Fed-EE: Federating Heterogeneous ASR Models using Early-Exit Architectures

Mohamed Nabih Ali, Daniele Falavigna, Alessio Brutti*
Digital Society Center, Fondazione Bruno Kessler, Trento, Italy.
`mnabih,falavi,brutti@fbk.eu`

Abstract

Automatic speech recognition models require large speech recordings for training. However, the collection of such data often is cumbersome and leads to privacy concerns. Federated learning has been widely used as an effective decentralized technique that collaboratively learns a shared model while keeping the data local on clients devices. Unfortunately, client devices often feature limited computation and communication resources leading to practical difficulties for large models. In addition, the heterogeneity that characterizes edge devices make unpractical federating a single model that fits all the different clients. Differently from the recent literature, where multiple different architectures are used, in this work we propose using early-exiting. This brings 2 benefits: a single model is used on a variety of devices; federating the models is straightforward. Experiments on the public dataset TED-LIUM 3 show that our proposed approach is effective and can be combined with basic federated learning strategies. We also shed light on how to federate self-attention models for speech recognition, for which an established recipe does not exist in literature.

1 Introduction

Deep learning-based approaches are now widely employed for automatic speech recognition (ASR) [20], mainly using large centralized training sets [28, 1, 18]. Unfortunately, centralized training poses issues related to data ownership, data privacy, latency, and cost; these aspects gained increasing attention with the proliferation of both edge devices and low-latency communication technologies [17]. Therefore, distributed training approaches, such as federated learning (FL), have recently received more spotlight [5, 9]. As depicted in Fig. 1(a), FL is a distributed machine learning approach that aims to train models by combining pieces of information collected on the edge devices [23]. In details, in each FL round a set of clients performs local training using the locally acquired data and share with the central server the information needed to update the central models (e.g. gradients, weights, etc.). The server agglomerates the received updates and sends the models back to the clients to perform their processing. In this way, most of the computation is executed on the edge, preserving at the same time, private data to be transmitted over communication networks.

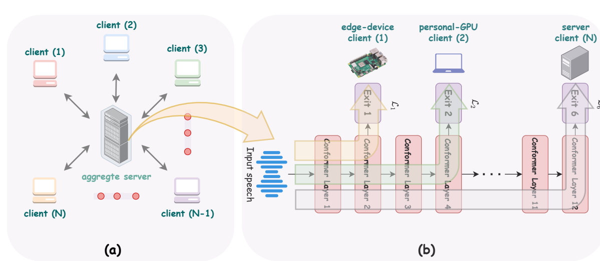

Figure 1: (a) FL scenario. (b) Example of EE architectures: different clients accommodate different exits.

*We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU

In a distributed environment edge devices exhibit a large variability of computational assets demanding for resource aware, i.e. client specific, neural networks. This issue can be handled by employing different architectures, with different resource requirements, and federating them via some shared layer or common processing blocks, as proposed in [49, 36, 4], eventually implementing articulated agglomeration strategies. Instead, in this work we propose to use early-exit (EE) architectures, that decode the outputs at different layers of an encoder, as depicted in Fig. 1(b). In this way, a single EE model is managed at the server side, while only the layers fitting the resources available on clients are actually processed locally. In particular, with this work we aim to contribute to the scientific literature in the three directions. **1)** We show that EE models allow federating heterogeneous models in a rather straightforward way. This surpasses the need for multiple models centrally aligned with edge-specific solutions of current approaches, as in [5]. **2)** The literature on FL for ASR is not uniform in relation to model pretraining and centralized training on held-out data [9, 32]. We experimentally confirm that in a cross-domain framework, pretraining even on out-of-domain data is indeed necessary, but the role of central training seems not to be crucial. **3)** We show that using the FedAdam agglomeration strategy [39] in combination with freezing part of the pretrained model (pretrained in an EE fashion) noticeably helps the convergence. Note that although FedAdam has been already used in literature, its efficacy on ASR tasks with EE architectures has never been experimentally verified.

