Splitformer: An improved early-exit architecture for automatic speech recognition on edge devices

1st Maxence Lasbordes
Université Paris-Dauphine, Université PSL,
Télécom SudParis, Institut Polytechnique de Paris
Paris, France
maxence.lasbordes@dauphine.eu

2nd Daniele Falavigna
Center for Augmented Intelligence
Fondazione Bruno Kessler
Trento, Italy
falavi@fbk.eu

3rd Alessio Brutti
Center for Augmented Intelligence
Fondazione Bruno Kessler
Trento, Italy
brutti@fbk.eu

***Abstract*—The ability to dynamically adjust the computational load of neural models during inference in a resource aware manner is crucial for on-device processing scenarios, characterised by limited and time-varying computational resources. Early-exit architectures represent an elegant and effective solution, since they can process the input with a subset of their layers, exiting at intermediate branches (the upmost layers are hence removed from the model).**

From a different perspective, for automatic speech recognition applications there are memory-efficient neural architectures that apply variable frame rate analysis, through downsampling/upsampling operations in the middle layers, reducing the overall number of operations and improving significantly the performance on well established benchmarks. One example is the Zipformer. However, these architectures lack the modularity necessary to inject early-exit branches.

With the aim of improving the performance in early-exit models, we propose introducing parallel layers in the architecture that process downsampled versions of their inputs. We show that in this way the speech recognition performance on standard benchmarks significantly improve, at the cost of a small increase in the overall number of model parameters but without affecting the inference time.

***Index Terms*—conformer, zipformer, early-exit, dynamic models, ASR**

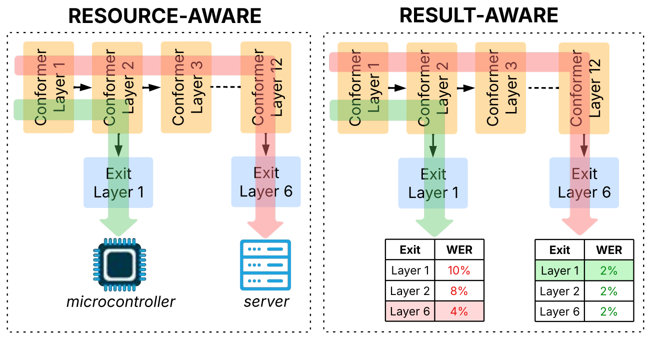
I. Introduction

Although the use of large speech foundation models (SFMs) is very widespread nowadays for automatic speech recognition (ASR) applications, their utilization on edge devices, where memory and computation resources are very limited, is still prevented. Resource-constrained applications require to use models that have much less parameters than those of SFMs and that can dynamically change their trade-off between computation and performance. To this end, we investigated in the past the use of early-exit architectures applied to large-vocabulary ASR [1].

"Early-exit" (EE) architectures introduce intermediate exit branches [2], [3] that allow to process the input by only a subset of layers exiting before reaching the top layer, thus saving computations. Early-exit architectures leverage the observation that, for simpler inputs, the lowest layers of the model may have already learned sufficient parameters for accurate predictions and have been succesfully applied to ASR [4], [5].

Figure 1 shows an example of an early-exiting network, where specific decoders process exits of intermediate layers. Note that this architecture is suited to be implemented either over a distributed environment, consisting of models with a device specific number of layers (”resource aware”), or using a single model that selects the best exit according to a given metric (”result aware”). In this work, we address only the resource-aware case, referring the reader to our previous work for details on the usage of early-exit metrics [1].

This work was partially funded by the PNRR project CN - HPC (Spoke 9) under the NRRP MUR program funded by the NextGenerationEU.



Exit	WER
Layer 1	10%
Layer 2	8%
Layer 6	4%

Exit	WER
Layer 1	2%
Layer 2	2%
Layer 6	2%

Fig. 1. *Resource-aware* and *result-aware* uses of early-exits. Left: the device is allowed to dynamically process only two layers, whereas the server can handle the whole model. Right: the first input needs to be processed by by all of the encoder layers; in the second case, the best transcription is produced after only two encoder layers.

