 2023; Fu et al., 2023a), with the goal of transferring knowledge from offline pre-trained models to streaming student models. However, these strategies require task-specific distillation, and generally involve complex, multi-stage training processes (Cao et al., 2021; Yang et al., 2022) or the need to balance multiple distillation objectives (Shim et al., 2023; Fu et al., 2023b,a). This limits their applicability as universal pre-trained models. Moreover, these methods do not fully exploit the potential of large-scale unlabeled data for streaming speech tasks.

In this paper, we focus on wav2vec 2.0 (Baevski et al., 2020b) and identify three critical design aspects of its architecture that hinder its applicability to streaming tasks. Accordingly, we propose a novel model, wav2vec-S, which incorporates simple modifications to accommodate streaming requirements. Specifically, we substitute layer normalization for group normalization in the feature extractor, shift from convolution-based relative position encoding to absolute position encoding, and implement a block-wise unidirectional self-attention mechanism, all designed to eliminate or restrict dependency on future context, as required for maintaining consistent representations during training and inference for streaming inputs. We demonstrate that instead of training from scratch, wav2vec-S can be efficiently adapted from pre-trained wav2vec 2.0 models through continued pre-training on unlabeled speech using the same self-supervised training objectives. In addition, we in-

\textsuperscript{*} Work was done during Biao Fu’s internship at Alibaba.
\textsuperscript{†} Corresponding author.
introduce mechanisms in both the pre-training stage and the fine-tuning stage to allow the model to meet various latency requirements in downstream tasks.

In summary, our contributions are threefold:

1. We present wav2vec-S, a model adapted from wav2vec 2.0 with straightforward architectural modifications tailored for streaming tasks, which can be efficiently adapted from pre-trained wav2vec 2.0 models through continued pre-training.

2. We conduct comprehensive experiments on two popular streaming speech tasks: ASR and ST, to demonstrate that fine-tuning from wav2vec-S outperforms multiple strong baseline approaches in balancing between quality and latency.

3. We release the source code and trained models to facilitate future research of pretrained models for streaming tasks.

2 Preliminary

Wav2vec 2.0 represents a significant advancement in speech processing and has been widely utilized as a backbone for various speech-related tasks (Yi et al., 2021; Pepino et al., 2021; Fan et al., 2021; Xu et al., 2021; Siuzdak et al., 2022; Popuri et al., 2022). Our proposed model is based on wav2vec 2.0 with essential modifications to support streaming applications. We first briefly review its model architecture and training objectives.

2.1 Model Architecture

The architecture of wav2vec 2.0 includes three main components: a feature encoder, a context network, and a quantization module.

**Feature Encoder.** The feature encoder converts the raw audio waveform into feature representations. It consists of seven temporal convolutional layers, each followed by group normalization (GN) (Wu and He, 2018) and a GELU activation function (Hendrycks and Gimpel, 2019).

**Context Network.** The context network adopts the Transformer architecture (Vaswani et al., 2017) along with a convolution-based relative position embedding layer. It processes the output of the feature encoder to extract contextualized representations of the speech.

**Quantization Module.** This module employs vector quantization to map the continuous feature representations from the feature encoder to discrete tokens via a codebook and a Gumbel-Softmax operation (Jang et al., 2016). These discrete tokens are used as target labels for self-supervised learning of the entire network.

2.2 Training Objectives

The training of wav2vec 2.0 integrates both contrastive loss and diversity loss, as follows:

\[ L = L_c + \alpha L_d, \]

(1)

**Contrastive Loss.** The loss \( L_c \) leverages the contextualized representation \( c_t \) from the context network, based on feature representations masked at time step \( t \). It aims to differentiate the true quantized vector \( q_t \) from a set of negative samples \( \bar{q}_t = \bar{q}_1, \ldots, \bar{q}_k \), sampled from different masked time steps of the same utterance. The loss is calculated as follows:

\[ L_c = -\sum_t \log \frac{\exp (\text{sim} (c_t, q_t) / \tau)}{\sum_{\tilde{q} \in \bar{q}_t} \exp (\text{sim} (c_t, \tilde{q}) / \tau)} \]

(2)

**Diversity Loss.** The loss \( L_d \) encourages the model to utilize the entire spectrum of quantized speech representations, by maximizing the entropy of the averaged distribution of codebook entries.

3 Our Proposal: wav2vec-S

We next describe how to adapt wav2vec 2.0 to wav2vec-S for streaming, focusing on three key aspects: architecture, pre-training, and fine-tuning.

3.1 Architectural Modifications

We identify three critical design aspects within the architecture of wav2vec 2.0 that hinder its application to streaming scenarios. We propose simple modifications to preserve its modeling capabilities while enabling streaming processing.

3.1.1 Feature Normalization

In the wav2vec 2.0 architecture, group normalization (GN) (Wu and He, 2018) is applied within the feature encoder, where the normalization process calculates the mean and standard deviation across both the sequence and feature dimensions of complete utterances. However, this methodology results in sub-optimal performance for streaming applications, in which only partial utterances are available during real-time processing, due to the
difference in statistical computation between the training and inference stages.