2 Related Works

In speech processing, federated learning has been applied to several tasks: ASR [13, 46], keyword spotting [22, 27, 14], speaker recognition [45] and other applications [21, 10, 8]. Generally, FL for ASR is a challenging task. Other than for the classic non-i.i.d and unbalancing data distributions, the main critical issues are: *a)* most of ASR architectures (e.g Transformers [48], Transducers [30, 47] and recurrent neural networks [34]) require powerful computational resources which are not available in most edge devices, and *b)* learning, from scratch, the proper alignment between the latent speech representation and the transcription [40] is unfeasible in a FL framework, since it requires large datasets, usually available only in the central servers. In [9], the authors prove the need to pretrain the global model in order to reach convergence and introduce a held-out data set, to be employed after FedAvg, to control model divergence between adjacent FL rounds. They also apply some (client-specific) weighting strategies in FedAvg agglomeration, showing the superior performance of word error rate (WER)-based weighting compared to loss-based weighting. Similar trends of results have been observed on LibriSpeech [35], as reported in [6]. [32] investigated the use of a global model initialized as in [9] or based on a pretrained self-supervised model (Wav2Vec 2.0 [2]), using FL to adapt to the TEDLIUM-3 dataset [15]. While FL did not improve the WER of the former model, the latter has demonstrated effective. In the following, we show that our proposed EE model allows applying FL on TEDLIUM-3 without the need of a large self-supervised pretrained model (pretraining is on LibriSpeech).

2.1 Federated learning for Heterogeneous models

Most FL approaches assume that all clients are "homogeneous" in their computational assets. However, this hypothesis cannot be applied in real scenarios, where devices may have severe limitations in their memory, computation capabilities, and power consumption and are characterized by a dynamic usage of the resources. Therefore, FL frameworks require managing multiple different "heterogeneous" architectures, under the guidance of personalized tasks and local resource constricts [42].

Previous works have addressed the client heterogeneity by employing multiple different architectures all sharing a common part. In particular: *a)* mixed model architectures [29], where local models share only a subset of parameters with the central model, *b)* knowledge distillation [33, 26] at the client side to preserve the global model parameters while learning local information, *c)* the usage of a contrastive loss [25] to decrease a distance metric between central and local models, and *d)* federated ensemble knowledge transfer (Fed-ET) [5], that uses a consensus distillation, derived from clients, to train a large model on the server (also in this case networks share some common layers).

With respect to the previous works, we propose using EE architectures that introduce intermediate exit branches [43, 37] to the network (see Fig. 1(b)): the input is processed by a subset of the layers of the neural network, resulting in multiple scaled versions of the same architectures. In section 3.1 we show that the EE architecture allows to agglomerate coherently the local model parameters at the server side and to further extract from the global model the suitable sub-models to be sent to the connected clients.

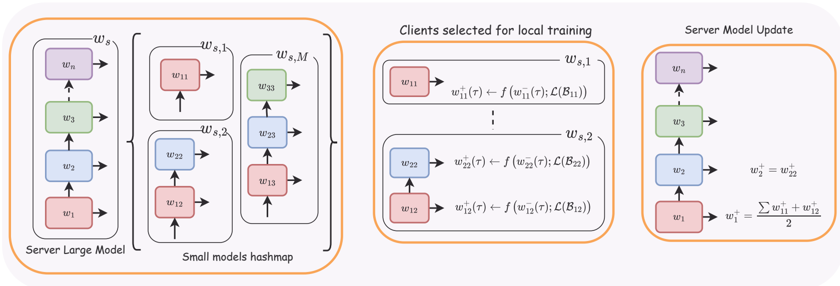

Figure 2: Illustration of our proposed approach for agglomerating heterogeneous models.

3 Proposed Approach

The idea of FL is to train multiple versions of a neural model on the client side and then agglomerate them on the server side. Let us assume that C clients are available at round τ, and each client $c \in \{1, ..., C\}$ observes its own training set $\mathcal{B}_c$. At the beginning of each round, clients receive the most recently trained model stored on the central server $\mathbf{w}_c^-(\tau) = \mathbf{w}_s(\tau-1)$. Note that an important requirement here is that all clients can accommodate the same architecture. Each model is updated using the local dataset $\mathbf{w}_c^+(\tau) \leftarrow f\left(\mathbf{w}_c^-(\tau); \mathcal{L}(\mathcal{B}_c)\right)$, where $\mathcal{L}(\mathcal{B}_c)$ is the loss computed on the dataset of client c and $f(\cdot)$ is a weight update strategy (i.e. SGD, ADAM, etc.). Local models are in turn used to update the central model $\mathbf{w}_s(\tau)$.