Recently the *Zipformer*, a faster and more memory-efficient encoder architecture, has been proposed for ASR [6]. This architecture overcomes all previous confomer based models [7]–[9] both in terms of performance and computational load (meant as both memory occupancy and inference time). The zipformer implements, in addition to various innovative features, a stack of layers that downsample the input sequence with different lower frame rates. The intuition behind this approach is that processing with a few additional layers, applying different sampling rates, allows for broader acoustic contexts to be considered. Unfortunately, these architectures, introducing different frame rates through the backbone, do not marry well with early exit branches.

Inspired by the Zipformer, we have modified our previous early-exit architecture [1] by exploiting both input layer downsampling and parallel connections between layers. Experimenting with different variants of this approach, we observed that the best model includes two parallel downsampling layers, in both the first and last encoder outputs.

We show that in this way the word error rate (WER) on standard ASR benchmarks can be significantly reduced, at the cost of a small increase in the overall number of model parameters. We have also measured small gains in the computation times in the lowest exits of the proposed architecture. In summary, the contribution of this work is as follows: *i)* the development of a new EE architecture inspired by zipformer; *ii)* an experimental analysis on well established ASR benchmarks showing the effectiveness of the proposed architecture; *iii)* an analysis of performance vs. model complexity, expressed both in terms of computation times and number of floating point operations (FLOPS), depending on each exit.

II. Related Work

Early-exiting methods were first introduced for computer vision in BranchyNet [2] by adding two branches to AlexNet [10]. The authors optimised the joint loss of the exits and defined a confidence measure, based on the entropy of the output class distribution, to decide the exit level. To accelerate inference time in ASR, [11] proposes to use confidence measures, given by CTC decoders, or entropies computed from the logits of early-exits of a pre-trained audio encoder (i.e. HUBERT). A deep analysis of "overthinking" of ASR encoders has been carried out in [12], where theoretical lower limits of inference speed vs. ASR performance have been derived for different early exit strategies. Similar investigations have been explored in [5] for streaming using recurrent neural networks. [4] have investigated early-exit fine-tuning strategies in the context of a large pre-trained WavLM [13] model, comparing them with approaches based on layer removal and input down-sampling.

In our previous work [1] we investigated the training dynamics of early-exit models, showing that training the model from scratch, jointly optimizing all exited layers, provides significant performance improvements over both conventional single-exit models and fine-tuned pre-trained models (particularly at the lowest exits).

From a different perspective, two previous works have optimized the conformer architecture, for ASR tasks, introducing variants both in several points of the basic conformer module and in the pipeline. The *Squeezeformer* architecture described in [9] uses: *a)* the U-net [14] temporal structure to reduce the computations required by the multi-head attention modules in the conformer pipeline and *b)* a simplified basic block, as an alternative to the Macaron structure of the conformer [7], similar to a standard transformer block. This work shows that the U-net structure, applying downsampling/upsampling operations to the input/output layers of the architecture, allows to reduce the number of operations in the whole architecture maintaining, at the same time, the resolution of the original input features. Similarly, the *Zipformer* architecture [6] operates in the middle layers at lower frame rates. It also applies a set of changes to the basic conformer architecture that improves both velocity and memory efficiency. Either architectures have been successfully applied to standard ASR benchmarks showing significant performance gain.

In this work, we inherit the idea of U-net and apply it to an early-exit architecture. To make this feasible we introduce parallel layers in the baseline model, that perform downsampling/upsampling of the input embeddings. This approach significantly improves the overall performance and also allows to slightly reduce the decoding times in the lowest exits.