To address this issue, we utilize layer normalization (LN) (Ba et al., 2016), which determines the mean and standard deviation exclusively along the feature dimension, disregarding the sequence length. This ensures consistent processing, whether dealing with complete or partial utterances. While other normalization techniques like weight normalization (Salimans and Kingma, 2016), employed in Encodec (Défossez et al., 2022) for streaming models, may also be viable, since it reparametrizes the layer’s parameters instead of altering the data values. We leave the investigation of various normalization methods for future research.

3.1.2 Positional Encodings

wav2vec 2.0 incorporates implicit relative positional encodings (PE), as detailed in (Mohamed et al., 2020), within its context network. These encodings are generated through a 1D convolution layer with a substantial kernel size, representing 2560ms of speech. This extensive context size necessitates significant latency to accurately compute positions at the most recent segments of streaming input, without resorting to padding, to maintain alignment with the training on complete utterances.

To overcome these challenges, wav2vec-S adopts absolute sinusoidal PE (Vaswani et al., 2017) for its context network. This method ensures stable positional information, regardless of the availability of future context, making it suitable for streaming applications.

3.1.3 Self-Attention

Despite its strong modeling capabilities by accessing both past and future contexts, the use of bidirectional self-attention in wav2vec 2.0 presents two limitations for streaming applications. Firstly, the absence of future context for the latest segment in streaming input creates a mismatch between training on complete utterances and inference on partial ones. Secondly, past segments require repeated computation of their bidirectional self-attention as their future context expands. Although unidirectional self-attention, as investigated in prior studies (Ren et al., 2020; Ma et al., 2020b), avoids these pitfalls by only leveraging past context, it results in degraded performance due to the lack of utilizing any future context. Research has shown that incorporating even a small amount of future context can markedly enhance the quality of streaming speech features with minimal latency (Fu et al., 2023a).

In response, we employ block-wise self-attention (Liu et al., 2021), which allows for the utilization of all past context and a limited amount of future context. Specifically, the hidden states in the $l$-th layer, denoted as $b^i_{0:T-1}$, are segmented into $m$-sized non-overlapping blocks, $b^i_l = h^i_{m \times 1 : m \times (i+1) - 1}$, and are calculated in a block-wise manner. The input hidden states at the $(l-1)$-th layer are partitioned into overlapping blocks, adopting a block processing strategy (Tsunoo et al., 2019; Liu et al., 2021) with a stride of $m$ and a block size of $m \times r$, under the constraint that $r \leq m/2$. This results in two parts for each block: the main block $b^i_{l-1} = h^i_{m \times 1 : m \times (i+1) - 1}$ and its right-side future context $r^i_{l-1} = h^i_{m \times (i+1) : m \times (i+1) + r - 1}$, which overlaps with the next block.

To compute the hidden states in each block $b^i_l$ of the $l$-th layer, the query, key, and value matrices are calculated by facilitating bidirectional attention within blocks and unidirectional attention across blocks, as follows:

$$
q^i_{l-1} = W_q \left[ b^i_{l-1}, r^i_{l-1} \right]
$$

$$
k^i_{l-1} = W_k \left[ b^i_{0:l-2}, \ldots, b^i_{l-1}, r^i_{l-1} \right]
$$

$$
v^i_{l-1} = W_v \left[ b^i_{0:l-2}, \ldots, b^i_{l-1}, r^i_{l-1} \right]
$$

3.2 Pre-training

The proposed wav2vec-S model can be pretrained from scratch using a substantial volume of unlabeled speech data, similarly to the wav2vec 2.0 model. However, such a process demands considerable computational resources and time, comparable to the efforts required for training other large pretrained speech models. Given its significant architectural resemblance to wav2vec 2.0, a more efficient strategy is to begin with the pre-trained weights from wav2vec 2.0 and continue pre-training on unlabeled speech data, adhering to the same training loss formula presented in Eq. (1). This method leverages the extensive computational work previously applied to the wav2vec 2.0 model and its learned complex speech patterns, enabling the wav2vec-S model to be trained more efficiently with significantly fewer updates. This approach
is validated by the empirical results discussed in Section 4.6.

The size of blocks in block-wise self-attention directly impacts the latency of the pre-trained model. Utilizing a fixed block size during pre-training could limit performance in streaming applications with diverse latency requirements, as will be demonstrated in Section 4.5. To make the pre-trained wav2vec-S model versatile in streaming scenarios with varied latency needs, we implement a mechanism to dynamically adjust the block size during the pre-training stage. Specifically, the sizes \( m \) of the main context and \( r \) of the future context are randomly selected during each training step from the ranges of \([160\text{ms}, 640\text{ms}]\) and \([80\text{ms}, 320\text{ms}]\), respectively, at increments of \(40\text{ms} \), with the constraint that \( r \leq \frac{m}{2} \).

### 3.3 Fine-tuning

In this work, we focus on streaming speech-to-text tasks as downstream applications, pairing wav2vec-S for parameter initialization of the speech encoder with a separate text decoder. Following recent research trends in this field (Liu et al., 2021; Tang et al., 2023), we primarily apply the Cross Attention Augmented Transducer (CAAT) (Liu et al., 2021) as the decoder due to its state-of-the-art performance. CAAT extends the Recurrent Neural Network Transducer (RNN-T) (Graves, 2012) to support streaming tasks involving reordering, such as speech-to-text translation. For details of the CAAT architecture, please see (Liu et al., 2021).