Recently, several works have been published to find the optimal strategy to agglomerate clients' model parameters (weights) [5, 38, 31] or to improve the type and amount of information shared between the clients and the server [24]. Federated averaging strategy (FedAvg) [19], which is based on FedSGD [7], is one of the most common strategies as it agglomerates the models by simple weighted averaging: $\mathbf{w}_s(\tau) = \frac{1}{C}\sum_{c=1}^{C} \eta_c(\tau)\mathbf{w}_c^+(\tau)$, where the weights $\eta_c(\tau)$ (such that $\sum_{c=1}^{C} \eta_c(\tau) = 1$) are estimates of the client confidence, eventually related to the size of the dataset $\mathcal{B}_c$, the loss $\mathcal{L}(\mathcal{B}_c)$ or the accuracy on a local or centralized development set.

One of the limitations of FedAvg is the need to use SGD on the clients in order to allow an effective averaging of the different models, affecting the overall convergence. FedAdam [39] is an alternative agglomeration strategy that updates the weights using one-step adaptive gradient optimization [44].

3.1 Federated Learning with Early-Exit models

In the presence of devices with different processing capabilities, using a single model $\mathbf{w}$ is not feasible. As mentioned above, current approaches employ U different networks $\mathbf{w}^u$, $u = [1, \dots, U]$ with different computation requirements which are all maintained on the centralized server. This solution requires managing multiple different models, with varying performance, and adopting articulated strategies for agglomerating them.

As mentioned in section 2.1 an interesting solution is offered by EE architectures. Let us assume that model $\mathbf{w}_s$ is split in M subnets $\mathbf{w}_{s,1}, \mathbf{w}_{s,2}, \dots, \mathbf{w}_{s,M}$ (not necessarily of the same type) each of them equipped with an exit layer (producing hypothesis $\hat{\mathbf{y}}^1, \dots, \hat{\mathbf{y}}^M$). The overall model is trained by optimizing the joint objective $\mathcal{L}_{EE}(\hat{\mathbf{y}}^1, \dots, \hat{\mathbf{y}}^M, \mathbf{y}) = \sum_m \mathcal{L}(\hat{\mathbf{y}}^m, \mathbf{y}) = \sum_m \mathcal{L}_m(\mathcal{B})$, where $\mathcal{L}(\hat{\mathbf{y}}^m, \mathbf{y})$ and $\mathbf{y}^m$ are the loss and the prediction of the m-th, while $\mathbf{y}$ is the ground-truth label.

It is worth noting that besides reducing the actual number of models, EE allows applying standard agglomeration strategies such as FedAvg and FedAdam. More in detail, each client c is equipped with a model $\mathbf{w}_{c,i}$, which includes all sub-nets up to $i \in [1, M]$. According to the notation above the weight updates are done as $\mathbf{w}_{c,i}^+(\tau) \leftarrow f\left(\mathbf{w}_{c,i}^-(\tau); \sum_{m=1}^{i} \mathcal{L}_m(\mathcal{B}_c)\right)$. Then the updated local weights are sent to the server where they are averaged according to all sub-nets. Fig.2 shows the graphical representation of the proposed approach in the case of only two clients. Note that while the subnet $\mathbf{w}_{s,1}$ is present in all clients, higher subnets ($\mathbf{w}_{s,M}$) are less frequent and may be not updated. Nevertheless, if $C \gg M$ it is likely that all subnets are present in some clients.

Table 1: WER of the model pretrained on Librispeech and TEDLIUM-3 tested on both Librispeech and TEDLIUM-3. The third column reports the heterogeneous FL performance using FedAdam and freezing the convolutional layers.

