**Project 1: Mamba model for speech tasks**

In recent years, there has been a growing interest in a new class of deep learning models for sequence processing, called [State Space Models](https://arxiv.org/abs/2111.00396) (SSM). These models effectively combine ideas derived from recurrent and convolutional neural networks and aim to mitigate some limitations of Transformers, such as the quadratic memory bottleneck that prevents them from processing very long sequences. So far, standard SSM models have limited success in practical application, and Transformers are still by far the dominant model for sequence processing.

Nevertheless, a recent paper introduced an extension of SSM called [Mamba](https://arxiv.org/pdf/2312.00752.pdf), which, according to the authors, significantly improves their performance and speed in different tasks. Compared to previous SSMs, Mamba allows the parameters to be functions of the input (i.e., the parameters are not the same for all time steps, because they dynamically change while processing the input sequence). The authors also propose an efficient hardware-friendly implementation of the model. Mamba spawned a lot of interest in the deep learning community, and many researchers are trying this model on their task.

The goal of this project is to evaluate the Mamba's performance for various speech processing tasks available in [SpeechBrain](https://github.com/speechbrain/speechbrain), with a particular focus on speech recognition.

The proposed plan is the following:

1. Conduct a comprehensive literature review on SSMs. Carefully read the Mamba paper and the follow-up papers.
2. Familiarize yourself with the [code](https://github.com/state-spaces/mamba) made available by the author. Make sure you can run it on your hardware. Learn how to import this model into the existing SpeechBrain recipes.
3. Now, you can focus on assessing the performance of Mamba on different speech processing tasks. You need to compare the performance of Mamba with that achieved by the available [recipes](https://github.com/speechbrain/speechbrain/tree/develop/recipes) of SpeechBrain (in particular, we are interested in a comparison with transformer-based recipes). You can start with speech recognition recipes, such as [LibriSpeech](https://github.com/speechbrain/speechbrain/tree/develop/recipes/LibriSpeech/ASR/transformer) or [CommonVoice](https://github.com/speechbrain/speechbrain/tree/develop/recipes/CommonVoice/ASR/transformer). You are welcome to explore as many speech processing tasks as you want. For instance, you can consider TTS with [LJSpeech](https://github.com/speechbrain/speechbrain/tree/develop/recipes/LibriTTS), emotion recognition with [IEMOCAP](https://github.com/speechbrain/speechbrain/tree/develop/recipes/IEMOCAP/emotion_recognition), keyword spotting with [Google Speech Commands](https://github.com/speechbrain/speechbrain/tree/develop/recipes/Google-speech-commands), language ID with [CommonLanguage](https://github.com/speechbrain/speechbrain/tree/develop/recipes/CommonLanguage), etc.

Note that this project requires much more than simply plugging the Mamba model into existing recipes. Very often, when dealing with new models on new tasks, a lot of work should be devoted to adapting the models and finding proper values of the hyperparameters to make them converge and perform well. As such, the project can be time and computationally demanding. Please, contact the instructor if you feel like you need access to more computational resources that are currently available to you.

**Project 2: Keyword Spotting with Discrete Representations**

In recent years, there has been a growing interest in discrete audio representations, also called audio tokens. These representations involve audio features that are inherently discrete, meaning they have a limited codebook to capture relevant information in the audio waveform. This differentiates them from traditional features like FBANKs or MFCCs, as well as self-supervised features such as [wav2vec](https://arxiv.org/abs/2006.11477), [Hubert](https://arxiv.org/abs/2106.07447), and [WavLM](https://arxiv.org/abs/2110.13900), which are continuous in nature. The appeal of discrete audio representations lies in their potential benefits, such as improved integration with large multimodal language models and NLP pipelines. Additionally, they transform regression problems into classification ones, which are generally easier to handle. However, a significant challenge is that the performance of discrete representations has yet to match that of continuous representations. The research community is actively addressing this challenge, and there are ongoing efforts to enhance the performance of discrete audio representations. Open-source models like [EncoDec](https://arxiv.org/abs/2210.13438) and [DAC](https://arxiv.org/abs/2306.06546) have been proposed as part of these efforts.

The goal of this project is to evaluate the performance of discrete audio representations for keyword spotting. The performance must be compared with that achieved by standard features and by popular self-supervised models such as Wav2vec, Hubert, and WavLM.