Central to the design of CAAT is its goal to jointly optimize a READ/WRITE policy and a translation model. Latency management within the CAAT framework is facilitated through a hyper-parameter known as the decision step \( d \), which dictates that read-write decisions are made every \( d \) blocks. In the original CAAT framework, to achieve optimal performance, the \( d \) value selected during inference must match the value used during training, leading to the increased cost of training and maintaining multiple models to accommodate different latency requirements. To mitigate this challenge, we propose randomly selecting different \( d \) values for each mini-batch within a single training session, akin to the dynamic block size approach in pre-training discussed in Section 3.2. This strategy enables training a single fine-tuned model capable of meeting a wide range of latency demands.

In addition we also adapt wav2vec-S to other streaming models that use wav2vec 2.0 as a speech encoder to verify its generalization, as will be demonstrated in Section 4.4.

### 4 Experiments

#### 4.1 Experiments Settings

**Datasets** For continued pre-training, we use the 960 hours of unlabeled speech data from LibriSpeech\(^3\) (Panayotov et al., 2015) or the 60k hours of unlabeled speech data from LibriVox (Kahn et al., 2020). For streaming ASR fine-tuning, we use the 960 hours of labeled speech data from LibriSpeech dataset as the training set. For streaming ST fine-tuning, we use the MuST-C v1\(^4\) English-German (EnDe) and English-Spanish (EnEs) datasets (Di Gangi et al., 2019). Detailed data statistics are given in Appendix A. Our models are tuned on the development sets of LibriSpeech or MuST-C.

**Pre-Training Settings** We pre-trained two variants for wav2vec-S: BASE and LARGE, both of which adhere precisely to the model configurations of their wav2vec 2.0 counterparts. In terms of training settings, our models align with the wav2vec 2.0 (Baevski et al., 2020b) settings, with the sole modification being the reduction of the warmup steps to 5000 due to its initialization with wav2vec 2.0 pre-trained weights. Our implementation is based on Fairseq\(^5\) (Ott et al., 2019).

**Finetuning Settings** The streaming ASR and ST models share the same model configuration. As with our pre-trained model, we test two model configurations: BASE and LARGE. The BASE and LARGE models contain 6 and 12 Transformer decoder layer, respectively, and maintain consistency in the decoder hidden size with their respective pre-trained encoders. The total number of parameters for our pretrained and fine-tuned models see Appendix B. More training details see Appendix C.

**Baseline Models** We compare our method with several strong baselines that have a similar number of parameters.

CAAT (Liu et al., 2021) utilizes an RNN-T style architecture with transformer layers. CAAT BASE and LARGE models have similar hidden dimensions and layers to our model configurations, with

---

\(^3\)https://www.openslr.org/12

\(^4\)https://ict.fbk.eu/must-c

\(^5\)https://github.com/pytorch/fairseq
the exception that they do not employ pre-trained models learned from unlabeled data.

**MU-ST** (Zhang et al., 2022) trains an offline ST model and then optimizes a speech segmenter to learn an adaptive segmentation policy to detect meaningful units. MU-ST **BASE** is trained from scratch, and MU-ST **LARGE** uses the pre-trained wav2vec **LARGE** and mBART50 to initialize their model and then fine-tunes with external ST data.

**FAST** (Fu et al., 2023a) uses wav2vec 2.0 as the speech encoder to train an offline ST model and then introduces future-aware distillation and future-aware inference to mitigate the mismatch between offline training and streaming inference, adapting this offline model to streaming ST.

**MoSST** (Dong et al., 2022) applies wav2vec 2.0 as the speech encoder with a monotonic segmentation module to train an offline ST model for streaming inference.

**Evaluation** We average the checkpoints of the best 10 epochs on development set for evaluation (Tang et al., 2023; Fu et al., 2023a). We measure the ASR quality by the word error rate (WER) and apply a text normalizer (Radford et al., 2023) to standardize text before the WER calculation. We use SacreBLEU (https://github.com/mjpost/sacrebleu) to compute case-sensitive detokenized BLEU for the translation quality. The latency is evaluated with Average Latency (AL) (Ma et al., 2019) in the SimulEval (https://github.com/facebookresearch/SimulEval).

### 4.2 Streaming Results

**Streaming ST** We present the streaming ST results in Figure 1. Compared to the CAAT **BASE** and MU-ST **BASE** models without self-supervised pre-training, our **BASE** model achieves higher BLEU in all latency regions. There is an improvement of over 1 BLEU score in both EnDe and EnEs. Other methods, such as MoSST, using original wav2vec 2.0 as the speech encoder, show inferior translation quality in low and medium latency areas due to the mismatch between bidirectional pre-training and streaming inference. While FAST alleviates this mismatch to some extent, it still exhibits a significant performance gap compared to our method.

When scaling the models to the **LARGE** size, CAAT **LARGE** shows no improvement in translation quality for EnDe, whereas for EnEs, the improvement is marginal. This can be attributed to overfitting due to the absence of pre-training. MU-ST **LARGE**, which directly uses bidirectional wav2vec 2.0, only improves in high latency scenarios, with no gains in low latency. In contrast, our **LARGE** model shows significant improvements in all latency regions, with approximately 0.4 BLEU in EnDe and about 0.8 BLEU in EnEs.