Train set	Librispeech-960		FedAdam - Hetero -freeze	TED-LIUM
Test set	Libri-Test-Clean	TED-LIUM-Test	TED-LIUM-Test @ 1400 rounds	TED-LIUM-Test (upper-bound)
exit 1	24.10	50.94	45.40	43.8
exit 2	11.78	34.50	29.93	23.4
exit 3	6.91	29.58	24.31	18.0
exit 4	6.28	28.61	23..43	16.1
exit 5	7.30	28.79	23.36	14.9
exit 6	5.34	27.35	21.83	14.6

4 Experimental Setup

We evaluate our proposed approach using the TEDLIUM-3 corpus [15] that contains TED talks with a total amount of 452 hours of speech data in English from about 2351 speakers. As previously mentioned and as observed in previous studies [32], training an ASR model from scratch in a federated fashion is unfeasible. Therefore, we pretrain our model using the whole training set (960 hours) of Librispeech. Following the best practice in literature and in an attempt to make the scenario as realistic as possible, the TEDLIUM training set is split such that each client sees data of a single speaker (this way mimicking personal devices). Performance is measured in terms of WER on the test set for each of the M exits of the resulting agglomerated model. Note that, differently from the current literature, we adapt the initial model, trained on LibriSpeech, to a new domain ("TEDLIUM"), instead of applying FL to the same domain ("LibriSpeech"). For ASR, we use the EE architecture depicted in Fig. 1(b): it consists of a stack of conformer layers with intermediate linear decoders every other conformer.[2] Further details are given in Sec. A.1.

We implement the FL framework using the Flower toolkit [3]. We deploy 2351 clients (one for each speaker in the training set). In each round, 10% of all available clients are randomly instantiated and used to train the model locally. Local training implements SGD for 5 epochs using a learning rate equal to 0.01. Models are centrally agglomerated using either FedAvg or FedAdam. Finally, we consider a classic scenario where *homogeneous models* are used (i.e. the full early-exit architecture) as well as the case where models are *heterogeneous*. In the latter, the number of exits available at each client randomly varies across clients and rounds with a uniform probability distribution. Finally, in order to improve the model convergence, we also experiment with freezing the convolutional front end of the pretrained model, training only the encoder-decoder part. Our code is publicly available[3].

4.1 Experimental Results

Table 1 reports the performance of our approach considering heterogeneous architectures. The first column reports the performance of our model pretrained and tested on Librispeech confirming that it is a solid baseline. The second column reports the performance of the pretrained model on TEDLIUM-3: this is the starting point for our FL experiment. The third column reports the WER obtained with 1400 FL rounds using heterogeneous architectures with FedAdam and freezing the feature extractor, while the last column reports the upper-bound on TEDLIUM-3 when applying central training. The first interesting result is that even if heterogeneous devices are used (i.e. the whole architecture is not available at all clients) agglomerating the models with a standard FL approach is viable: note that WERs are considerably better than those of the initial model at all exits.

Fig. 3 compares the WER obtained with heterogeneous and homogeneous architectures as a function of the FL rounds for 3 exits: 1,3 and 6. Note that the performance in both cases are very similar using either FedAvg (lines orange and black) or FedAdam (lines green and red) for all exits. This further confirms the efficacy of EE models in this scenario. The figure also confirms that both FedAdam and freezing the convolutional layers noticeably speed up the convergence of the model with respect to FedAvg. As a matter of fact, the convolutional front-end is in charge of extracting robust features for speech recognition, so it makes sense that it does not need to be adapted to a new speech domain.

[2] https://github.com/augustgw/early-exit-transformer

[3] https://github.com/mnabihali/ASR-FL

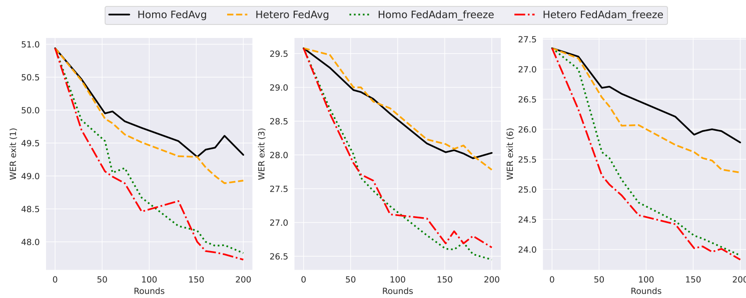

Figure 3: WER achieved with homogeneous and heterogeneous models, using FedAvg and FedAdam with freezing the convolutional front-end. Three different exits are reported.

5 Conclusions

In this paper, we have presented an investigation on FL for ASR in the presence of heterogeneous clients using early-exit architectures. The experimental results obtained on popular benchmarks proved the efficacy of the EE models in this scenario. Differently from other works in the literature, we employ a pre trained EE model from out-of-domain data. We demonstrate that centralized training is not crucial for model convergence. Finally, we observed significant performance improvements when freezing the convolutional layers of the pretrained model.

