



**HAL**  
open science

## Fine-tuning strategies for faster inference using speech self-supervised models: a comparative study

Salah Zaiem, Robin Algayres, Titouan Parcollet, Slim Essid, Mirco Ravanelli

► **To cite this version:**

Salah Zaiem, Robin Algayres, Titouan Parcollet, Slim Essid, Mirco Ravanelli. Fine-tuning strategies for faster inference using speech self-supervised models: a comparative study. ICASSP 2023 - International Conference on Acoustics, Speech, and Signal Processing, Jun 2023, Rhodes, Greece. hal-04076307

**HAL Id: hal-04076307**

**<https://hal.science/hal-04076307v1>**

Submitted on 20 Apr 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# FINE-TUNING STRATEGIES FOR FASTER INFERENCE USING SPEECH SELF-SUPERVISED MODELS: A COMPARATIVE STUDY

Salah Zaiem<sup>\*,‡</sup> Robin Algayres<sup>◇</sup> Titouan Parcollet<sup>†</sup> Slim Essid<sup>\*</sup> Mirco Ravanelli<sup>‡</sup>

<sup>\*</sup> Telecom Paris, Palaiseau, France, <sup>◇</sup> ENS, INRIA, INSERM, UPEC, PSL Research University, France

<sup>†</sup> Samsung AI Center, Cambridge, United-Kingdom,

<sup>‡</sup> Mila-Quebec AI Institute, Université de Montréal, Concordia University, Canada

## ABSTRACT

Self-supervised learning (SSL) has allowed substantial progress in Automatic Speech Recognition (ASR) performance in low-resource settings. In this context, it has been demonstrated that larger self-supervised feature extractors are crucial for achieving lower downstream ASR error rates. Thus, better performance might be sanctioned with longer inferences. This article explores different approaches that may be deployed during the fine-tuning to reduce the computations needed in the SSL *encoder*, leading to faster inferences. We adapt a number of existing techniques to common ASR settings and benchmark them, displaying performance drops and gains in inference times. Interestingly, we found that given enough downstream data, a simple downsampling of the input sequences outperforms the other methods with both low performance drops and high computational savings, reducing computations by 61.3% with an WER increase of only 0.81. Finally, we analyze the robustness of the comparison to changes in dataset conditions, revealing sensitivity to dataset size.

*Index Terms*— Speech recognition, self-supervised learning.

## 1. INTRODUCTION

Self-supervised learning (SSL) has emerged as the main approach for leveraging unlabelled data to achieve significant performance improvements in a wide range of downstream tasks. This is particularly beneficial as it reduces the need for expensive and imprecise manual annotation. Various approaches have been introduced in the literature, including predictive coding, [1], multi-task learning [2, 3], auto-encoding techniques [4] or contrastive learning [5].

However, recent trends in SSL for speech have shown that the improvements in terms of performance are often driven by larger architectures, leading to potentially long inference times [6]. For instance, Sanyuan et al. [6], have shown that switching from WavLM Large to WavLM Base halved the observed word error rate (WER) on a held-out English ASR task. Preserving reasonable inference times while increasing the capacity of the model is of critical interest to maximize the impact of this new technology on real-life products.

As a matter of fact, several approaches have been proposed to shorten inference times using SSL models. Some attempted to distill state-of-the-art models by using shallower or thinner networks [7, 8] or through downsampling the inputs [9, 10]. However, while the downstream performance of distilled student models is comparable to larger teacher models on most speech classification tasks, a large gap is still witnessed for more complex tasks such as ASR [11]. Also, low-bit quantization during pretraining has recently emerged as a successful approach for faster inference times [12]. Compared

to our proposed methods, these two approaches bear the advantage of leading to generalist models usable for multiple downstream tasks. However, they have two major downsides. First, they necessitate access to the very large pretraining dataset, which may or may not be publicly and commercially available, such as for recent and large-scale state-of-the-art speech recognition models [13]. Second, even if the pretraining set is available, applying quantization or distillation to these large models remains a particularly challenging and costly task due to the original dataset and model sizes. For instance, academic attempts for distilling SSL models have been solely applied to Base models (*i.e.* less than 100M parameters), and are generally restricted to a thousand hours of speech pretraining data, compared to 94k hours for 317M parameters WavLM Large.