This is the proposed plan:

1. Conduct a comprehensive literature review on discrete audio representations.
2. For this study, use the [Google Speech Command Dataset](https://www.tensorflow.org/datasets/catalog/speech_commands). Take a look at the existing [recipe](https://github.com/speechbrain/speechbrain/tree/develop/recipes/Google-speech-commands) already available in SpeechBrain, run them, and make sure the performance is similar to the expected one (mentioned in the readme file).
3. Modify the existing recipe by replacing standard FBANK features with self-supervised features such as Wav2vec, Hubert, and WavLM. Note that these features are already supported in [SpeechBrain](https://github.com/speechbrain/speechbrain/tree/develop/speechbrain/lobes/models). On top of these features, you likely need to plug something different than the [xvector](https://www.danielpovey.com/files/2018_icassp_xvectors.pdf) model used for standard features. Try different simple architecture, such a simple MLP or a linear layer coupled with average pooling and take the best one. Compare the performance achieved by freezing or fine-tuning the self-supervised features.
4. Now, design a system that works well with discrete audio features. Consider both EncoDec and DAC features (already supported in SpeechBrain). On top of that, design an architecture that achieves the best performance. You can compare different types of neural networks, such as RNNs, Transformers, CNNs, MLPs, etc.
5. The performance can be very good (e.g, > 98%) with current systems. To make the task a bit harder, as a last step, we also ask the students to compare their models after adding some noise and reverberation to the training, validation, and test sentence. You can use the data augmentation engine available in SpeechBrain for that.

**Project 3: Imagined Speech Recognition**

This project focuses on developing a system for converting imagined speech into corresponding text. The system leverages brain activity recorded through EEG signals, along with machine learning algorithms to predict speech commands. This technology could benefit individuals who cannot speak by allowing them to think and execute commands.

Building upon recent advancements in the field, the project will be implemented using the [SpeechBrain](https://github.com/speechbrain/speechbrain) or [SpeechBrain-MOABB](https://github.com/speechbrain/benchmarks/tree/main/benchmarks/MOABB) framework. The latter library is designed to simplify the development and evaluation of neural networks for multiple EEG tasks, including motor imagery. [SpeechBrain-MOABB](https://github.com/speechbrain/benchmarks/tree/main/benchmarks/MOABB) efficiently handles various aspects related to the EEG pipeline, including data loading, preprocessing, data augmentation, batching, training/evaluation loops on different subjects, multiple seed evaluations, hyperparameter tuning, and more. Consequently, students can primarily concentrate on designing their neural networks.

A newly released dataset for inner speech recognition is the following:

* Paper: <https://www.nature.com/articles/s41597-022-01147-2>
* GitHub: <https://github.com/N-Nieto/Inner_Speech_Dataset>

The plan of action is as follows:

1. Conduct a comprehensive literature review on imagined speech recognition, focusing on studies that use the newly released dataset (e.g., [this paper](https://arxiv.org/pdf/2210.06472.pdf)).
2. Familiarize yourself with SpeechBrain-MOAB .Explore the comprehensive README.md file in the [SpeechBrain-MOABB](https://github.com/speechbrain/benchmarks/tree/main/benchmarks/MOABB) repository for details about the toolkit and EEG processing. Utilize the provided tutorials to gain a better understanding of how the library operates.
3. Learn how to import the Inner Speech Dataset into SpeechBrain-MOABB. This dataset is not natively supported by MOABB and you thus need to create a proper dataloader for it.
4. Write recipe for standard models already available in SpeechBrain-MOABB such as [EEGNET](https://arxiv.org/abs/1611.08024), [ShallowNET](https://arxiv.org/pdf/1703.05051.pdf), [EEGConformer](https://ieeexplore.ieee.org/document/9991178). Make sure to use the experimental protocol proposed in SpeechBrain-MOAB, which includes multi-seed evaluation and mutti-step hyperparameter tuning (see the readme file of the SpeechBrain-MOAB repo for more information).
5. Seek to improve the model's performance by exploring other models or variations of the models already available.

**Project 4: Audio-Visual Speech Recognition**

Conventional speech recognition systems rely solely on audio inputs. However, incorporating visual information, such as lip reading, can greatly enhance the accuracy of speech recognition in noisy environments. The objective of this project is to create an audio-visual speech recognition system using the [Lip Reading Sentences 2 (LRS2) Dataset](https://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrs2.html) within the SpeechBrain framework.