References

[1] Mohamed Nabih Ali et al. Direct enhancement of pre-trained speech embeddings for speech processing in noisy conditions. *Computer Speech & Language*, 81:101501, 2023.

[2] Alexei Baevski et al. Wav2Vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[3] Daniel J Beutel et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[4] Hyunsung Cho et al. Flame: Federated learning across multi-device environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(3):1–29, 2022.

[5] Yae Jee Cho et al. Heterogeneous ensemble knowledge transfer for training large models in federated learning. *arXiv preprint arXiv:2204.12703*, 2022.

[6] Dimitrios Dimitriadis et al. A Federated Approach in Training Acoustic Models. In *Proc. of Interspeech*, pages 981–985, 2020.

[7] Mohammad Navid Fekri et al. Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks. *International Journal of Electrical Power & Energy Systems*, 137:107669, 2022.

[8] Meng Feng et al. Federated self-supervised learning for acoustic event classification. In *Proc. of ICASSP*, pages 481–485. IEEE, 2022.

[9] Yan Gao et al. End-to-end speech recognition from federated acoustic models. In *Proc. of ICASSP*, pages 7227–7231. IEEE, 2022.

[10] Yan Gao et al. Federated self-supervised speech representations: Are we there yet? *arXiv preprint arXiv:2204.02804*, 2022.

[11] A. Graves et al. Speech recognition with deep recurrent neural networks. In *Proc. of ICASSP*, pages 6645–6649, 2013.

[12] A. Graves and N. Jaitly. Towards End-To-End Speech Recognition with Recurrent Neural Networks. In *Proc. of ICASSP*, 2014.

[13] Dhruv Guliani et al. Enabling on-device training of speech recognition models with federated dropout. In *Proc. of ICASSP*, pages 8757–8761. IEEE, 2022.

[14] Andrew Hard et al. Production federated keyword spotting via distillation, filtering, and joint federated-centralized training. *arXiv preprint arXiv:2204.06322*, 2022.

[15] François Hernandez et al. TED-LIUM 3: Twice as much data and corpus repartition for experiments on speaker adaptation. In *Speech and Computer: International Conference, SPECOM, Leipzig, Germany*, pages 198–208. Springer, 2018.

[16] Ning Huang et al. Wireless federated learning with hybrid local and centralized training: A latency minimization design. *IEEE Journal of Selected Topics in Signal Processing*, 17(1):248–263, 2022.

[17] Renuga Kanagavelu et al. Two-phase multi-party computation enabled privacy-preserving federated learning. In *IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, pages 410–419. IEEE, 2020.

[18] Lokesh Khurana et al. Speech recognition with deep learning. In *Journal of Physics: Conference Series*, volume 1854, page 012047. IOP Publishing, 2021.

[19] Jakub Konečnỳ et al. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[20] Akshi Kumar et al. A survey of deep learning techniques in speech recognition. In *ICACCCN*, pages 179–185. IEEE, 2018.

[21] Siddique Latif et al. Federated learning for speech emotion recognition applications. In *IPSN*, pages 341–342. IEEE, 2020.

[22] David Leroy et al. Federated learning for keyword spotting. In *Proc. of ICASSP*, pages 6341–6345. IEEE, 2019.

[23] Li Li et al. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[24] Qinbin Li et al. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[25] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10713–10722, 2021.

[26] Tao Lin et al. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020.

[27] Nil Llisterri Giménez et al. On-device training of machine learning models on microcontrollers with federated learning. *Electronics*, 11(4):573, 2022.

[28] Ambuj Mehrish et al. A review of deep learning techniques for speech processing. *Information Fusion*, page 101869, 2023.

[29] Jed Mills, Jia Hu, and Geyong Min. Multi-task federated learning for personalised deep neural networks in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):630–641, 2021.

[30] Takafumi Moriya et al. Improving scheduled sampling for neural transducer-based asr. In *Proc. of ICASSP*, pages 1–5. IEEE, 2023.

[31] Hung T Nguyen et al. Fast-convergent federated learning. *IEEE Journal on Selected Areas in Communications*, 39(1):201–218, 2020.

