MLMI14 Practical 1: TIMIT/LibriSpeech Speech Recognition Using Foundation Models

Jan 27, 2023

Contents

1 Introduction **2**

1.1 Allocation of GPU hours 2

2 Fine-tuning Wav2Vec2 on TIMIT **2**

2.1 Brief Introduction to the Model 2

2.2 The Baseline (Default) Setting for TIMIT 2

2.3 Complete fine-tuning procedure 4

2.4 Comparison with MLMI2 5

3 Experiments on LibriSpeech **6**

3.1 Loading Data 6

3.2 Decoding and Default Model for LibriSpeech 7

3.3 Probing intermediate representations 8

3.4 Freezing part of the model 9

3.5 Combining representations from different layers (optional) 11

3.6 Fine-tuning with a frozen foundation model (optional) 13

3.7 Using other pre-trained foundation models (optional) 15

4 Summary **17**

4.1 Overall "best" system for TIMIT 17

4.2 Overall "best" system for LibriSpeech 17

4.3 Conclusion 18

References **19**

1 Introduction

The task of this practical builds on and extends the MLMI2 Speech Recognition exercise by utilising foundation models. We will first investigate using foundation models on TIMIT dataset for phone-level recognition task and compare the results with those in MLMI2 practical. Then, we will also investigate applying foundation model to a more challenging word-level automatic speech recognition (ASR) task on the LibriSpeech dataset. We will use the connectionist temporal classification (CTC) loss for all experiments in this practical. This practical is done in PyTorch.

1.1 Allocation of GPU hours

In this practical, There are more exercises associated with LibriSpeech experiments and hence more GPU hours will be allocated to LibriSpeech experiments. In all exercises of LibriSpeech, more GPU hours are allocated to exercises in section **3.4**, **3.6** and **3.7** where I spend more time investigating (these exercises allow for more explorations). The GPU hours spent on each sections of exercises are summarised below:

	Time (min)
section **2.2**	25 mins
section **2.3**	47 mins
section **3.2**	29 mins
section **3.3**	45 mins
section **3.4**	70 mins
section **3.5**	24 mins
section **3.6**	31 mins
section **3.7**	84 mins

Table 1: GPU time allocation

2 Fine-tuning Wav2Vec2 on TIMIT

2.1 Brief Introduction to the Model

The foundation model used in this part of the exercise is Wav2Vec2 [1] and more specifically, we will use the base version of Wav2Vec2 consisting of a series of Convolutional Feature Encoder layer followed by 12 Transformer layers with model input/output dimension of 768. The Wav2Vec2-base foundation model has been pre-trained on 960 hours of audio from the LibriSpeech corpus without the transcriptions. The entire Automatic Speech Recognition (ASR) system used in section 2 is constructed by simply stacking an extra feed forward layer on top of Wav2Vec2-base followed by a Softmax layer.

2.2 The Baseline (Default) Setting for TIMIT

The model architecture used for the default setting is exactly the same as described in section **2.1**. The optimiser used in the default model setting is Stochastic Gradient Descent (SGD)

with a learning rate of 0.001. The batch size used is 4 and the number of epochs is 20.

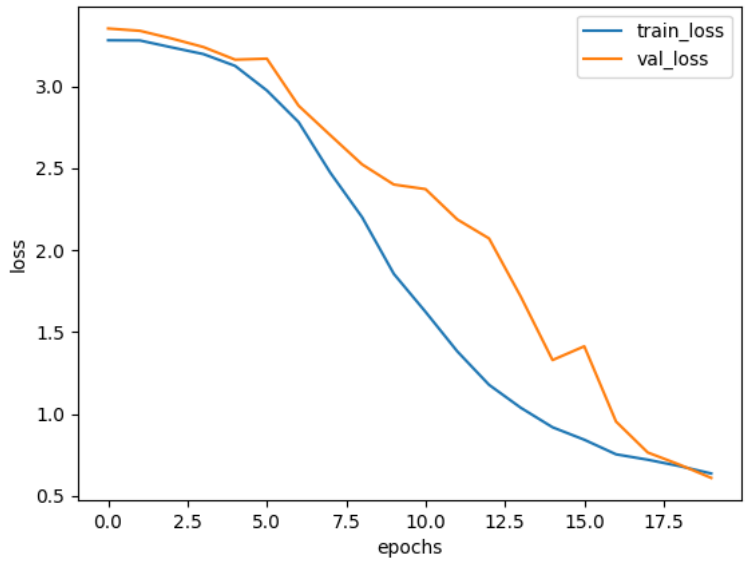


Figure 1: Training and Validation loss of default model (TIMIT)

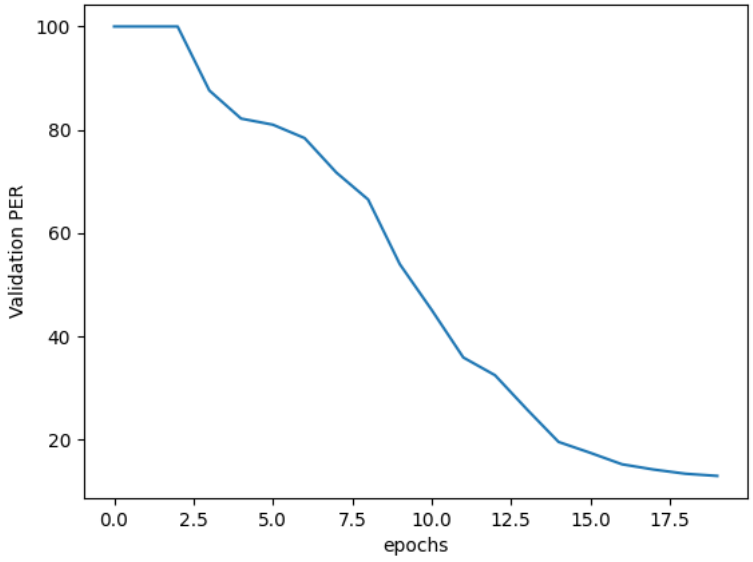


Figure 2: Validation PER of default model (TIMIT)

From Figure 1 and 2, the training and validation loss has dropped to 0.64 and 0.61 respectively. Also, the validation Phone Error Rate (PER) goes to around 13% (13.03%) at the 20th epoch. The training losses are generally lower than the validation losses except at the 20th epoch, which is as expect as we are minimising loss on training set. The training losses generally decreases faster than the validation loss before the 16th epoch, after that the two losses seems to decrease at a similar rate.