III. Early-Exit Loss

Given an input sequence of acoustic observations $\mathbf{x}$ and a neural model Θ, an ASR system estimates the output sequence $\hat{\mathbf{y}} = \hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_L$ as:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} P[\mathbf{y}|\mathbf{x}, \Theta], \tag{1}$$

where $\mathbf{y} \in \mathcal{Y}^*$, for some vocabulary $\mathcal{Y}$, such as graphemes, phonemes, or word-pieces. Usually Θ is factored into an encoder and a decoder. The encoder extracts an high-dimensional representation $\mathbf{h}_1^T$ of the input $\mathbf{x}$ and the decoder maps this representation into the output sequence $\hat{\mathbf{y}}$. Since $L \ll T$ in general, ASR decoders either use; *a)* an *alignment function* ($\mathcal{B} : \mathbf{a}_1^T \to \mathbf{r}$) for sequence training ($\mathbf{r}$ is the reference labels string), or *b)* a *cross attention mechanism* with label-based cross-entropy optimization [15]. Our goal is to apply early-exiting to ASR by adding decoders at some intermediate layers of the encoder (see Fig. 1). Assuming to use M intermediate decoders, giving outputs $\hat{\mathbf{y}}(1), \ldots, \hat{\mathbf{y}}(M)$, the overall model is trained by optimizing the following joint objective:

$$\mathcal{L}_{EE}(\hat{\mathbf{y}}(1), \ldots, \hat{\mathbf{y}}(M), \mathbf{r}) = \sum_{m=1}^{M} \mathcal{L}(\hat{\mathbf{y}}(m), \mathbf{r}), \tag{2}$$

where $\mathcal{L}(\hat{\mathbf{y}}(m), \mathbf{r}) = -\log P[\mathbf{r}|\mathbf{x}, \Theta_m]$, and Θ_m denotes the subset of parameters of Θ from the first to the m-th layer. In this work the encoder computes $\mathbf{h}_1^T(m), 1 \leq m \leq M$, and the decoders are linear layers with softmax function. They allow to estimate the probability of $\mathcal{Y} \cup \{\phi\}$ for each input frame, being ϕ a "blank" token indicating "no label issued". The intermediate loss function is the connectionist temporal classification (CTC) [16], defined as:

$$\mathcal{L}_{\text{CTC}}(\hat{\mathbf{y}}, \mathbf{r}) = -\log \sum_{\mathbf{a}_1^T \in \mathcal{B}^{-1}} \prod_{t=1}^{T} P[a_t|\mathbf{h}_1^T], \tag{3}$$

where $a_t \in \mathcal{Y} \cup \{\phi\}$. To ease the notation, in the equation above we have removed the dependence of all the variables on layer m.

IV. Proposed Architecture

While the Zipformer incorporates numerous blocks and modules specifically designed to optimize ASR performance in a "single exit" architecture, our study focuses solely on the integration of downsampling and upsampling operations within an EE architecture. To assess the effectiveness of this approach, we conducted a preliminary experiment comparing the performance of a "single exit" audio encoder, based on the U-net architecture, with that of a baseline "single exit" conformer-based encoder.

Conformer-baseline consists of a 1-D convolutional front end that downsamples the input sequence, represented by 80-dimensional Mel Frequency Cepstral Coefficients (MFCC), by

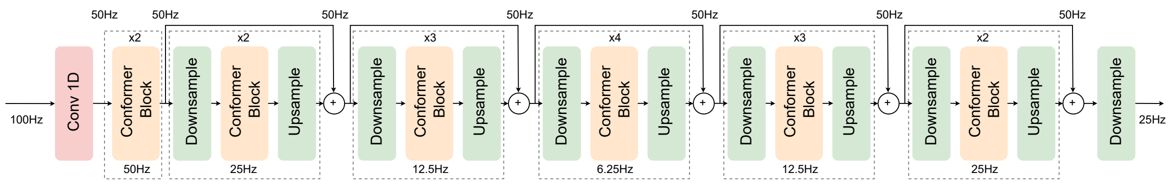


Fig. 2. The U-net modified architecture of the audio encoder.

a factor of two (from 100 Hz to 50 Hz), followed by a positional encoding module that feeds a stack formed by 12 conformer layers. In the "U-net modified" encoder architecture, shown in Figure 2, the output of the first two conformer layers is sent to a stack of 5 blocks, each formed by a variable number of conformer layers, with a residual connection between them. Similarly to zipformer each block is preceded and followed by downsampling and upsampling operations, respectively, that process the input stream at different frame rates, namely: 50 Hz, 25 Hz, 12.5 Hz and 6.25 Hz. In both architectures the optimized loss is the CTC loss evaluated from the output of the linear soft-max decoder (note that for the sake of clarity the decoder blocks are not depicted in the figures).