**Streaming ASR** Figure 2 illustrates the results of streaming ASR on the LibriSpeech test-clean and test-other sets when finetuning on the LibriSpeech 960h labeled data. Our **BASE** slightly outperforms the CAAT **BASE** across all latency regions on test-clean set and achieves comparable performance on test-other set. The relatively small difference between two **BASE** mod-
Figure 2: ASR quality (WER) against the latency metrics (AL) on the LibriSpeech test sets when fine-tuning on the LibriSpeech 960h data.

Figure 3: ASR quality (WER) against the finetuning data size on the LibriSpeech test sets.

models may be attributed to two primary factors: firstly, the BASE model was pre-trained solely on the LibriSpeech 960h data, without introducing additional data compared to the fine-tuned data; secondly, the large quantities of fine-tuning data leveraged may mitigate performance gains.

Upon scaling to the LARGE configuration, which benefits from pre-training on a large-scale Libri-Light 60k hours of unlabeled data, we observe more substantial performance gains. Our LARGE model not only achieves lower WER compared to its BASE counterpart but also maintains a significant advantage over the CAAT LARGE model. However, we observe that the performance of the CAAT LARGE model becomes inferior when scaled up. As in Section 4.2, this is attributed to overfitting caused by the lack of pre-training on large-scale unlabeled data.

To further explore the second hypothesis regarding the influence of fine-tuning data size, Figure 3 compares the streaming ASR performance of our BASE and CAAT BASE models\(^{11}\) when fine-tuning on the 960h, 500h, 360h and 100h subsets of Librispeech. As the fine-tuning data is scaled down from the full 960h to a more constrained 100h, we witness a striking enhancement in the performance gains of our BASE model relative to the CAAT BASE. Notably, when fine-tuned with just 100h of data, our BASE model achieves a remarkable increase in WER by ~6 absolute points on the test-clean subset and ~10 absolute points on the test-other subset. These results suggest the effectiveness of our model in scenarios with limited fine-tuning data due to pre-training on large-scale unlabeled data.

### 4.3 Offline Results

Besides the streaming speech-to-text tasks, we also verify the capability of wav2vec-S on offline tasks. **Offline ST** We implement a baseline ST model Wav2vec2-Transformer following Ye et al. (2021), which integrates wav2vec 2.0 with a two-layer CNN and a Transformer. Our offline ST model follows the same configuration as the baseline, except the original wav2vec 2.0 component is replaced by our wav2vec-S to investigate its performance in a non-streaming setting, where the block size is set to 960ms (\(m = 640\text{ms}\) and \(r = 320\text{ms}\)). Table 1 reports the comparison results on MuST-C tst-COMMON set.

<table>
<thead>
<tr>
<th>Model</th>
<th>EnDe</th>
<th>EnEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fairseq ST (Wang et al., 2020a)</td>
<td>22.70</td>
<td>27.20</td>
</tr>
<tr>
<td>ESPnet-ST (Inaguma et al., 2020)</td>
<td>22.91</td>
<td>27.96</td>
</tr>
<tr>
<td>Wav2vec2-Transformer (Ye et al., 2021)</td>
<td>24.15</td>
<td>28.10</td>
</tr>
<tr>
<td>wav2vec-S-Transformer (Ours)</td>
<td>23.82</td>
<td>28.24</td>
</tr>
</tbody>
</table>

Table 2: The performance of offline ASR finetuned on LibriSpeech 960h labeled data and evaluated the LibriSpeech test sets without LM or lexicon. The results of baselines are obtained from the corresponding papers.

For a fair comparison, we try our best to set the AL of different data size to be approximately equal (AL$\approx$ 1000ms). The complete latency-quality curves for different data sizes are shown in the Appendix D.2.

\(^{11}\)Since our LARGE model has achieved significant gains, we only compare the performance of the BASE models in this experiment.
C EnDe and EnEs datasets. In comparison to Fairseq-ST and ESPnet-ST, which do not apply pre-training, wav2vec based models show significant better performance. When compared to the Wav2vec2-Transformer, our model shows a decrease in EnDe by a narrow margin of 0.33 BLEU but an increase for EnEs by 0.14 BLEU. We hypothesize that translation performance disparities among language pairs are due to syntactic variances in word order. Our wav2vec-S’s block-size self-attention, limited by a narrow contextual window, struggles with long-range dependencies. This limitation hampers translations into languages like German with subject-object-verb (SOV) order, unlike English’s subject-verb-object (SVO) order. However, for languages with similar SVO word order, like Spanish, wav2vec-S’s localized context suffices for precise translation. These results demonstrate that our wav2vec-S, even when applied in an offline setting, maintains competitive performance against established offline models.