	SUB	DEL	INS	COR	PER
Default (TIMIT)	6.10%	3.99%	4.23%	89.91%	14.32%

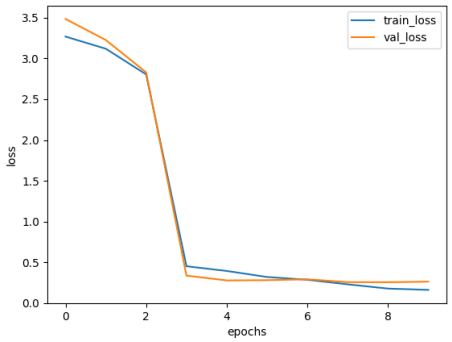
Table 2: Decoding results for default model (TIMIT)

The decoding results are in Table 2, where SUB stands for substitutions, INS stands for insertions, DEL stands for deletions, COR is the percentage of correctly predicted phones.

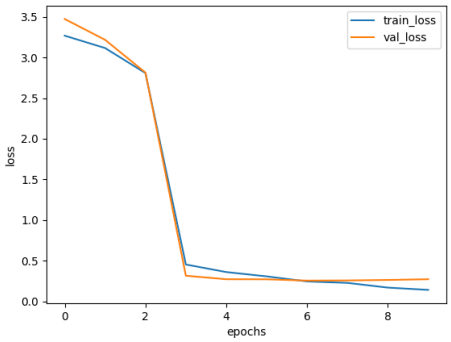
2.3 Complete fine-tuning procedure

In this section, we will be completing and exploring fine-tuning steps for the Wav2Vec2-base model on TIMIT. The optimiser used for all experiments in this section is Adam with a learning rate of 0.0001 and the number of epochs for all experiments starting from this section would be 10 (the batch size is still 4). The model architecture is exactly the same as in section **2.2**.

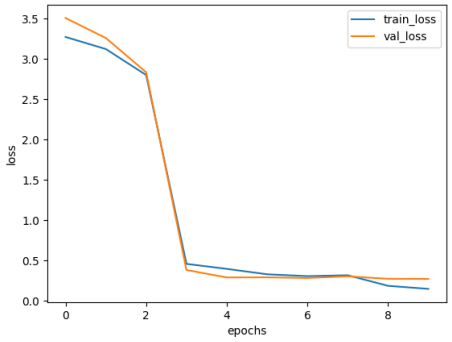
We start by freezing the entire Wav2Vec2-base model (except the output layer) for the first three epochs and unfreezing the whole model except the feature encoder during the next seven epochs, hold the learning rate constant for 50% of training steps (i.e. batches in this case) and then decay the learning rate linearly to 0 for the remainder of the training steps. Then, we also explore different setups for for the scheduler by start to decay the learning rate from the 4th epoch (i.e. constant learning rate for the first 30% steps and linearly-decayed learning rate for the rest 70% steps) and the 8th epoch (i.e. constant learning rate for the first 70% steps and linearly-decayed learning rate for the rest 30% steps) respectively (in all of these settings, we start unfreezing the Wav2Vec2-base model from the third epoch). Moreover, I also explore the effect of adding a warmup phase by linearly increasing the learning rate from 0 to 0.0001 for the first 10% training steps, maintaining a constant learning rate for the next 40% of training steps and linearly decaying the learning rate to 0 for the rest 50% of training steps.



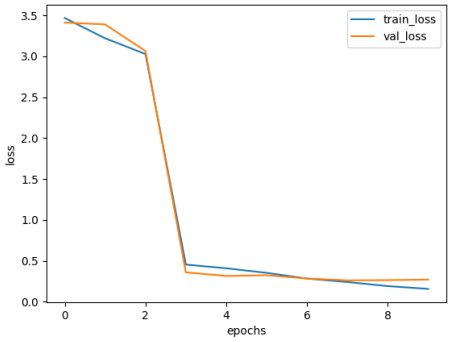
(a) Train and validation losses for 50/50 learning rate schedule



(b) Train and validation losses for 30/70 leaning rate schedule



(c) Train and validation losses for 70/30 leaning rate schedule



(d) Train and validation losses for 50/50 learning rate schedule with warmup

Figure 3: Train and validation losses for fine-tuning on TIMIT dataset with different settings

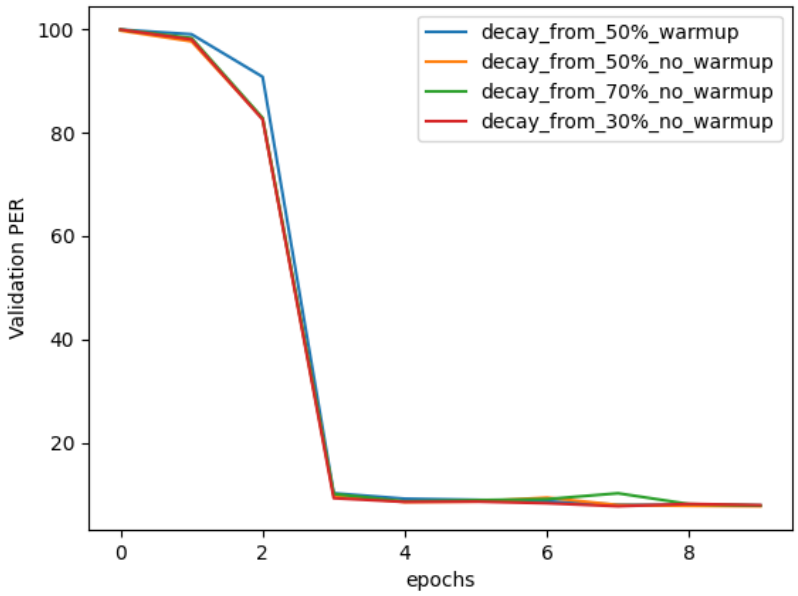


Figure 4: Validation PER of fine-tuning model on TIMIT with different settings

	SUB	DEL	INS	COR	PER
Default (TIMIT)	6.10%	3.99%	4.23%	89.91%	14.32%
50/50 scheduler, no warmup	5.32%	1.79%	2.11%	92.88%	9.23%
50/50 scheduler, with warmup	5.07%	2.09%	2.00%	92.84%	9.16%
30/70 scheduler, no warmup	4.85%	2.52%	1.88%	92.63%	9.25%
70/30 scheduler, no warmup	5.21%	1.95%	1.93%	92.84%	**9.09**%

Table 3: Decoding results for fine-tuning model on TIMIT with different settings

From Figure 3 and 4, in all of the four settings, the training and validation losses drops to about 0.15 and 0.30 in 10 epochs and the validation PER drops to around 8%. In Table 3, we can see that adding warmup and decaying the learning rate at different points do not really make a difference, all these four settings have achieved a test PER of around 9% at the end.