Table I reports the results achieved with these two architectures on the LibriSpeech benchmark (see Section V), where significant improvements yielded by the "U-net modified" encoder can be seen[1].

TABLE I
%WERs ACHIEVED WITH THE "SINGLE-EXIT" MODELS ON THE LIBRISPEECH EVALUATION DATA SETS.

Architecture	test-clean	test-other
Conformer-baseline	6.1	17.3
U-net modified	4.4	13.3

Therefore in order to exploit, similarly to U-net, variable time resolution processing in an early-exit architecture, we introduce *Splitformer*, the architecture shown in Figure 3, where: *a)* one exit decoder is inserted every two conformer layers, and *b)* the first and last encoder exits are computed by summing the outputs of a standard (two conformer layers) block and of one parallel layer that processes the input with a downsampling factor equal to 2. Despite its simplicity, the experimentation with different configurations, i.e. exploiting a different number of parallel branches and applying different values for the sampling factors, did not produce significant benefits, the architecture of Figure 3 being the best compromise between complexity and performance level.

Finally, in the experiments reported below *Splitformer* is compared with *EE-baseline*, derived from the Conformer-baseline by simply including one exit decoder every 2 conformer layers. In both of these EE architectures the optimized loss is the one of Equation 2.

[1]Note that the WERs of the original Zipformer architecture, as reported in [6], are 2.4% and 5.7% on test-clean and test-other, respectively. The performance discrepancy observed with respect to the two architectures presented in Table I, can primarily be attributed to the application of data augmentation techniques during the training of the Zipformer — techniques that were not employed when training either the Conformer-baseline or the "U-net modified" architectures.

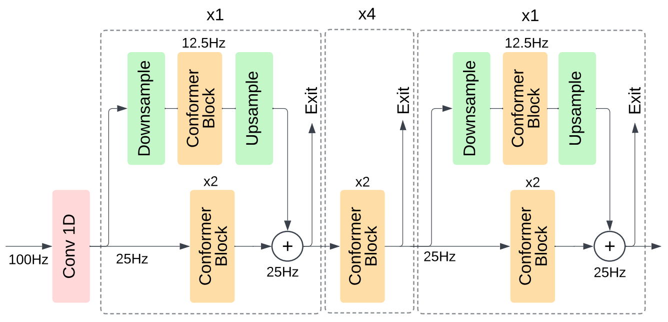


Fig. 3. The Splitformer architecture of the audio encoder.

V. Experiments

We carried out all our experiments on two well-established public benchmarks for ASR: LibriSpeech [17] and TEDLIUM [18] datasets. **LibriSpeech** contains ≈1,000 hours of read-aloud english audiobooks. The training set includes 2338 speakers and the evaluation set includes 146 speakers. We refer to [17] for further details and baseline performance. **TEDLIUM-V3** comprises of ≈452h of transcribed English speeches from TED video conferences for training and ≈6h for evaluation. Besides comparing our proposed Splitformer with its baseline counterpart (EE-baseline) both trained from scratch, we also experimented with two well known pre-trained models, namely: Wav2Vec2 [19] and WavLM [13], including early-exits in their architectures and fine-tuning their encoders/decoders layers. Also in this case, the decoders applied in each exit are linear layers followed by soft-max functions.

Table II provides details for all the architectures employed. Differently from the conformer based architectures (i.e. EE-baseline and Splitformer), both Wav2Vec2 and WavLM take as input the raw waveforms and use grapheme base tokenizers with 32 tokens (28 characters + 1 blank token + 2 sentence boundary tokens + 1 unknown token) per their official recipe.