In contrast, by focusing on a given downstream task of interest, our approach allows for reducing the inference time using off-the-shelf large self-supervised models, directly training only on the small considered downstream dataset. This is achieved by benefiting from the fine-tuning phase to reduce the inference time. More precisely, this article considers three families of techniques originating from the recent SSL literature: i) layer dropping or replacement, ii) early-exiting, and iii) input sequence downsampling. The fine-tuning phase allows the models to adapt to a shrunk architecture, or shorter inputs while maximizing the downstream performance. Thus, our contributions are twofold: i) We compare a number of recent shrinking approaches (Section 2) used while fine-tuning large SSL models on an ASR task providing inference speed metrics, and ii) we analyze the robustness of these methods to changes in the dataset and in the annotated data quantity (Section 3). The SpeechBrain code base is released for replication and further advancements [14].<sup>1</sup>

## 2. SETTING AND METHODS

This section outlines the global setting for comparing the considered techniques before providing a detailed description of the approaches and their motivations.

### 2.1. Benchmarking setting

The study is conducted under the two strong yet realistic following conditions. First, we suppose that we do not have access to the pretraining dataset that would enable, for instance, extensive distillation or quantization approaches. Second, we limit ourselves to using only the annotated training data of the target dataset during fine-tuning, eliminating transfer-learning based approaches. The second condition is relevant when the target domain is rare and specific enough

<sup>1</sup>[https://github.com/salah-zaiem/speeding\\_inferences](https://github.com/salah-zaiem/speeding_inferences)

not to take advantage of transfer learning from classic large annotated datasets. This is generally true in two popular cases where using self-supervised models is privileged: low-resourced languages and speech datasets with specific acoustic conditions [15].

We use the released pre-trained and non fine-tuned WavLM Large [6] as the SSL model, as it tops speech self-supervised learning benchmarks and exhibits resilience to noisy conditions [11]. In all the experiments of this section, we use the *train-clean-100* split of LibriSpeech [16] as our training set, the *dev-clean* split for validation and finally the *test-clean* split for testing. Following common practices [17], we freeze the convolutional front-end and only fine-tune the transformers part of WavLM Large consisting of 24 transformers layers. The self-supervised *encoder* outputs a frame vector of dimension 1,024 for every segment of 320 speech samples which corresponds in the case of a 16-kHz sampling rate to 20 ms of audio signal. Two fully connected layers with a hidden size of 1,024 map each frame vector to the probabilities of the considered characters. Connectionist Temporal Classification (CTC) [18] loss is used for training.

During inference, the decoding of the probabilities of the characters is completed in two ways: with or without using a Language Model (LM). In the experiments labeled as *Without LM*, greedy decoding is applied, outputting the character with the maximal probability at each step before applying CTC-based reformatting to get the predicted words. In the *With LM* experiments, we use the Librispeech official 4-gram language model, trained using the KenLM library [19], and decode the sentence using shallow fusion [20] considering the language modelling score of a beam of the most acoustically probable sequences. An n-gram model is chosen over more complex language modelling approaches to reduce word error rates while keeping low inference times. This is done using the PyCTCDecode<sup>2</sup> library with default parameters. To understand the impact of this aspect on the results tables, it is crucial to note that this decoding is performed on CPU and that the decoding time is prolonged when the model is uncertain about its predictions. Indeed, PyCTCDecode proceeds to prune elements of the beam that are scored too low by the language model compared to the maximal beam score. It leads in this work to a penalty for models with high error rates before the LM addition, as they systematically had longer decoding times. With the stage of the comparison set, we will proceed to the descriptions of the selected candidates

## 2.2. Layer dropping and replacement

With multiple studies on layer-wise probing of self-supervised models showing that phonetic content is divided among the layers of the transformers [21], removing layers has emerged as a possibility for faster inferences. Experiments led mainly on text language models have shown that dropping higher level layers is preferable to avoid heavy performance drops [22]. In a first experiment, we will study the effect of fine-tuning the SSL model after having removed a number of layers. In a second one, given the fact that WavLM has been trained with *layerdrop* [23], *i.e.* random layer omission during training, we fine-tune it with *layerdrop* probability equal to  $q = 0.5$  and study the effect of keeping various *layerdrop* rates during test.