The reason for this learning rate scheduler is that usually we want to have larger learning rate at the first few epochs of the training procedure to let the optimiser get close to the (local or global) minimum faster and we want smaller learning rate in later epochs of the training procedure such that the optimiser would not oscillate around too much or even jump to another false minimum. The learning rate scheduler we implemented here exactly does this and achieved better validation PER in only 10 epochs compared to the default setting where we have a constant learning rate.

2.4 Comparison with MLMI2

Compared with the results in MLMI2 where the best model gives a test PER of around 20% in 20 epochs, the improvement by using foundation model is quite large where we achieved around 9% test PER within only 10 epochs. Moreover, note also that in MLMI2 practical, we need to first convert raw waveform into log Mel filterbanks (FBank) features before feeding them into the Long short-term memory (LSTM) layer. Here, the input of the foundation model is simply the raw waveform, which does not require any feature engineering.

3 Experiments on LibriSpeech

In this part of the practical, we will do word-level speech recognition task using the 10h subset of the LibriLight data as the training set. Now there are two test datasets, `test_clean` and `test_other`. In each experiment we do, we will report Word Error Rate (WER) on both `test_clean` and `test_other`. We will use Wav2Vec2-base in all experiments we do in this part except section **3.7** (where we will use WavLM-base as the foundation model instead).

3.1 Loading Data

The training procedure is almost the same as the TIMIT experiments. In the LibriSpeech experiments, we will be working with characters as output. Thus, the first thing to do here is to create a `vocab.txt` for LibriSpeech by collecting all characters that appeared in the training set `train_10h.json`. Also, our `vocab.txt` contains an inter-word space character `<space>`, a special symbol `<unk>` for unseen characters and a special symbol `<blank>` for blank which has index 0 in `vocab.txt`. Our `vocab.txt` file contains 26 English characters, the apostrophe character ' and the three special symbols above.

The next step is to create the dataloader for LibriSpeech. The dataloader reads the word transcripts, converts them into character sequences, and converts each character into its index as the target based on `vocab.txt`. The code for dataloader is as follows:

```
# self.vocab is defined in class TDataset and is a dict mapping character to index.
class TDatasetwav(TDataset):
    def __getitem__(self, index):
        data = self.data[self.data_idx[index]]
        data_path = data["wav"]
        wav, _ = torchaudio.load(data_path)
        wav = wav[0]
        wav_mean = torch.mean(wav, dim=0, keepdims=True)
        wav_std = torch.std(wav, dim=0, keepdims=True)
        wav = (wav - wav_mean) / wav_std
        word = data["word"]
        tgt_list = []
        for i in word:
            if i in self.vocab.keys():
                tgt_list.append(self.vocab[i])
            elif i == " ":
                tgt_list.append(self.vocab["<space>"])
            else:
                tgt_list.append(self.vocab["<unk>"])

        tgt = torch.tensor(tgt_list)
        duration = data["duration"]
        return wav, tgt, duration
```

3.2 Decoding and Default Model for LibriSpeech

The next step is to modify the `decode()` function in `decoder.py` to calculate WER. I simply added the following codes in the original `decode()` for retrieving predicted sentences and ground truth transcripts:

```
outputs = [[idx2grapheme[i] for i in j] for j in outputs.tolist()]
outputs = [[v for i, v in enumerate(j) if i == 0 or v != j[i - 1]] for j in outputs]
outputs = [list(filter(lambda elem: elem != "<blank>", i)) for i in outputs]
outputs = ["".join(i) for i in outputs]
outputs = [i.replace("<space>", " ") for i in outputs]

trans = [[idx2grapheme[i.item()] for i in j] for j in trans]
trans = ["".join(i) for i in trans]
trans = [i.replace("<space>", " ") for i in trans]
```

This new `decode()` would be called twice after training for both `test_clean` and `test_other`.

After creating the new `decode()`, we then start training a default model for LibriSpeech experiments. The model architecture used here is exactly the same as in section **2** (Wav2Vec2-base + one feed forward layer followed by Softmax). We still use Adam with a learning rate of 0.0001, the number of epochs is still 10 and the batch size is still 4. The fine-tuning procedure is: freeze the entire Wav2Vec2-base model (except the output layer) for the first three epochs and unfreeze the whole model except the feature encoder during the next seven epochs, hold the learning rate constant for 50% of training steps and then decay the learning rate linearly to 0 for the remainder of the training steps. We do not use warmup here as it does not really improve the results significantly, as shown in section **2.3**.