[32] Tuan Nguyen et al. Federated learning for ASR based on Wav2Vec 2.0. In *Proc. of ICASSP*, pages 1–5. IEEE, 2023.

[33] Xuanming Ni et al. Federated optimization via knowledge codistillation. *Expert Systems with Applications*, 191:116310, 2022.

[34] Jane Oruh, Serestina Viriri, and Adekanmi Adegun. Long short-term memory recurrent neural network for automatic speech recognition. *IEEE Access*, 10:30069–30079, 2022.

[35] Vassil Panayotov et al. Librispeech: an asr corpus based on public domain audio books. In *Proc. of ICASSP*, pages 5206–5210. IEEE, 2015.

[36] JaeYeon Park and JeongGil Ko. FedHM: Practical federated learning for heterogeneous model deployments. *ICT Express*, 2023.

[37] M. Phuong and Lampert. Distillation-based training for multi-exit architectures. In *Proc. of ICCV*, 2019.

[38] Krishna Pillutla et al. Robust aggregation for federated learning. *IEEE Transactions on Signal Processing*, 70:1142–1154, 2022.

[39] Sashank Reddi et al. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[40] Andrew Rosenberg et al. End-to-end speech recognition and keyword search on low-resource languages. In *Proc. of ICASSP*, pages 5280–5284. IEEE, 2017.

[41] Rico Sennrich et al. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[42] Alysa Ziying Tan et al. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[43] T. Teerapittayanon et al. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. *arXiv:1709.01686*, 2017.

[44] Yujia Wang et al. Communication-efficient adaptive federated learning. In *International Conference on Machine Learning*, pages 22802–22838. PMLR, 2022.

[45] Abraham Woubie and Tom Bäckström. Federated learning for privacy-preserving speaker recognition. *IEEE Access*, 9:149477–149485, 2021.

[46] Wentao Yu et al. Federated learning in ASR: Not as easy as you think. In *Speech Communication; ITG Conference*, pages 1–5. VDE, 2021.

[47] Mohammad Zeineldeen et al. Conformer-based hybrid ASR system for switchboard dataset. In *Proc. of ICASSP*, pages 7437–7441. IEEE, 2022.

[48] Albert Zeyer et al. A comparison of transformer and lstm encoder decoder models for asr. In *ASRU*, pages 8–15. IEEE, 2019.

[49] Shuai Zhu et al. On-device training: A first overview on existing systems. *arXiv preprint arXiv:2212.00824*, 2022.

A Supplementary Material

A.1 ASR model

For the ASR model, we use the early-exit architecture shown in Fig. 1(b). The network takes as input 80 Mel Frequency Cepstral Coefficients (MFCCs). This MFCC sequence is passed through a series of a stack of $N = 12$ conformer layers with $M = 6$ intermediate linear decoders (one every 2 conformer layers, $M = \frac{N}{2}$). The optimal sequence of labels in each decoder is chosen by means of the CTC algorithm [11, 12]. The model uses a byte pair encoding (BPE) based tokenizer [41] with 256 tokens. Table 2 summarizes the main hyperparameters for the model.

Table 2: Hyperparameters for the early-exit model architecture shown in Fig.1(b).

Params	Encoder	Attention dim.	# of heads	Feed-forward dim	Decoder	Input	Loss	Output units	LM scoring
31 M	12-layers	256	8	2048	Linear	80-MFCC	CTC	BPE (256)	✗