Instead, both EE-baseline and Splitformer use a byte pair encoding (BPE) based tokenizers [20], with 256 tokens. For their training the learning rate followed the scheduling scheme reported in [15]. Specifically, we employed a number of warm-up steps equal to the size of each train dataloader (e.g., 17580 for the 960h LibriSpeech training set, with batch size equal to 16). We used the Adam optimiser [21] with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 1e^{-9}$. Dropout, with probability $p = 0.1$, was applied before the summation inside each conformer residual block. To limit the number of "pad" tokens in the batches, we removed from each audio dataset training samples having a

TABLE II
PARAMETERS OF THE EARLY-EXIT ARCHITECTURES USED IN THE EXPERIMENTS

Feature	EE-baseline	Splitformer	Wav2Vec2	WavLM (Base+)
# params (M)	31.5M	36.7M	94.0M	94.7M
Encoder	12-layer Conf.	14-layer Conf.	12-layer Transf.	12-layer Transf.
Attention dim.	256	256	768	768
Number heads	8	8	8	8
Feed-forward dim.	2048	2048	3072	3072
Decoder	Linear	Linear	Linear	Linear
Inputs	80-d MFCC	80-d MFCC	Waveform *	Waveform *
Loss function	$\mathcal{L}_{\text{CTC}}$	$\mathcal{L}_{\text{CTC}}$	$\mathcal{L}_{\text{CTC}}$	$\mathcal{L}_{\text{CTC}}$
Output tokens	BPE	BPE	Grapheme	Grapheme
LM rescoring	✗	✗	✗	✗
Data augmentation	✗	✗	✓	✓

TABLE III
%WERS ACHIEVED ON THE ON LIBRISPEECH EVALUATION SETS WITH THE DIFFERENT EARLY-EXIT MODELS.

Layer	EE-baseline		Splitformer		Wav2Vec2		WavLM	
	`test-clean`	`test-other`	`test-clean`	`test-other`	`test-clean`	`test-other`	`test-clean`	`test-other`
2	31.0	51.0	28.1	48.3	33.7	56.0	28.0	48.5
4	11.7	27.8	10.8	26.4	17.4	36.7	13.9	27.3
6	7.1	19.8	6.7	19.2	9.6	23.7	8.7	18.4
8	5.8	16.6	5.5	16.3	5.8	15.9	4.8	12.4
10	5.3	15.3	5.1	15.1	4.5	12.6	4.0	9.5
12	5.1	14.8	4.8	14.7	4.3	12.2	3.6	8.8

length greater than 600 characters. We have run 70 training epochs and for inference we averaged the models of the last 20 epochs.

To train both Wav2Vec2 and WavLM we freezed the features extraction layers of the pre-trained models (we used ”Wave2Vec2-base” model) and fine tuned optimizing the loss in equation 2. The parameters of the Wav2Vec2 model are those of the default configuration in the huggingface repository[2]. We trained it using 50 epochs. The code for: EE-baseline, Splitformer and Wav2Vec2 is available[3,4], while the fine-tuning of the WavLM model follows the related SpeechBrain recipe[5].

Finally, as mentioned in Section I, in this work we only focused on the overall EE model performance, without addressing the task of automatic exit selection according to some measure of reliability. For this topic we refer the reader to our previous work [1].

VI. RESULTS

Table III shows the results achieved with all the above mentioned models on the LibriSpeech evaluation data sets.

Notice that the Splitformer delivers superior performance than EE-baseline in all exits, more consistent in the lowest ones. Note also that the performance of Splitformer, in the lowest exits, are often better than those achieved with both Wav2Vec2 and WavLM, despite its lower number of parameters. In the upmost exits the pre-trained models exhibit significantly better performance, especially on the ”test-other” noisy dataset. However, it has to be considered that, differently from Wav2Vec2/WavLM, Splitformer has not been trained applying data augmentation. Finally, comparing the results of EE-baseline in Table III with those in Table I note also the superior performance of the EE loss of equation 2, as discussed in our previous paper [1].

TABLE IV
%WERS OBTAINED WITH EE-BASELINE AND SPLITFORMER ON THE TEDLIUM EVALUATION SETS.