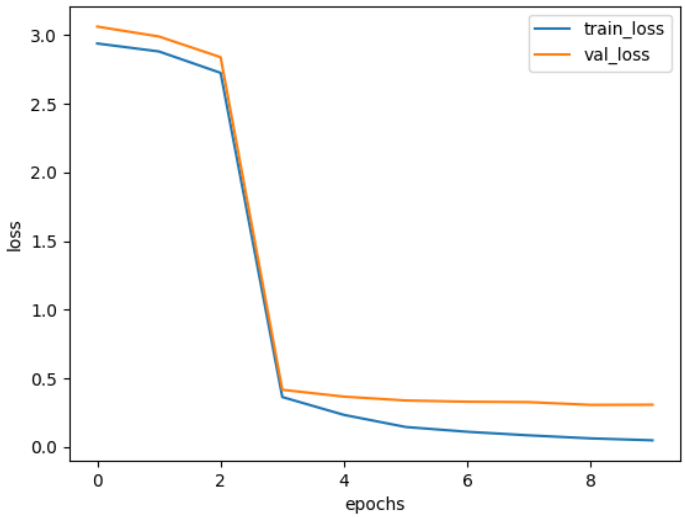


Figure 5: Training and Validation loss of default model on LibriSpeech

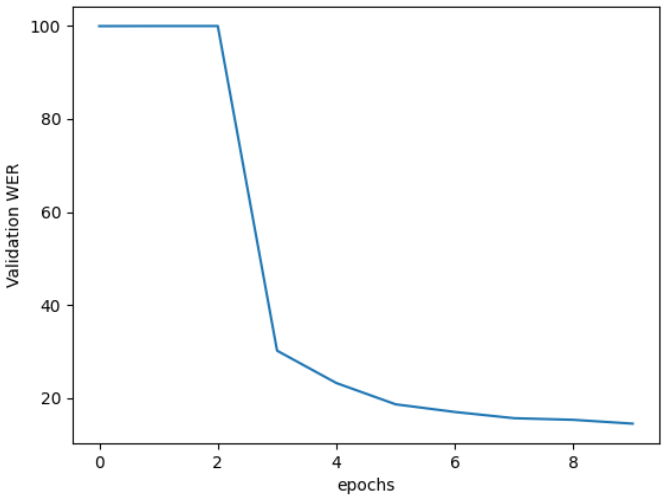


Figure 6: Validation WER of default model on LibriSpeech

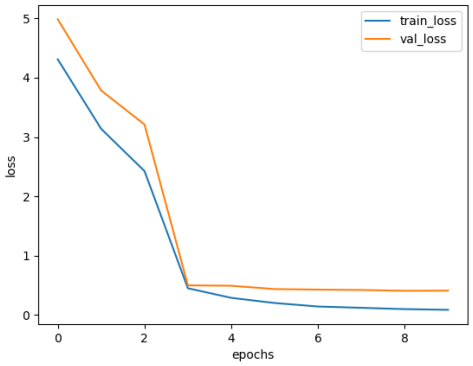
	Test Clean WER	Test Other WER
Default (LibriSpeech)	11.78%	21.25%

Table 4: Decoding results for default model on LibriSpeech

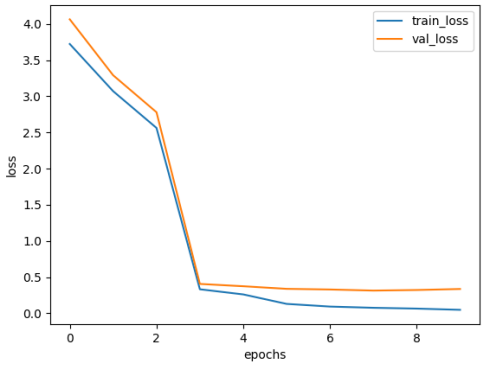
From Figure 5 and 6, training loss and validation loss goes down to around 0.05 and 0.3 after 10 epochs while validation WER goes down to around 14.5% after 10 epochs. In Table 4, WER for `test_other` is about twice the WER for `test_clean`.

3.3 Probing intermediate representations

All previous results have used the 12th Transformer block outputs. In this part, we will investigate using the hidden representations after (i) the 8th Transformer block and (ii) the 10th Transformer block (the representations after these intermediate Transformer blocks would be fed directly into the output layer) and see whether the performance would change much (all the other settings, such as fine-tuning procedure, are the same as the default model on LibriSpeech).



(a) Train and validation losses for probing the 8th Transformer block on LibriSpeech



(b) Train and validation losses for probing the 10th Transformer block on LibriSpeech

Figure 7: Train and validation losses for probing the intermediate Transformer blocks on LibriSpeech

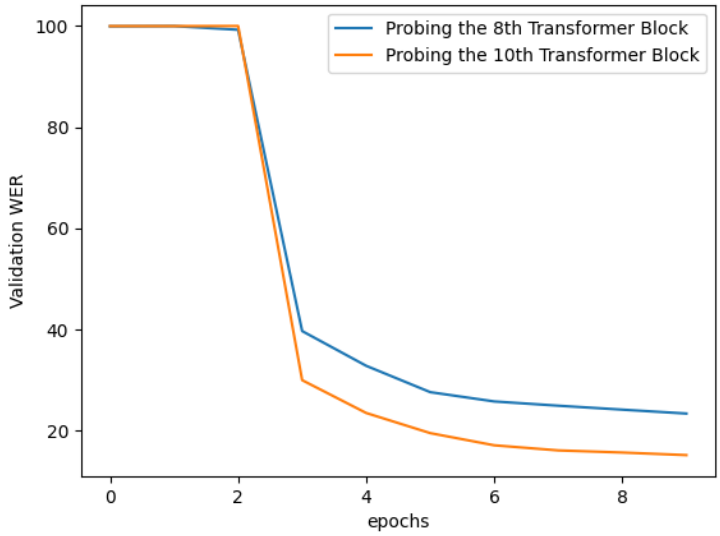


Figure 8: Validation WER for probing the intermediate Transformer blocks on LibriSpeech

	Test Clean WER	Test Other WER
Default (LibriSpeech)	**11.78**%	**21.25**%
8th Transformer block	18.18%	30.75%
10th Transformer block	12.75%	22.45%

Table 5: Decoding results for probing the intermediate Transformer blocks on LibriSpeech

From Figure 7 and 8, representations from the 8th Transformer block result to a higher training loss, validation loss and validation WER (around 0.09, 0.4 and 24% respectively) at the 10th epoch compared to the default model while representation from the 10th Transformer block gives similar training loss, validation loss and validation WER at the 10th epoch compared to the default model.

Table 5 shows similar results, the WERs on `test_clean` and `test_other` for the 8th Transformer block representations are significantly worse than those for the default model while the WERs on `test_clean` and `test_other` for the 10th Transformer block representations are only slightly worse than those for the default model.

The reason for this difference in performance is due to the fact that later Transformer blocks of Wav2Vec2-base capture fine-grained phonetic-level information while earlier Transformer blocks, such as the 8th block, capture global utterance-level information. Therefore, for phonetic-level downstream tasks such as speech recognition, using representations from later Transformer blocks would be better, representations from the 8th block might be more suitable for tasks like speaker diarisation.

3.4 Freezing part of the model

As shown in previous experiments, the validation loss in later epochs seems to get stuck between 0.3 and 0.33, which possibly indicates some overfitting. Therefore, in this part, we will investigate the effect of freezing (i) the first 6 Transformer blocks and (ii) the first 9 Transformer blocks (iii) the first 10 Transformer blocks during all 10 training epochs (all other training settings are exactly the same as the default model, which means the feature encoder would still

be frozen throughout the 10 epochs).