## 2.3. Early-exiting

Similarly, early-exiting is a relevant approach to reduce computations during inference [24, 25]. It consists in allowing the model to

use an early layer representation and feed it directly to the decoder, saving the computation of further layers. During fine-tuning, starting from the twelfth layer, a specific downstream decoder is learned on top of every layer. During inference, a heuristic metric computed after each layer indicates whether the model should output a decoded sequence using the current layer or go further. Given well calibrated heuristics, early-exiting should reduce the mean exit layer and, thus, the mean inference time. Furthermore, studies on what has been called “overthinking” [24] have shown that SSL models could benefit from early exiting both inference time and performance. Inspired by previous works, two heuristics are tested: the entropy of the decoder outputs, and a measure of similarity between the representations of consecutive layers.

As described in Section 2.1, each downstream decoder consists of two linear layers outputting logit probabilities after each time frame of 20 ms. Each layer  $i$  of the transformer outputs a representation  $R_i$  of size  $[N, A]$  with  $N$  the number of signal time frames and  $A$  the hidden dimension of WavLM Large ( $A = 1,024$ ). Operating on this representation through the decoder  $D_i$ , the vectors of logits probabilities  $L_i$  are of size  $[N, P]$  with  $P$  the number of different characters in the dataset (in our case  $P = 31$ ). The entropy  $E_i$  of the output of layer  $i$  is defined as:

$$E_i = -\frac{1}{NP} \sum_{j=1}^N \sum_{k=1}^P L_{i,j,k} \log(L_{i,j,k}). \quad (1)$$

with  $L_{i,j,k}$  being the probability of the character  $k$  at time frame  $j$  predicted by decoder  $D_i$ . During fine-tuning, to learn the weights of every decoders weights, we pass through all the layers of the model and sum the CTC losses over the outputs of all decoders. During inference, after each layer  $i$  starting from the twelfth,  $R_i$  is decoded and  $E_i$  is computed. If  $E_i$  is lower than a fixed entropy threshold, we do not go further in the SSL transformer, and decode the logit probabilities into words. Hence, reducing the entropy threshold increases the confidence required for exiting, leading systematically in this work to later exits and thus, higher inference times. The second exiting heuristics is layer representation similarity. For each layer  $i \geq 12$ , we compute the cosine similarity between  $R_i$  and  $R_{i-1}$ . Similarly to the first approach, if the similarity is higher than a fixed threshold at layer  $i$ , the model exits to  $D_i$  and decodes the logits into a word sequence. This second approach is slightly faster as it does not involve computing the decoding into logits at each layer.

## 2.4. Sequence Downsampling

Inspired by works on distilling speech models with smaller sampling rates [8, 10], and given the quadratic memory bottleneck of transformers architectures as a function of input lengths, we assess the capacity of the SSL model, trained on 16-kHz audio inputs, to adapt to lower sampling rates. Given a speech file  $x$  consisting in  $T$  speech samples  $x = (x_i)_{i \in [1, T]}$  and a downsampling factor  $k$ , a function  $f$ , learned or unlearned depending on the chosen method, downsamples  $x$  to  $x' = f(x) = (x'_i)_{i \in [1, \lfloor T/k \rfloor]}$ , a sequence of size  $\lfloor T/k \rfloor$ . The downsampled sequence  $x'$  is then fed to the SSL feature extractor instead of  $x$ . Three methods for downsampling the input sequences are evaluated. The first one is signal decimation (*i.e.* classic signal downsampling). Second, we test a learned downsampling strategy, through a one-dimensional (1D) convolution layer of kernel size 160 and a stride equal to the downsampling factor, ran on the input waveform. Finally, we test an averaging 1D downsampling with a fixed window of size 16 (*i.e.* a constant convolution).