**Offline ASR** We strictly follow wav2vec 2.0 (Baevski et al., 2020b) to fine-tune our offline ASR models on LibriSpeech 960h data with connectionist temporal classification (CTC) loss (Graves and Graves, 2012). The block size is consistent with the settings used in our offline ST model. Table 2 shows the performance of offline ASR models without a language model (LM) or lexicon in decoding. wav2vec-S BASE model exhibits an increase in Word Error Rate (WER) by 1.1 on the test-clean set and by 4.1 on the test-other set compared to wav2vec 2.0. This reduction in performance is largely due to the block-size self-attention mechanism used in wav2vec-S, which is tailored for streaming scenarios but struggles with modeling the long-term dependencies. When scaling up to the LARGE configuration and pretraining on more unlabel data, the performance gap narrows. Despite the decrease in performance when compared to wav2vec 2.0, it is imperative to recognize that wav2vec 2.0’s architecture is not inherently suited for streaming applications.

### 4.4 Adapting to Other Streaming ST Models

To validate the generalizability of wav2vec-S, we extended its application to other streaming ST methods, including MosST and ITST, which traditionally employ wav2vec 2.0 directly. Figure 4 presents a comprehensive comparison of ITST, MosST, and our adapted versions of these models, which we refer to as wav2vec-S-ITST and wav2vec-S-MosST, respectively. We employ AL represented by solid curves, and AL_CA depicted by dashed curves to measure ideal and computation-aware latency, respectively.

Our adapted models, wav2vec-S-MosST and wav2vec-S-ITST, outperform the original MosST and ITST models in both low and high latency settings (AL < 2s), with an improvement of approximately 4 BLEU for MosST and 1 BLEU for ITST. These results indicate that our adjustments to wav2vec 2.0 benefit streaming speech-to-text (ST) tasks. Considering computational latency (AL_CA), our models demonstrate a significant reduction in latency due to eliminating the need for re-encoding speech frames, unlike the original models. Overall, our experiments confirm that wav2vec-S delivers both high translation quality and reduced computational latency.

### 4.5 Ablation Study

To study the effectiveness of our methods, we compare our BASE model with several variants in the streaming ST task. The results are shown in Figure 5. All ablation results are evaluated on the MuST-C EnDe tst-COMMON set.

To validate the effectiveness of our proposed modifications, we sequentially remove three key modifications. The experimental results, as illustrated in Figure 5, include the following variants: "w/o FN modification" (removal of the feature normalization modification), "w/o FN and PE
modifications" (removal of both the feature normalization and positional encoding modifications), and "w/o any modifications" (removal of all three modifications). We observe that each variant performs significantly worse than "Ours" model in the low latency region. Moreover, the streaming translation performance progressively deteriorates as the modifications are incrementally removed. These results indicate that all three modifications must be implemented to adequately support streaming applications; otherwise, a training/inference mismatch problem may occur.

Subsequently, we investigate appropriate streaming settings to explore the effects of different model configurations. Specifically, we evaluate:

1. **The impact of dynamic versus fixed block size.** Our results, as shown in Figure 5, demonstrate that a dynamic block size ("Ours") can adapt to different latency settings without compromising performance, achieving consistent improvement over the "fixed block size" model.

2. **The effects of different training strategies.** We compare the approach of pre-training wav2vec-S from scratch versus continued pre-training with parameters initialized from wav2vec 2.0. The superior performance of "Ours" compared to the "pre-training from scratch" model, as illustrated in Figure 5, underscores the benefits of the continued pre-training strategy.

3. **The impact of using different pre-trained models for downstream streaming ST task.** In Figure 5, we contrast the performance of a streaming model (with the same encoder architecture as wav2vec-S) fine-tuned from wav2vec-S ("Ours") against models fine-tuned from wav2vec 2.0 ("fine-tuning w/ w2v2 init") or without a pre-trained model ("w/o pretraining"). The results show that "Ours" consistently outperforms others across all latency regions, demonstrating the advantage of pre-trained streaming models.

In summary, these results demonstrate the contribution of our pretraining method and the superiority of streaming pretraining for streaming tasks.

### 4.6 Analysis on Continued Pre-training

In this section, we investigate the pretraining efficiency of our continued pre-training strategy. Figure 6 illustrates the changes of BLEU scores and loss values with pre-training steps for continued pre-training and pre-training from scratch. Firstly, the continued pre-training approach exhibits a notably faster convergence rate in terms of loss reduction, signifying a more efficient optimization process. Moreover, the loss values associated with continued pre-training remain lower throughout the training process compared to pretraining from scratch. This suggests that continuous pre-training not only accelerates the convergence process, but also enhances the model’s overall optimization, leading to better performance. Additionally, the
continued pre-training strategy achieves a comparable BLEU score to the pre-training from scratch with significantly fewer training steps. For instance, at the 100k training steps, the continued pre-training model exhibits a BLEU score improvement that the pre-training from scratch only matches at a later stage, up to 400k steps. These results demonstrate the efficiency of our continued pretraining.

5 Related Works

Pre-trained Speech models Existing work can be broadly categorized by the pre-training objective into three main classes: contrastive learning-based models (Schneider et al., 2019; Baevski et al., 2020a,b), mask prediction-based models (Hsu et al., 2021; Chen et al., 2022; Baevski et al., 2022), and masked autoencoder-based models (Gong et al., 2022; Huang et al., 2022; Baevski et al., 2023; Chen et al., 2023). However, these methods cannot be effectively applied to streaming scenarios as they are essentially designed for offline tasks.