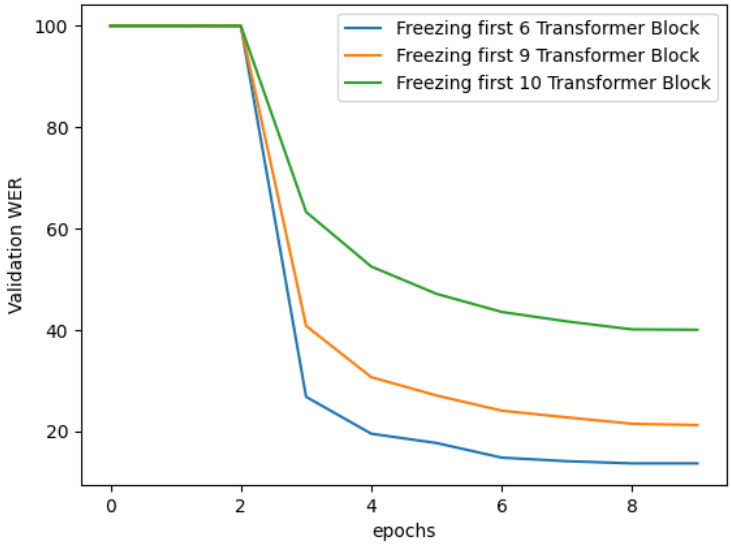
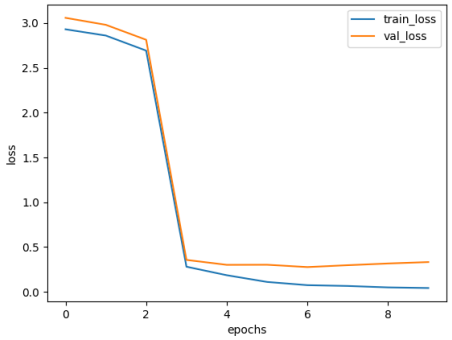
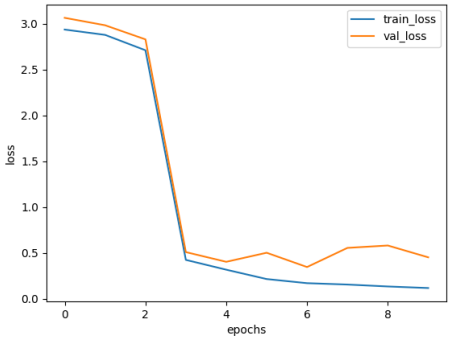


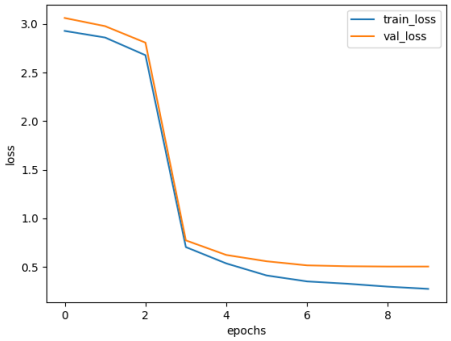
Figure 9: Validation WER for freezing different numbers of Transformer blocks on LibriSpeech



(a) Train and validation losses for freezing the first 6 Transformer blocks on LibriSpeech



(b) Train and validation losses for freezing the first 9 Transformer blocks on LibriSpeech



(c) Train and validation losses for freezing the first 10 Transformer blocks on LibriSpeech

Figure 10: Train and validation losses for freezing different numbers of Transformer blocks on LibriSpeech

	Test Clean WER	Test Other WER
Default (LibriSpeech)	**11.78**%	21.25%
Freezing first 6 Transformer Blocks	12.95%	**19.99**%
Freezing first 9 Transformer Blocks	22.15%	28.57%
Freezing first 10 Transformer Blocks	36.46%	44.76%

Table 6: Decoding results for freezing different numbers of Transformer blocks on LibriSpeech

In Figure 9 and 10, we can see that freezing the first 6 Transformer blocks has very minor effect on the training loss, validation loss and validation WER compared to the default setting. Freezing the first 9 Transformer blocks makes the training loss, validation loss and validation WER drop to around 0.12, 0.45 and 21.3% respectively after 10 epochs, which is worse than the default model. Freezing the first 10 Transformer blocks gives significantly worse training loss, validation loss and validation WER after 10 epochs (around 0.27, 0.5 and 40% respectively) compared to the default model. It seems that freezing Transformer blocks has not really alleviate overfitting at all.

In Table 6, we have observed a similar pattern. Freezing the first 6 Transformer blocks results to slightly worse WER on `test_clean` and surprisingly better WER on `test_other` compared to the default setting. Freezing the first 9 Transformer blocks gives about 9% worse WER on `test_clean` and `test_other` compared to freezing the first 6 transformer blocks while freezing the first 10 Transformer blocks results to significantly worse performance on `test_clean` and `test_other` compared to freezing the first 9 Transformer blocks.

The reason for such behaviour is similar to section **3.3**. Later Transformer blocks (from the 10th onwards) capture fine-grained phonetic-level information while earlier Transformer blocks, such as the 8th Transformer block, capture global utterance-level information (for speech recognition, phonetic-level information is more useful), which is why freezing the first 10 Transformer blocks gives significant worse performance compared to freezing the first 9 transformer blocks and why freezing the first 6 transformer blocks gives almost no difference in performance compared to the default setting. It seems that the 10th Transformer block is very significant for the speech recognition task here.

3.5 Combining representations from different layers (optional)

As manually finding the best Transformer block would be time-consuming, in this part, we apply a series of learnable weights on the representations from each of the 12 Transformer blocks of the foundation model to obtain a weighted combined representation from these Transformer blocks and this combined representation would be fed into the output layer (all other settings are the same as the default model for LibriSpeech). Note also that the weights here are unnormalised logits instead of normalised probabilities.