<sup>2</sup><https://github.com/kensho-technologies/pyctcdecode>

Technique		WER ↓	GPU (s)	CPU (s)	WER-LM ↓	GPU-LM (s)	CPU-LM (s)	MACs (G)
<b>Baseline</b>	<b>Full Model</b>	4.09	134	1121	3.31	152	1128	386.53
<b>Layer Drop</b>	<b>Drop Prob</b>							
	0.5	11.28	96	721	5.89	156	776	244.19
	0.4	8.32	102	816	4.58	145	844	272.28
	0.3	6.56	109	888	3.84	157	913	300.98
	0.25	5.91	113	932	3.72	148	950	314.24
<b>Layer Removal</b>	<b>Num. Kept Layers</b>							
	12	14.39	93	726	8.64	127	739	236.64
	16	8.16	109	852	5.53	131	861	286.60
	20	5.14	117	988	3.62	142	989	336.57
<b>Early Exit : Entropy Threshold</b>	<b>Mean Exit Layer</b>							
0.06	13.80	12.08	96	757	9.25	122	765	252.36
0.03	17.61	7.67	116	974	6.55	137	976	326.28
0.025	20.52	6.66	128	1127	5.87	149	1132	364.92
0.01	23.98	6.20	142	1275	5.49	165	1280	386.53
<b>Early Exit : Layer Sim. Threshold</b>	<b>Mean Exit Layer</b>							
0.92	15.97	10.23	99	812	8.17	123	819	274.11
0.95	17.18	8.78	104	850	7.35	126	864	291.68
0.965	21.44	6.79	120	1070	5.93	131	1073	358.85
0.98	24.00	6.20	1280	1153	5.49	149	1153	386.51
<b>Two Steps EE : Layer Sim. Threshold</b>	<b>Mean Exit Layer</b>							
0.96	14.52	21.95	102	866	8.75	180	938	285.68
0.97	21.46	6.17	126	1138	4.34	152	1167	382.00
0.98	23.0	4.54	130	1175	3.87	151	1196	386.54
<b>Downsampling Technique</b>	<b>Downsampling Factor</b>							
<b>Convolutional Downsampling</b>	2	4.61	84	582	3.48	98	600	192.97
	3	<b>5.47</b>	<b>69</b>	<b>414</b>	<b>4.12</b>	<b>91</b>	<b>436</b>	<b>134.86</b>
	4	21.88	67	335	14.60	106	340	96.11
<b>Averaging Downsampling</b>	2	4.93	80	570	3.66	98	578	192.97
	3	6.01	64	406	4.27	90	422	134.86
	4	26.84	60	326	18.02	115	385	96.11
<b>Signal Downsampling</b>	2	4.85	86	569	3.58	97	575	192.97
	3	5.83	72	427	4.08	89	458	134.86
	4	16.08	63	330	11.10	97	369	96.11
<b>DistilHuBERT</b>	Linear Decoder	30.74	56	240	16.20	130	311	101.74
	BiLSTM Decoder	16.30	95	545	10.57	128	613	161.06

**Table 1.** Word error rates and inference metrics on LibriSpeech *test-clean* split for the considered approaches and various parameters per method. All models are finetuned on LibriSpeech *train-clean-100*. “GPU” and “CPU” indicate the inference times in seconds on GPU and CPU. “-LM” suffixes indicate that the decoding uses a language model. “Drop Prob” is the probability of randomly dropping layers during inference. Early-exiting experiments come with an exiting threshold and a resulting mean exit layer computed over the test set.

Each one of the techniques is evaluated with 3 downsampling factors: 2, 3 and 4. For instance, this corresponds for the first approach to downsampling the signals from 16 kHz to, respectively, 8000, 5333, and 4000 Hz. As explained in Section 2.1, the SSL model outputs a character every 320 audio samples, which corresponds to 20 ms of audio with a 16-kHz sampling rate. With lower sampling rates, the number of output characters may become lower than the lengths of the corresponding textual sequences. This is why, when dealing with downsampling factors 3 and 4, the size of the decoder output layer is doubled. It is then reshaped to fit the number of considered characters, before being fed to the classifier softmax function. This allows every frame of audio to output two characters instead of one.

### 3. RESULTS AND ROBUSTNESS STUDY

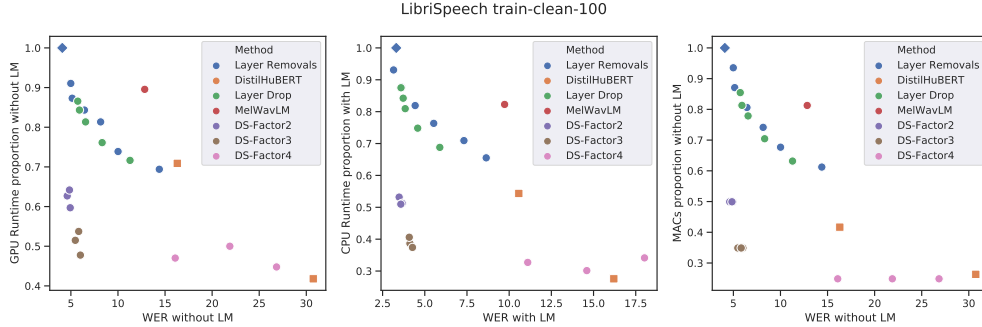
Table 1 shows the results obtained with the different techniques. Reported GPU inference times are for a Nvidia Tesla V100 SXM2 32 Go GPU, while CPU inferences are using one Intel Cascade Lake 6248 processor with a 27.5 MB cache and 2.50 GHz clock speed. The inference times and MACs are those for running inference on the whole 5.4 hours of the *test-clean* split.