Streaming ST This method consists of a speech translation model and a read-write policy. Existing research primarily focuses on the exploration of the read-write policy, categorizing them into fixed (Ren et al., 2020; Ma et al., 2020b; Zeng et al., 2021; Papi et al., 2022; Wang et al., 2023) and adaptive policy (Liu et al., 2021; Zhang and Feng, 2022, 2023b; Papi et al., 2023a;b; Zhang et al., 2023a; Zhao et al., 2023; Chen et al., 2024). Additionally, recent studies leverage the wav2vec 2.0 model to enhance the quality of the ST model (Dong et al., 2022; Zhang and Feng, 2022, 2023a;b; Zhang et al., 2023b; Yang et al., 2023). While significant performance improvements are achieved at higher latency region, this integration has resulted in poorer performance at lower and medium latency regions due to the mismatch between offline training and streaming inference (Fu et al., 2023a). To this end, some research efforts introduce various distillation losses (Fu et al., 2023a) or multi-stage training objectives (Zhang et al., 2023a), which have been effective in mitigating the mismatch to some extent.

Streaming ASR Current methods can be broadly classified into transducer-based approaches, such as RNN-T (Graves, 2012), Neural Transducer (Jaitly et al., 2015), and Transformer Transducer (Yeh et al., 2019), and local attention-based methods, including monotonic attention (Raffel et al., 2017; Ma et al., 2020c), chunk/block-wise attention (Chiu and Raffel, 2018; Zhang et al., 2020; Wang et al., 2020b). Additionally, some studies apply wav2vec 2.0 or large amounts of unlabeled data to streaming ASR (Cao et al., 2021; Yang et al., 2022; Shim et al., 2023; Fu et al., 2023b). However, such approaches often involve complex training procedures and require intricate balancing of multiple objectives, limiting their application to other streaming tasks.

Distinctively, wav2vec-S, through streaming pre-training, addresses above issues, achieving significant improvements across all latency regions.

6 Conclusion

In this paper, we propose a streaming pre-trained speech model, wav2vec-S, which is effectively adapted from the pretrained wav2vec 2.0 model for streaming tasks through simple architectural modifications and continued pretraining. Experiments on both streaming ASR and ST tasks show the superiority of wav2vec-S in terms of performance, pre-training and inference efficiency.

Acknowledgements

We would like to thank all the anonymous reviewers for the insightful and helpful comments. This work is supported by National Science and Technology Major Project (Grant No. 2022ZD0116101), the Key Support Project of NSFC-Liaoning Joint Foundation (Grant No. U1908216), and the Project of Research and Development for Neural Machine Translation Models between Cantonese and Mandarin (Grant No. WT135-76) and the Major Scientific Research Project of the State Language Commission in the 13th Five-Year Plan (Grant No. WT135-38), University-Industry Cooperation Programs of Fujian Province of China (No. 2023H6001), and Alibaba Research Intern Program.

Limitation

Our method primarily focus on adapting the wav2vec 2.0 framework for streaming speech inputs. While wav2vec 2.0 serves as a robust foundation due to its proven effectiveness in various speech-related tasks, our approach does not extend to other pre-trained speech models such as HUBERT (Hsu et al., 2021). However, these pretrained models share an identical or similar Transformer architecture as wav2vec 2.0. Thus, the principles behind the modification strategies we applied to
wav2vec 2.0 can also be extended to these models, with necessary changes in some cases. We focus on wav2vec 2.0 serves as a case study to illustrate the applicability and effectiveness of adapting pretrained models for streaming applications. Additionally, the lack of computing resources also prevented us from conducting comprehensive studies using other pretrained models. We found that focusing on wav2vec 2.0 strikes a good balance as it allows us to directly compare with baseline methods that are based on wav2vec 2.0. We leave the extension to other pre-trained models for future research.

Our evaluation of the proposed wav2vec-S model is confined to ASR and ST tasks. While these tasks are critical benchmarks for spoken language understanding, our method has not been validated across a wider spectrum of speech-related tasks, such as speech-to-speech task.

References


A Data Statistics

We evaluate our model on MuST-C V1 English-German (EnDe) and English-Spanish (EnEs), and LibriSpeech 960h datasets. For ST data, following (Dong et al., 2022; Papi et al., 2023b; Fu et al., 2023a), we filter out short speech of less than 1000 frames (62.5ms) and long speech of more than 480,000 frames (30s) in the training set. The data statistics are illustrated in Table 3.

<table>
<thead>
<tr>
<th>split</th>
<th>EnDe</th>
<th>EnEs</th>
<th>LibriSpeech</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>225,271</td>
<td>260,041</td>
<td>281,242</td>
</tr>
<tr>
<td>dev</td>
<td>1,418</td>
<td>1,312</td>
<td>2,703 (clean) 2,864 (other)</td>
</tr>
<tr>
<td>test</td>
<td>2641</td>
<td>2502</td>
<td>2620 (clean) 2939 (other)</td>
</tr>
</tbody>
</table>

Table 3: Number of samples for each split of finetuning data.

B The Number of Parameters of The Models

The number of parameters of wav2vec-S and the fine-tuned model are provided in Table 4 and 5.