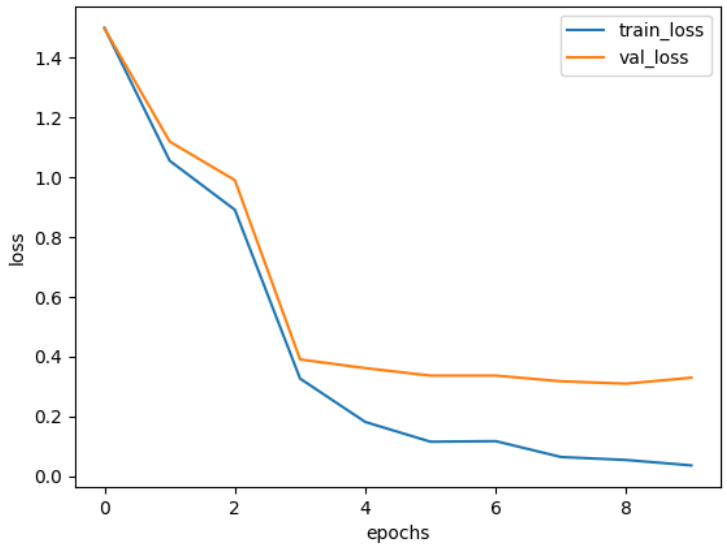


Figure 11: Train and Validation loss for combining representations from 12 Transformer blocks on LibriSpeech

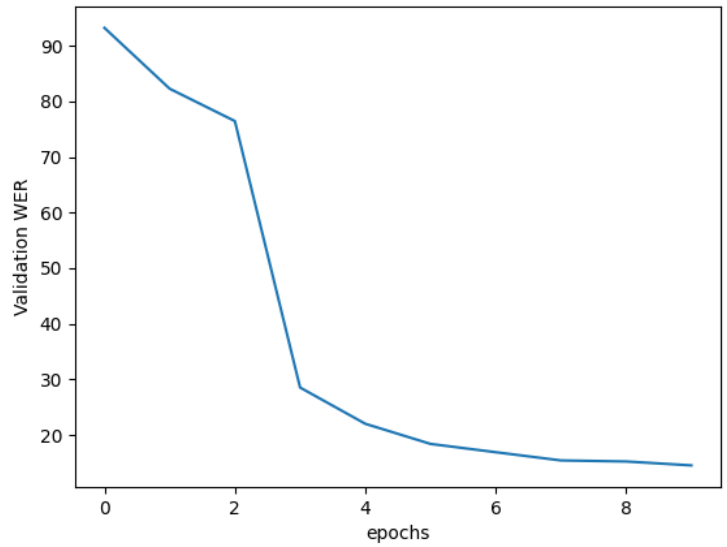


Figure 12: Validation WER for combining representations from 12 Transformer blocks on LibriSpeech

	Test Clean WER	Test Other WER
Default (LibriSpeech)	11.78%	21.25%
Combined Representation	**11.70**%	**20.37**%

Table 7: Decoding results for combining representations from 12 Transformer blocks on LibriSpeech

The performance for combining representations from 12 Transformer block is a bit better than the default setting with similar WER on `test_clean` and smaller WER on `test_other` in Table 7. The training loss, validation loss and validation WER for combining representations from 12 Transformer block are similar to those of the default model.

	Learned Weight
1st Transformer block	0.8405
2nd Transformer block	0.8507
3rd Transformer block	0.9013
4th Transformer block	0.9912
5th Transformer block	1.0702
6th Transformer block	1.1137
7th Transformer block	1.1449
8th Transformer block	1.1724
9th Transformer block	**1.2101**
10th Transformer block	**1.2146**
11th Transformer block	1.1252
12th Transformer block	1.0942

Table 8: Learned weights for representations from 12 Transformer blocks

Table 8 gives the distribution of the trained weights for the 12 Transformer blocks. We can see that higher weights are given to Transformer blocks 6-10 and block 9 and 10 are give the top two highest weights, which agrees with our findings in section **3.3** and **3.4** that later Transformer blocks are more important and the 10th Transformer block is very significant for the speech recognition task here.

3.6 Fine-tuning with a frozen foundation model (optional)

Sometimes people only train extra layers on top of the foundation models instead of fine-tuning the whole model. The reason for this might be that:

- The labelled training set is very small, fine-tuning all parameters of the foundation model might leads to overfitting.
- The labelled training data is too large such that training with all parameters of the foundation model would be computationally costly.
- Also, when the data used for pre-training the foundation model is highly similar to the data used for fine-tuning, the features learnt by the foundation model from the pre-trained data might also be very relevant to the data for fine-tuning and thus fine-tuning all parameters of the foundation model would not be necessary and we may only train the output layer on top.

In this part of the exercise, we will freeze the Wav2Vec2-base foundation model during the entire fine-tuning procedure and we will investigate whether changing the architecture of the output layer to more complex structures would be of any help (all other fine-tuning settings would be the same as the default setting). Other than one feed forward neural network on top, we try adding 2 feed forward layers (connected by ReLU, the first feed forward layer has dimension 384 and the second feed forward layer has dimension 30) on top of the foundation model and also try adding a 128-dimensional bidirectional Long short-term memory (biLSTM) followed by a feed forward layer with dimension 30 on top of the foundation model (all of these layers would be followed by Softmax for prediction).

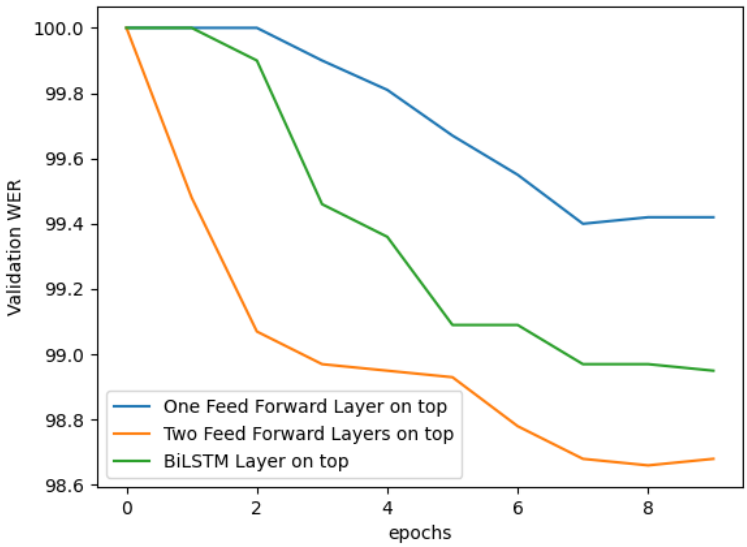
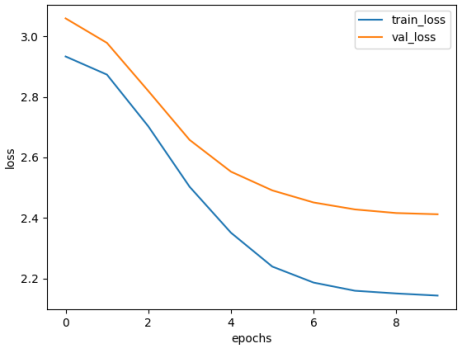
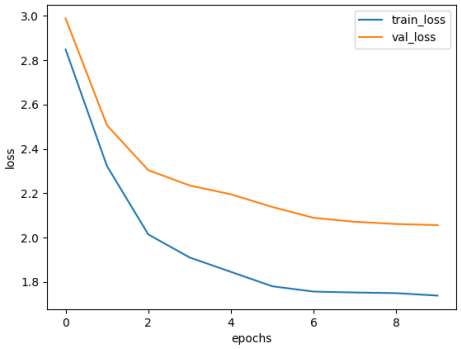