Layer	EE-baseline		Splitformer	
	`dev`	`test`	`dev`	`test`
2	45.3	45.8	37.0	37.9
4	20.5	21.0	18.3	18.0
6	13.8	13.9	13.7	13.0
8	11.6	11.3	11.4	11.4
10	10.8	10.6	10.6	10.3
12	10.5	10.3	10.3	9.9

When considering only the EE-baseline and Splitformer architectures, the trends observed on the TEDLIUM dataset mirror those reported in Table III, as illustrated in Table IV. Also in this case, the performance improvements of Splitformer with respect to the baseline are noticeable, especially in the lowest exits. As a general remark notice that the improvements in the Splitformer are not only localized in the lowest exit which, due to the insertion of the parallel downsampling layer, employs 50% more parameters than the corresponding EE-baseline exit, but also propagate to the upmost exits.

A. Computational costs

Finally, we have carried out some comparisons in terms of the computational costs. Table V shows for each exit: the total execution time (including both CPU and GPU times) employed to generate the automatic transcripts and the number of tera-FLOPS spent in the corresponding encoder layers. The values of the Table have been computed on the whole evaluation sets of both LibriSpeech and TEDLIUM.

[2]https://huggingface.co/docs/transformers/model_doc/wav2vec2
[3]https://github.com/augustgw/early-exit-transformer
[4]https://github.com/augustgw/wav2vec2-ee
[5]https://github.com/speechbrain/speechbrain

TABLE V
TOTAL GPU/CPU EXECUTION TIMES (IN SECONDS) AND NUMBER OF TERA-FLOPS, SPENT IN THE ENCODER, NEEDED TO GENERATE THE TRANSCRIPTIONS IN EACH EXIT LAYER. THE VALUES HAVE BEEN COMPUTED ON THE WHOLE EVALUATION SETS OF BOTH LIBRISPEECH AND TEDLIUM. THE NUMBERS OF MODEL PARAMETERS BEFORE EXITING EACH LAYER ARE ALSO GIVEN.

Layer	**EE-baseline**					**Splitformer**				
	`LibriSpeech`		`TEDLIUM`			`LibriSpeech`		`TEDLIUM`		
	Time (sec)	**TFLOPS**	**Time** (sec)	**TFLOPS**	**#params**	**Time** (sec)	**TFLOPS**	**Time** (sec)	**TFLOPS**	**#params**
2	6112	10.5	3178	3.3	5.4M	5931	12.3	2890	3.9	8.0M
4	3217	20.2	1801	6.4	10.6M	3168	22.1	1830	7.0	13.2M
6	2090	30.0	1315	9.6	15.8M	2339	31.8	1283	10.1	18.4M
8	1660	39.7	1054	12.7	21.1M	1881	41.6	1144	13.2	23.7M
10	1414	49.5	1071	15.8	26.3M	1898	51.3	1080	16.4	28.9M
12	1343	59.3	1084	18.9	31.5M	1921	63.5	1081	20.2	36.7M

We observe the following two main results: *a)* overall computation times are much higher at lower outputs than at higher ones; *b)* the difference in computation times between the two architectures is quite small, the spliformer ones are slightly lower at the lowest exits, while they are a bit higher at the highest exits. The result coming from the first point above seems counter-intuitive. In fact one expects that processing time significantly increases at highest encoder layers, since more operations are needed to traverse all layers. Actually, as expected, the number of FLOPS required by the encoder increases linearly with the number of parameters. We attribute this behaviour to the pruning operations, applied to the "blank" tokens, in the CTC decoder. The decoder used in our experiments[6] prunes "blank" probabilities higher than 0.95 before expanding hypotheses in the CTC trellis. Since the number of "blank" tokens is the majority, their pruning has a strong impact on the overall computation time. Using the EE-baseline model we measured, over the test sets of LibriSpeech: *a)* around 83% of tokens generated in the upmost exit are "blank" tokens and *b)* those pruned in the last exit (i.e. above the threshold) are five times those of the first exit. Therefore, we speculate that the longer times required to traverse the encoder from bottom to top are counterbalanced by much shorter decoding times at the higher exits.