<table>
<thead>
<tr>
<th></th>
<th>BASE</th>
<th>LARGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>strides</td>
<td>5, 2, 2, 2, 2, 2, 2</td>
<td>5, 2, 2, 2, 2, 2</td>
</tr>
<tr>
<td>kernel width</td>
<td>10, 3, 3, 3, 3, 2, 2</td>
<td>10, 3, 3, 3, 3, 2, 2</td>
</tr>
<tr>
<td>channel</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Encoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>layer</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>embedding dim.</td>
<td>768</td>
<td>1024</td>
</tr>
<tr>
<td>inner FFN dim.</td>
<td>3072</td>
<td>4096</td>
</tr>
<tr>
<td>attention heads</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Projection</td>
<td>dim.</td>
<td>256</td>
</tr>
<tr>
<td>Number of Parameters</td>
<td>91M</td>
<td>309M</td>
</tr>
</tbody>
</table>

Table 4: Model architecture summary for wav2vec-S BASE and LARGE models

C Additional Training Details

For speech data, we use the raw 16-bit 16kHz mono-channel audio wave. For text data, the SentencePiece\(^{14}\) model (Kudo and Richardson, 2018) is used to learn a unigram vocabulary of size 10k. We use an Adam optimizer with \(\beta_1 = 0.9, \beta_2 = 0.98\). For streaming ST finetuning, the learning rate is set to 2e-4 for the BASE and LARGE models. We warm up the learning rate for the first 4k steps and then decay it with the inverse square root schedule. For streaming ASR finetuning, we set learning rate to 5e-4 and 1e-4 for the BASE and LARGE models, respectively. Following wav2vec 2.0 (Baevski et al., 2020b), we use a tri-state rate schedule that initially warms up the learning rate during the first 10% of updates, maintains it at a steady level for the subsequent 30%, and then applies a linear decrease for the remaining updates, and wav2vec-S is not updated in the first 10k steps. The label smoothing and dropout rate are set to 0.1 for robust training. We train on 8 A100 GPUs with the max speech tokens of 1,400,000 (87.5 seconds) per GPU for BASE model, and the max speech tokens of 800,000 (50 seconds) per GPU for LARGE model. If the loss on the dev set does not decrease over ten epochs, the training process will be stopped early. Both models are trained for about 30h, where the LARGE model converges faster.

D Additional Results

D.1 Comparison with EDAtt

The baselines we compared in Section 4.2 do not leverage additional machine translation (MT) data, which can typically provide further improvement. EDAtt (Papi et al., 2023b), one of the strong baselines, utilizes an MT model trained on the OPUS dataset for knowledge distillation, making a direct comparison unfair. Therefore, we opted not to include a direct comparison with EDAtt in Section 4.2. Despite not utilizing additional MT data, our model still significantly outperforms EDAtt in low-latency regions, with improvements of about 6 BLEUs on both the EnDe and EnEs datasets, as indicated in the Figure 7. Moreover, Our model achieves comparable performance to EDATT in medium to high latency areas.

D.2 Latency-Quality Curves for Different Data Sizes

We present the latency-quality curves for different fine-tuned data sizes in Figures 8, 9, and 10.

\(^{14}\)https://github.com/google/sentencepiece
<table>
<thead>
<tr>
<th>encoder layer</th>
<th>decoder layer</th>
<th>embed dim</th>
<th>inner FFN dim</th>
<th>#Params (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our BASE</td>
<td>12</td>
<td>6</td>
<td>768</td>
<td>183</td>
</tr>
<tr>
<td>Our LARGE</td>
<td>24</td>
<td>12</td>
<td>1024</td>
<td>622</td>
</tr>
<tr>
<td>CAAT BASE</td>
<td>12</td>
<td>6</td>
<td>768</td>
<td>182</td>
</tr>
<tr>
<td>CAAT LARGE</td>
<td>24</td>
<td>12</td>
<td>1024</td>
<td>620</td>
</tr>
<tr>
<td>MU-ST BASE</td>
<td>12 + 12</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MU-ST LARGE</td>
<td>24 + 12</td>
<td>12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FAST</td>
<td>12 + 8</td>
<td>6</td>
<td>768</td>
<td>202</td>
</tr>
<tr>
<td>MoSST</td>
<td>12 + 8</td>
<td>6</td>
<td>768</td>
<td>202</td>
</tr>
</tbody>
</table>

Table 5: Model architecture summary for our finetuning and baseline models. MU-ST additionally contains a 12-layer Transformer layer for speech Segmentor. FAST and MoSST additionally include an 8-layer Transformer layer semantic encoder.

D.3 Numeric Results for the Figures

We also provide the numeric results for Figures 1 and 2 in Tables 6.

Figure 7: The translation quality (BLEU) against the latency metrics (AL) on the test-COMMON set of MuST-C EnDe and EnEs datasets.

Figure 8: ASR quality (WER) vs. latency metrics (AL) on LibriSpeech 500h subset.

Figure 9: ASR quality (WER) vs. latency metrics (AL) on LibriSpeech 360h subset.

Figure 10: ASR quality (WER) vs. latency metrics (AL) on LibriSpeech 100h subset.