**Layer removals.** The results of the layer removal and dropping approaches are displayed in the upper part of Table 1. Surprisingly, for a given proportion of layers dropped, keeping the layerdrop performs better than training with the reduced number of layers. For instance, randomly dropping 50% of the layers during the test leads to 11.28 of WER, compared to 14.39 when dropping the last 12 layers during fine-tuning (*i.e.* again 50% of the layers). It suggests that while training systematically with the same layers adapts the models

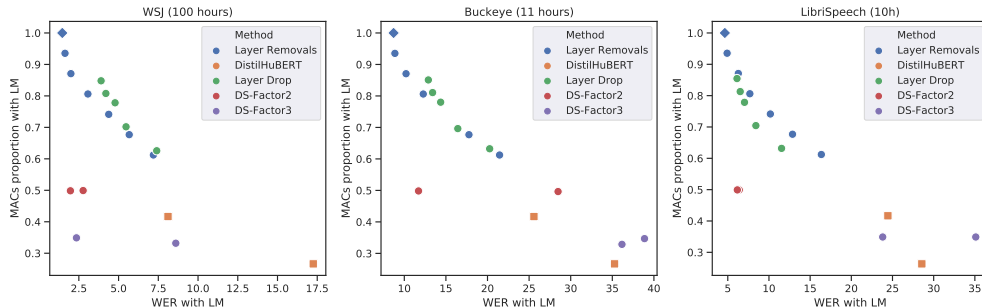
directly to inference conditions, removing the information contained in the last layers of the model harms too severely the performance.

**Early-exiting.** The middle part of Table 1 shows the obtained results using early-exiting with different values of entropy and similarity thresholds. Increasing the entropy threshold (or decreasing the similarity one) leads naturally to earlier exits (lower “Mean Exit Layer” values) and reduced inference times but higher WERs. Results show that using our two considered heuristics to control exiting does not prevent significant performance drops in the case of earlier exits. We can also observe that even when using the whole network (*i.e.* low entropy cases), this technique leads to lower performances compared to the full model trained without early exiting. We suggest the following explanation: since early exits encourage the model to push the phonetic content required for decoding towards early layers, it undermines the ability of the model to learn hierarchical features, ultimately resulting in poorer performance even when exiting later in the model. To verify this explanation, we propose a two-step approach where the SSL model weights are first fine-tuned without early-exits, before freezing them when learning the early-exit decoders. As shown in Table 1, this leads to good performances in case of late-exiting, but with the cost of steeper drops when exiting earlier. This suggests that successful early-exiting should decouple hierarchical feature extraction and decoding preparation. In both cases, early exiting lags behind layer-removal techniques in terms of ratio inference gains / performance drop.

**Downsampling.** Results of the downsampling experiments are shown in the last part of Table 1. Downsampling by factors 2 and 3 lead to high gains in inference times without significant drops in performance. For instance, compared to running the full model,



**Fig. 1.** WER and inference metrics with or without language modelling for the presented techniques fine-tuned on LibriSpeech-100h. The best techniques, characterized by both low Word Error Rates (WERs) and inference times, are Factor2 and Factor3 downsamplings, located in the bottom left of the figures. The full model is indicated by a blue diamond, while DistilHubert baselines are represented by orange squares. Inference time measurements are shown as a proportion of the measure done with the full model.