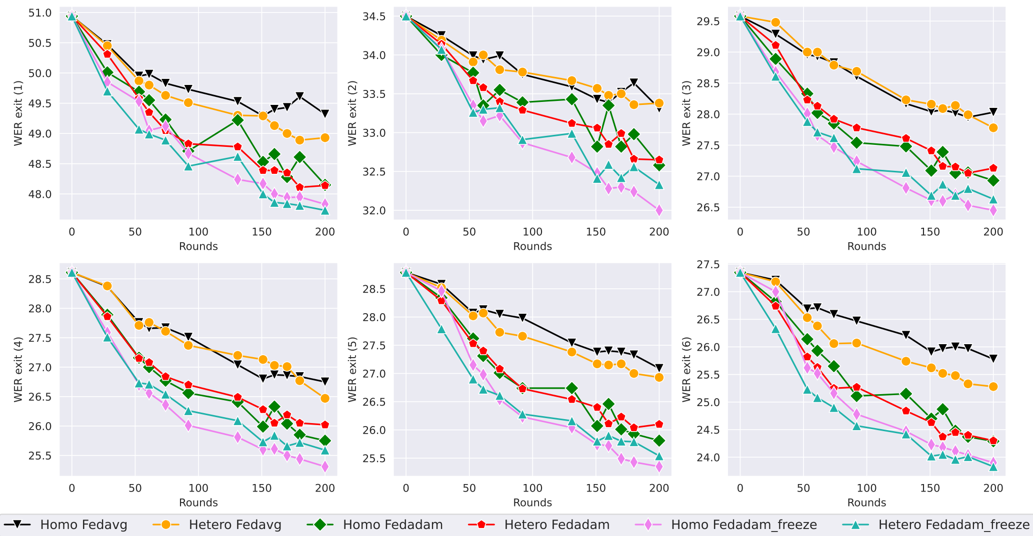

Figure 4: WER achieved by different FL strategies on TEDLIUM 3. The figure shows results with homogeneous and heterogeneous models, using FedAvg and FedAdam, as well as freezing the convolutional front-end.

A.2 Further experimental results

Figure 4 complements the results reported above with the performance of FedAdam without freezing and considering all the exits. Note that both FedAdam and freezing the convolutional layers contribute to improving the convergence of the model.

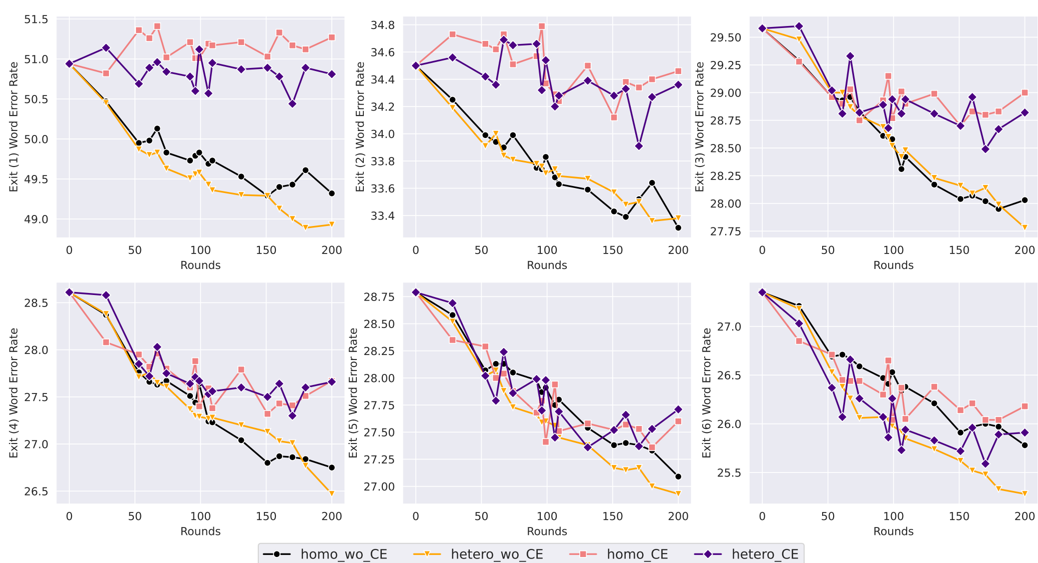

Figure 5: WER using FedAvg strategy with and without a central training stage.

A.3 On the use of central training

We also experiment with centrally training the agglomerated model using the TEDLIUM 3 development set. This is a common practice in literature when a pretrained model is available and FL is applied on data from the same domain [9, 16]. Typically a part of the training set is held out for the central training whose main role is to avoid the model diverges. In our scenario, we used the development set of TEDLIUM-3 as central training: it includes 8 speakers for a total of 1.6 hours. After agglomerating the local models we run 15 epochs of SGD on the development set. Fig. 5 shows the results. Note that for the last two exits the use of central training does not bring any improvement over a basic FedAvg method. Interestingly, for earlier exits the use of a small set for central training is instead detrimental. The reason behind this behavior is that, the TEDLIUM-3 development set has not enough samples to train the server large model. Hence, the server model tends to overfit the development set.