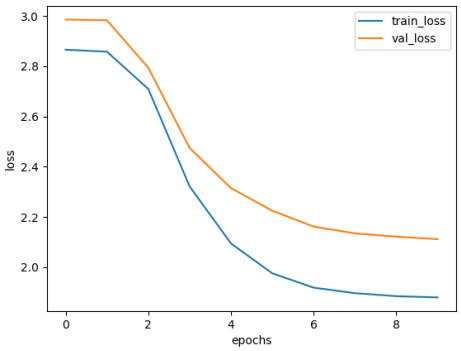
Figure 13: Validation WER for for freezing the foundation model for all 10 epochs with different layers on top (LibriSpeech)



(a) Train and validation losses for training with one feed forward on top (LibriSpeech)



(b) Train and validation losses for training with 2 feed forward layers on top (LibriSpeech)



(c) Train and validation losses for training with one biLSTM layer on top (LibriSpeech)

Figure 14: Train and validation losses for freezing the foundation model for all 10 epochs with different layers on top (LibriSpeech)

	WER (Clean)	WER (Other)	DEL (Clean)	DEL (Other)	SUB (Clean)	SUB (Other)
Default (LibriSpeech)	**11.78**%	**21.25**%	**0.66**%	**1.98**%	**10.48**%	**18.12**%
One Feed Forward layer	99.64%	99.54%	65.68%	63.34%	33.96%	36.19%
Two Feed Forward layers	98.49%	98.82%	49.16%	50.38%	49.32%	48.41%
One biLSTM layer	98.84%	99.02%	47.07%	45.73%	51.72%	53.20%

Table 9: Decoding results for freezing the foundation model for all 10 epochs with different layers on top (LibriSpeech)

From Figure 13, Figure 14 and Table 9, we can see that adding one extra feed forward neural network is definitely not enough when freezing the whole foundation model, resulting to large training, validation loss and almost 100% WERs on `test_clean` and `test_other`. Surprisingly, using more complex structure such as 2 feed forward layers and even biLSTM does not really make the performance better when the whole foundation model is frozen, the WERs on `test_clean` and `test_other` are still nearly 100%. Also, we can see that when freezing the whole foundation model, we get extremely high DEL (especially for the one feed forward layer case), which indicates that the model is outputting `<blank>` much too often and this is a major source of error (the SUB is also very high, which is another major error source).

This kind of bad performance indicates that the representations from the pre-trained foundation model itself without being fine-tuned does not really provide useful information for the downstream speech recognition task on the 10h LibriLight data and hence fine tuning the parameters of the foundation model is still necessary for achieving reasonable results.

3.7 Using other pre-trained foundation models (optional)

All the above experiments are done with Wav2Vec2, but Wav2Vec2 is not the only foundation model available for our speech recognition task. In this part, we will use [2] as our foundation model, we will use the base version of the WavLM model, WavLM-base, such that the results are comparable with Wav2Vec2-base. The WavLM-base model also has 12 Transformer blocks with about 94.70M parameters which is very similar to Wav2Vec2-base. What's different for WavLM is that WavLM pre-training masks both speech prediction and speech denoising which helps to improve the potential on both ASR and non-ASR tasks and also WavLM added gated relative position bias [3] to Transformer block which improves its ASR performance. The output layer on top of WavLM-base would still be one feed forward layer (followed by Softmax) and we will still run 10 epochs. We will follow the fine-tuning procedure in the original paper [2]:

- The whole foundation model for 4 epochs and then unfreeze all parameters except the feature encoder.
- We will use a learning rate scheduler that contains a warm-up phase for the first 10% of total training update steps, followed by a constant learning rate for the next 40% of the total steps, and then decay the learning rate linearly for the last 50% of steps (this scheduler is used for all WavLM experiments in Table 10) .

The learning rate (referred to as `lr` in Table 10) of Adam used in [2] is 2e-5, but in practice we find that this learning rate is too small here for our task and 1e-4 is more appropriate. Also, we have investigated freezing the whole WavLM-base model for 3 epochs instead of 4 before fine-tuning the WavLM-base model parameters (the number of epochs we freeze the foundation model before fine-tuning parameters of the foundation model is referred to as `num_epoch_freeze` in Table 10). Finally, the pre-trained weights are loaded through `'microsoft/wavlm-base'`.

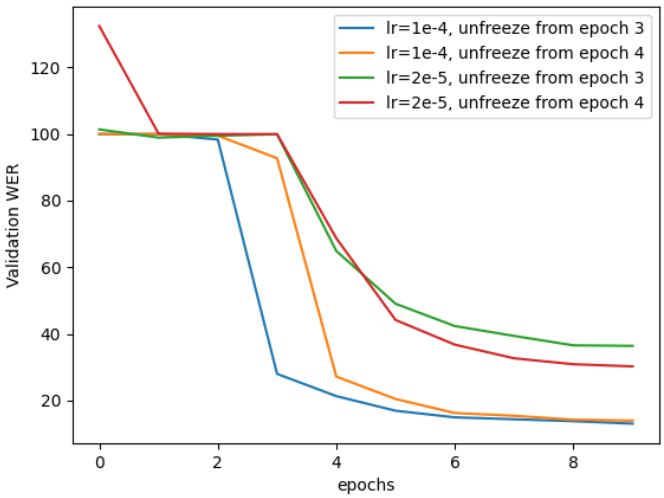
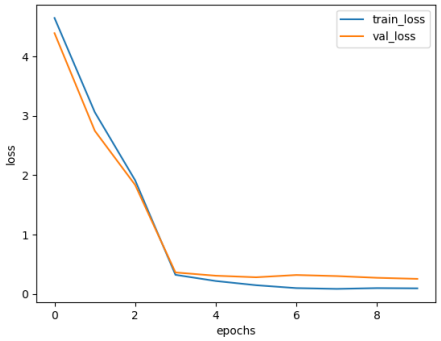
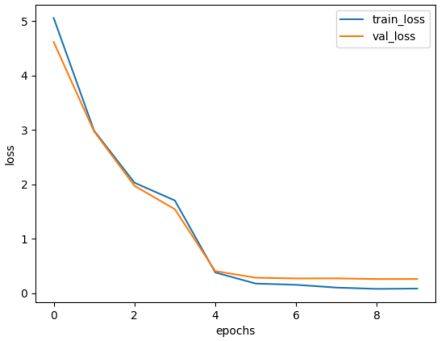