**Fig. 2.** WER with LM decoding and MACs for the considered methods on WSJ, Buckeye and LibriSpeech-10h sets. While WSJ exhibits results similar to LibriSpeech, reducing the quantity of fine-tuning data causes significant performance drops for the downsampling methods.

signal downsampling with a factor 3 using a language model for decoding, leads to 61.34% relative CPU inference time reduction, with an absolute increase of only 0.81 in WER. Downsampling with factor 4, while naturally leading to further gains in inference times, results in intolerable performance costs. The three considered downsampling strategies are very similar in terms of error rates and computational savings, with a slight advantage for convolutional downsampling when sequences lengths are reduced with factors 2 and 3. For comparison with baselines, we add two experiments using DistilHuBERT [7]. When using the simple linear decoder used in this benchmark, DistilHuBERT shows performances largely below the ones in the original paper. For a fair comparison, we produced an experiment with a BiLSTM decoder. While improving largely the performance, this comes at a high cost in terms of inference times.

Figure 1 presents a visual comparison between all the presented methods. Clearly, factor 2 and 3 downsampling techniques are the best performing methods with low WER, jointly with low GPU and CPU inference times. While being the fastest, higher downsampling factors and DistilHuBERT suffer from high error rates.

### 3.1. Robustness to changes in the downstream dataset

Finally, we test the robustness of these conclusions to changes in the characteristics of the downstream dataset. Three datasets are considered. We tested the same methods with a 100-hour subset of the Wall Street Journal (WSJ) dataset [26]<sup>3</sup>. We also test the robustness

<sup>3</sup>We combined WSJ0 and WSJ1, 70-hour long each, and removed all utterances that contain non-letter symbols in their transcriptions. Then, we extracted a 100-hour random subset of the remaining sentences

of the approach to dataset size variation by reducing the fine-tuning dataset to LibriSpeech-10h train set in first experiment and training on a small spontaneous English dataset, the Buckeye corpus [27] containing 11 hours of data, in a final one.

Figure 2 shows the performance obtained with the presented methods on the three datasets. While WSJ shows very similar patterns to the first set of experiments, we can see in the case of less fine-tuning data that the downsampling method performance drops significantly. Downsampling with factor 3, while suffering a relative WER augmentation of only 33.7% for Librispeech-100h and 39.1% for WSJ, witnesses a drop of 384.9% with Buckeye and 571.7% with LibriSpeech-10h compared to the full model performance. In contrast, the other methods seem more resilient to reduced data quantity. Despite this, downsampling the sequences by a factor 2 using a learned convolution remains a good option for highly reduced inference times without unacceptable performance drop.

## 4. CONCLUSION

In this work, we explored different methods to reduce speech recognition inference times using large self-supervised models through fine-tuning. The comparison of these methods indicates that sequence downsampling is the best performing option allowing substantial computation gain with low performance drops. Experiments led on other downstream datasets show that the size of the downstream dataset is critical to avoid high error rates.

## 5. ACKNOWLEDGEMENTS

This work has benefited from funding from l'Agence de l'Innovation de Défense, and was performed using HPC resources from GENCI-IDRIS (Grant 2021-AD011012801R1).