11479
<table>
<thead>
<tr>
<th>Model</th>
<th>d</th>
<th>AL</th>
<th>AP</th>
<th>DAL</th>
<th>BLEU</th>
<th>AL</th>
<th>AP</th>
<th>DAL</th>
<th>BLEU</th>
<th>AL</th>
<th>AP</th>
<th>DAL</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AL</td>
<td>AP</td>
<td>DAL</td>
<td>WER</td>
<td>AL</td>
<td>AP</td>
<td>DAL</td>
<td>WER</td>
<td>AL</td>
<td>AP</td>
<td>DAL</td>
<td>WER</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>683</td>
<td>0.68</td>
<td>1327</td>
<td>23.08</td>
<td>550</td>
<td>0.67</td>
<td>1340</td>
<td>27.91</td>
<td>644</td>
<td>0.60</td>
<td>1205</td>
<td>6.37</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1101</td>
<td>0.74</td>
<td>1799</td>
<td>29.06</td>
<td>928</td>
<td>0.73</td>
<td>1796</td>
<td>29.06</td>
<td>978</td>
<td>0.65</td>
<td>1704</td>
<td>4.69</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2140</td>
<td>0.86</td>
<td>3052</td>
<td>25.88</td>
<td>2000</td>
<td>0.85</td>
<td>3111</td>
<td>30.00</td>
<td>2086</td>
<td>0.79</td>
<td>3332</td>
<td>3.81</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3518</td>
<td>0.95</td>
<td>4385</td>
<td>26.48</td>
<td>3471</td>
<td>0.94</td>
<td>4530</td>
<td>30.15</td>
<td>3823</td>
<td>0.91</td>
<td>5164</td>
<td>3.52</td>
</tr>
<tr>
<td>Our LARGE</td>
<td>2</td>
<td>711</td>
<td>0.68</td>
<td>1336</td>
<td>22.64</td>
<td>555</td>
<td>0.67</td>
<td>1339</td>
<td>26.95</td>
<td>701</td>
<td>0.61</td>
<td>1258</td>
<td>6.11</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1119</td>
<td>0.74</td>
<td>1795</td>
<td>24.49</td>
<td>941</td>
<td>0.73</td>
<td>1800</td>
<td>28.17</td>
<td>1024</td>
<td>0.66</td>
<td>1737</td>
<td>4.57</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2157</td>
<td>0.86</td>
<td>3056</td>
<td>25.43</td>
<td>1993</td>
<td>0.85</td>
<td>3101</td>
<td>29.23</td>
<td>2103</td>
<td>0.79</td>
<td>3345</td>
<td>3.93</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3528</td>
<td>0.95</td>
<td>4385</td>
<td>26.05</td>
<td>3470</td>
<td>0.94</td>
<td>4523</td>
<td>29.40</td>
<td>3826</td>
<td>0.91</td>
<td>5169</td>
<td>3.90</td>
</tr>
<tr>
<td>Our BASE</td>
<td>2</td>
<td>769</td>
<td>0.69</td>
<td>1363</td>
<td>21.51</td>
<td>544</td>
<td>0.67</td>
<td>1334</td>
<td>25.95</td>
<td>753</td>
<td>0.62</td>
<td>1322</td>
<td>6.29</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1152</td>
<td>0.74</td>
<td>1812</td>
<td>22.60</td>
<td>940</td>
<td>0.73</td>
<td>1797</td>
<td>27.31</td>
<td>1070</td>
<td>0.67</td>
<td>1776</td>
<td>5.20</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2175</td>
<td>0.86</td>
<td>3055</td>
<td>23.63</td>
<td>1989</td>
<td>0.85</td>
<td>3102</td>
<td>28.06</td>
<td>2134</td>
<td>0.80</td>
<td>3350</td>
<td>4.66</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3524</td>
<td>0.95</td>
<td>4375</td>
<td>23.78</td>
<td>3455</td>
<td>0.94</td>
<td>4522</td>
<td>28.32</td>
<td>3845</td>
<td>0.91</td>
<td>5167</td>
<td>4.50</td>
</tr>
<tr>
<td>CAAT LARGE</td>
<td>2</td>
<td>803</td>
<td>0.69</td>
<td>1398</td>
<td>21.66</td>
<td>582</td>
<td>0.68</td>
<td>1362</td>
<td>25.61</td>
<td>705</td>
<td>0.61</td>
<td>1270</td>
<td>6.19</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1172</td>
<td>0.75</td>
<td>1826</td>
<td>22.72</td>
<td>965</td>
<td>0.73</td>
<td>1811</td>
<td>26.96</td>
<td>1031</td>
<td>0.66</td>
<td>1744</td>
<td>4.88</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2170</td>
<td>0.86</td>
<td>3062</td>
<td>23.82</td>
<td>2009</td>
<td>0.85</td>
<td>3105</td>
<td>27.64</td>
<td>2107</td>
<td>0.79</td>
<td>3340</td>
<td>4.25</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>3515</td>
<td>0.95</td>
<td>4379</td>
<td>24.09</td>
<td>3469</td>
<td>0.94</td>
<td>4523</td>
<td>27.95</td>
<td>3829</td>
<td>0.91</td>
<td>5166</td>
<td>3.96</td>
</tr>
</tbody>
</table>

Table 6: Numeric results on MuST-C EnDe and EnEs tst-COMMON sets, and LibriSpeech test-clean and test-other sets for our models and CAAT models (Figure 1).