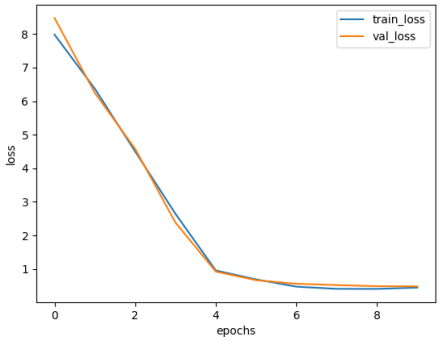
Figure 15: Validation WER for different fine-tuning settings of WavLM-base based model on LibriSpeech



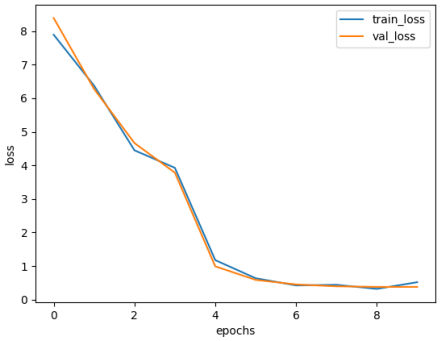
(a) Train and validation losses for WavLM-base based model with lr=1e-4 and unfreezing after 3 epochs on LibriSpeech



(b) Train and validation losses for WavLM-base based model with lr=1e-4 and unfreezing after 4 epochs on LibriSpeech



(c) Train and validation losses for WavLM-base based model with lr=2e-5 and unfreezing after 3 epochs on LibriSpeech



(d) Train and validation losses for WavLM-base based model with lr=2e-5 and unfreezing after 4 epochs on LibriSpeech

Figure 16: Train and validation losses for different fine-tuning settings of WavLM-base based model on LibriSpeech

	Test Clean WER	Test Other WER
Default (LibriSpeech)	11.78%	21.25%
Combined Representation (section **3.5**)	11.70%	20.37%
WavLM-base + `lr`=1e-4 + `num_epoch_freeze`=3	**10.40**%	**18.22**%
WavLM-base + `lr`=1e-4 + `num_epoch_freeze`=4	11.53%	19.07%
WavLM-base + `lr`=2e-5 + `num_epoch_freeze`=3	31.66%	41.51%
WavLM-base + `lr`=2e-5 + `num_epoch_freeze`=4	26.40%	33.36%

Table 10: Decoding results for different fine-tuning settings of WavLM-base based model on LibriSpeech

From the above, we can see that a learning rate of 2e-5 is indeed not appropriate for our ASR task here where the training loss, validation loss, validation WER and test WER are all worse than the default setting. When we use 1e-4 as the learning rate, the performance of WavLM-base based model has indeed outperformed the best-performed Wav2Vec2-base based model (the one in section **3.5** constructed by combined representations from all 12 Transformer blocks of Wav2Vec2-base) for both `num_epoch_freeze`=3 and `num_epoch_freeze`=4 on both `test_clean` and `test_other`, from Table 10.

4 Summary

4.1 Overall "best" system for TIMIT

In the TIMIT experiments, the overall "best" ASR system, in terms of test WER, is the Wav2Vec2-base based model with an extra feed forward layer (followed by Softmax) on top trained with Adam with a learning rate of 1e-4 for 10 epochs. The fine-tuning procedure for the best ASR system for TIMIT is that the learning rate is kept constant for 70% of the training steps and decayed linearly to 0 for the rest 30% steps and the whole foundation model is frozen for the three epochs before the entire model (except the feature encoder) start to train.

However, from Table 3, we can see that all the four different fine-tuning settings have achieved pretty much similar performance on test set, the "best" setting is only slightly better than the other three settings.

4.2 Overall "best" system for LibriSpeech

The overall "best" system for LibriSpeech, instead, is the WavLM-base based model with an extra feed forward layer on top (followed by Softmax) trained with Adam with a learning rate of 1e-4. The fine-tuning procedure for the best ASR system for LibriSpeech is that we use a learning rate scheduler that contains a warm-up phase for the first 10% of total training update steps, followed by a constant learning rate for the next 40% of the total steps, and then decay the learning rate linearly for the last 50% of steps and the whole foundation model is frozen for the three epochs before the entire model (except the feature encoder) start to train.

From Table 10, we can see that this WavLM-base based system has indeed outperformed the best Wav2Vec2-base based system on LibriSpeech on both `test_clean` and `test_other`, which demonstrates the power of gated relative position bias and masked speech denoising on ASR.

4.3 Conclusion

In this practical, we have explored the use of foundation models Wav2Vec2 and WavLM for ASR tasks. We first completed and investigated fine-tuning procedures of Wav2Vec2 on TIMIT and has achieved much better results compared to MLMI2 coursework. Then, we further applied Wav2Vec2 to more challenging word-level ASR tasks on LibriSpeech and investigated the Wav2Vec2 model architecture in detail. Finally, we adapted WavLM for the LibriSpeech ASR and achieved even better results, which is not far from the state-of-the-art in this field (fusing Language Models during decoding might lead to even better performance on ASR tasks).

References

[1] Baevski, A., Y. Zhou, A. Mohamed, and M. Auli (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems 33*, 12449–12460.

[2] Chen, S., C. Wang, Z. Chen, Y. Wu, S. Liu, Z. Chen, J. Li, N. Kanda, T. Yoshioka, X. Xiao, J. Wu, L. Zhou, S. Ren, Y. Qian, Y. Qian, M. Zeng, and F. Wei (2021). Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing 16*, 1505–1518.

[3] Chi, Z., S. Huang, L. Dong, S. Ma, B. Zheng, S. Singhal, P. Bajaj, X. Song, X.-L. Mao, H. Huang, et al. (2021). Xlm-e: Cross-lingual language model pre-training via electra. *arXiv preprint arXiv:2106.16138*.