## 6. REFERENCES

- [1] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli, “wav2vec 2.0: A framework for self-supervised learning of speech representations,” 2020.
- [2] Mirco Ravanelli, Jianyuan Zhong, Santiago Pascual, Pawel Swietojanski, Joao Monteiro, Jan Trmal, and Yoshua Bengio, “Multi-task self-supervised learning for robust speech recognition,” 2020.
- [3] Salah Zaiem, Titouan Parcollet, Slim Essid, and Abdelwahab Heba, “Pretext tasks selection for multitask self-supervised audio representation learning,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 6, pp. 1439–1453, 2022.
- [4] Robin Algayres, Mohamed Salah Zaiem, Benoît Sagot, and Emmanuel Dupoux, “Evaluating the reliability of acoustic speech embeddings,” *arXiv preprint arXiv:2007.13542*, 2020.
- [5] Salah Zaiem, Titouan Parcollet, and Slim Essid, “Automatic data augmentation selection and parametrization in contrastive self-supervised speech representation learning,” *arXiv preprint arXiv:2204.04170*, 2022.
- [6] Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, Jian Wu, Long Zhou, Shuo Ren, Yanmin Qian, Yao Qian, Jian Wu, Michael Zeng, Xiangzhan Yu, and Furu Wei, “WavLM: Large-Scale Self-Supervised Pre-Training for Full Stack Speech Processing,” *IEEEJSTSP*, 2021.
- [7] Heng-Jui Chang, Shu-wen Yang, and Hung-yi Lee, “DistilHuBERT: Speech Representation Learning by Layer-wise Distillation of Hidden-unit BERT,” *ICASSP*, 2021.
- [8] Rui Wang, Qibing Bai, Junyi Ao, Long Zhou, Zhixiang Xiong, Zhihua Wei, Yu Zhang, Tom Ko, and Haizhou Li, “LightHuBERT: Lightweight and Configurable Speech Representation Learning with Once-for-All Hidden-Unit BERT,” in *INTER-SPEECH 2022*, Sep.2022.
- [9] Hsuan-Jui Chen, Yen Meng, and Hung-yi Lee, “Once-for-all sequence compression for self-supervised speech models,” 2022.
- [10] Yeonhyeon Lee, Kangwook Jang, Jahyun Goo, Youngmoon Jung, and Hoirin Kim, “Fithubert: Going thinner and deeper for knowledge distillation of speech self-supervised learning,” *arXiv preprint arXiv:2207.00555*, 2022.
- [11] Tzu-hsun Feng, Annie Dong, Ching-Feng Yeh, Shu-wen Yang, Tzu-Quan Lin, Jiatong Shi, Kai-Wei Chang, Zili Huang, Haibin Wu, Xuankai Chang, Shinji Watanabe, Abdelrahman Mohamed, Shang-Wen Li, and Hung-yi Lee, “SUPERB @ SLT 2022: Challenge on Generalization and Efficiency of Self-Supervised Speech Representation Learning,” oct 2022.
- [12] Ching-Feng Yeh, Wei-Ning Hsu, Paden Tomasello, Abdelrahman Mohamed, and Meta Ai, “Efficient Speech Representation Learning with Low-Bit Quantization ,” .
- [13] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever, “Robust speech recognition via large-scale weak supervision,” 2022.
- [14] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, Renato De Mori, and Yoshua Bengio, “Speech-brain: A general-purpose speech toolkit,” 2021.
- [15] Juan Zuluaga-Gomez, Amrutha Prasad, Iuliia Nigmatulina, Saeed Sarfjoo, Petr Motlicek, Matthias Kleinert, Hartmut Helmke, Oliver Ohneiser, and Qingran Zhan, “How does pre-trained wav2vec 2.0 perform on domain shifted asr? an extensive benchmark on air traffic control communications,” 2022.
- [16] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, “Librispeech: An asr corpus based on public domain audio books,” in *2015 (ICASSP)*, 2015, pp. 5206–5210.
- [17] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed, “Hubert: Self-supervised speech representation learning by masked prediction of hidden units,” *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 29, 2021.
- [18] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks,” 2006, ICML '06.
- [19] Kenneth Heafield, “KenLM: Faster and smaller language model queries,” in *Proceedings of the Sixth Workshop on Statistical Machine Translation*, Edinburgh, Scotland, July 2011.
- [20] Shubham Toshniwal, Anjali Kannan, Chung-Cheng Chiu, Yonghui Wu, Tara Sainath, and Karen Livescu, “A comparison of techniques for language model integration in encoder-decoder speech recognition,” 12 in *SLT 2018*, pp. 369–375.
- [21] Ankita Pasad, Ju-Chieh Chou, and Karen. Livescu, “Layer-wise analysis of a self-supervised speech representation model,” *ASRU 2021*.
- [22] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov, “On the effect of dropping layers of pre-trained transformer models,” *Computer Speech and Language*, vol. 77, pp. 101429, jan 2023.
- [23] Angela Fan, Edouard Grave, and Armand Joulin, “Reducing transformer depth on demand with structured dropout,” in *ICLR 2020*.
- [24] Dan Berrebbi, Brian Yan, and Shinji Watanabe, “Avoid overthinking in self-supervised models for speech recognition,” 2022.
- [25] Ji Won Yoon, Beom Jun Woo, and Nam Soo Kim, “HuBERT-EE: Early Exiting HuBERT for Efficient Speech Recognition,” .
- [26] Douglas B. Paul and Janet M. Baker, “The design for the Wall Street Journal-based CSR corpus,” in *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992, 1992*.
- [27] Mark Pitt, Keith Johnson, Elizabeth Hume, Scott Kiesling, and William Raymond, “The buckeye corpus of conversational speech: Labeling conventions and a test of transcriber reliability,” *Speech Communication*, vol. 45, pp. 89–95, 01 2005.