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, we have investigated early-exit architectures for ASR by comparing two architectures, one based on a stacked layer encoder, the other employing parallel layers that process the input at half of the frame rate. We have proven that the introduction of the parallel layers (the Splitformer architecture) improves the ASR performance in comparison with: *a)* the EE-baseline model and *b)* the lowest exits of two foundational pre-trained models. We have also shown that the computational cost of Splitformer is comparable with that of EE-baseline. In future work we will evaluate the effectiveness of the proposed approach on EE models with much fewer parameters (e.g. with lower attention dimension, or reducing the size of the feed-forward network, or the number of attention heads) and experiment on spoken language understanding domains.

REFERENCES

[1] G. A. Wright, U. Cappellazzo, S. Zaiem, D. Raj, L. O. Yang, D. Falavigna, M. N. Ali, and A. Brutti, "Training early-exit architectures for automatic speech recognition: Fine-tuning pre-trained models or training from scratch," in *Proc. of ICASSP Workshops*, 2024, pp. 685–689.
[2] T. Teerapittayanon, B. McDanel, , and H. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," *arXiv:1709.01686*, 2017.
[3] M. Phuong and C. H. Lampert, "Distillation-based training for multi-exit architectures," in *Proc. of ICCV*, 2019, pp. 1355–1364.
[4] S. Zaiem, R. Algayres, T. Parcollet, S. Essid, and M. Ravanelli, "Fine-tuning strategies for faster inference using self-supervised models: A comparative study," *arXiv:2303.06740*, 2023.
[5] R. Tang, K. V. S. M. Kumar, J. Xin, P. Vyas, W. Li, G. Yang, Y. Mao, C. Murray, and J. Lin, "Temporal early exiting for streaming speech commands recognition," in *Proc. of ICASSP*, 2022.
[6] Z. Yao, L. Guo, X. Yang, W. Kang, F. Kuang, Y. Yang, Z. Jin, L. Lin, and D. Povey, "Zipformer: A faster and better encoder for automatic speech recognition," in *Proc. of ICLR*, 2024.
[7] A. Gulati, J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," in *Proc. of Interspeech*, 2020.
[8] Y. Peng, S. Dalmia, I. Lane, and S. Watanabe, "Branchformer: Parallel MLP-attention architectures to capture local and global context for speech recognition and understanding," in *Proc. of ICML*, 2022.
[9] S. Kim, A. Gholami, A. Shaw, N. Lee, K. Mangalam, J. Malik, M. W. Mahoney, and K. Keutzer, "Squeezeformer: An efficient transformer for automatic speech recognition," in *Proc. of NeurIPS*, 2022.
[10] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[11] W. N. Hsu, B. Bolte, Y. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed, "HuBERT: self-supervised speech representation learning by masked prediction of hidden units," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 3451–3460, 2021.
[12] D. Berrebbi, B. Yan, and S. Watanabe, "Avoid overthinking in self-supervised models for speech recognition," in *Proc. of ICASSP*, 2022.
[13] S. Chen, 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, "WavLM: Large-scale self-supervised pre-training for full stack speech processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 16, no. 6, pp. 1505–1518, 2022.
[14] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. of of International Conference on medical image computing and computer-assisted intervention*, 2015, pp. 234–241.
[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. of NIPS*, 2017, pp. 6000–6010.
[16] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proc. of ICML*, 2006.
[17] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. of ICASSP*, 2015, pp. 5206–5210.
[18] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Esteve, "TED-LIUM 3: Twice as much data and corpus repartition for experiments on speaker adaptation," in *SPECOM*. Springer, 2018, pp. 198–208.
[19] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "Wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Proc. of NeurIPS*, 2020.
[20] S. R., B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proc. of ACL*, 2016, pp. 1715—1725.
[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[6]https://pytorch.org/audio/main/tutorials/asr_inference_with_cuda_ctc_decoder_tutorial.html