The project plan is as follows:

1. Conduct a comprehensive review of the literature on lip reading and audio-visual speech recognition.
2. Implement simple baselines for lip reading using a visual-only approach, for instance by replicating the system proposed [here](https://arxiv.org/pdf/1809.02108.pdf) and compare the results with state-of-the-art methods.
3. Set up a speech recognition baseline using audio-only modality. Utilize pre-trained models such as whisper or wav2vect in SpeechBrain or fine-tune them using the LRS2 dataset if necessary. Compare the results with state-of-the-art methods.
4. Integrate the audio and visual modalities to create an audio-visual speech recognizer, either by implementing a new model or using a pre-existing one in the literature. Compare the results with state-of-the-art methods.

The project is computationally demanding. Make sure you have access to GPUs (e.g, ColabPro or Gradient) for the duration of the project. The dataset is pretty large too (make sure you have enough space to store it). Contact the instructor if you think you need more computational resources than what you have available.

**Project 5: Efficient Neural Networks for Speech Recognition**

This project aims to develop efficient neural networks for speech recognition, addressing the challenge of large, effective models being difficult to implement in production due to high inference costs.

The project plan is as follows:

1. Conduct a literature review on efficient neural networks in speech recognition.
2. Implement the [SqueezeFormer](https://arxiv.org/abs/2206.00888), integrating it into the [librispeech CTC recipe](https://github.com/speechbrain/speechbrain/tree/develop/recipes/LibriSpeech/ASR/CTC) in SpeechBrain, taking inspiration from the [Nemo implementation](https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/asr/intro.html). Evaluate and optimize performance.
3. Implement [Citrinet](https://arxiv.org/abs/2104.01721), an efficient end-to-end speech recognition network, again using SpeechBrain for evaluation and optimization. Make sure the results are comparable to that obtained in Nemo. Tune the hyperparameters to improve the performance as much as possible.
4. Implement one additional efficient model, either a new design or a reimplementation of an existing one.

Note that access to GPUs (e.g., ColabPro or Gradient) is required for the duration of the project as the project is computationally demanding. Contact the instructor if you think you need more computational resources than what you have available.

**Project 6: TTS Models (VITS + TransformerTTS)**

Transformers are flexible models designed for handling sequence-to-sequence processing, making them adaptable for a diverse range of tasks. In the context of this project, the primary objective is to create a text-to-speech model leveraging the Transformer architecture. In particular, the students have to implement in SpeechBrain the popular Transformer TTS model (or any recently proposed TTS system based on transformers).

The project plan is as follows:

1. Conduct a literature review on Transformers in text-to-speech.
2. Implement the Transformer TTS model described in [this paper](https://arxiv.org/pdf/1809.08895.pdf), drawing inspiration from the [Tacotron2](https://github.com/speechbrain/speechbrain/tree/develop/recipes/LJSpeech/TTS/tacotro) recipe in SpeechBrain. The model will be trained and evaluated on the [LJSpeech dataset](https://keithito.com/LJ-Speech-Dataset/), with performance evaluated against [Tacotron2 samples](https://drive.google.com/drive/folders/1CbkXPvtLFVrRBeeuMnmTmNCyagNKO6uX?usp=sharing).
3. Optimize the model through hyperparameter tuning, striving for the best performance.
4. Develop an inference API for the final model (that should be similar to [this one](https://huggingface.co/speechbrain/tts-tacotron2-ljspeech)).

Upon completion, the best model will be included in the main SpeechBrain project and made available on the SpeechBrain HF repository with an inference interface. Note that access to GPUs (e.g., ColabPro or Gradient) is required for the duration of the project. Contact the instructor if you think you need more computational resources than what you have available.

**Project 7: Overlap detector + speaker counter**

The objective of this project is to build a speaker counter for meeting recordings using speech technology. Meetings often have overlapping speech and speech separation technologies such as SepFormer (implemented in SpeechBrain) can separate the speech into individual tracks for each speaker. However, before speech separation can be applied, the segments of the recording that contain overlapping speech must be identified. This can be done with a neural network that inputs a short speech segment and outputs the number of speakers present in the segment. The output could be 0, 1, 2, or 3, representing no speakers, one speaker, two speakers, or three or more speakers, respectively.

The student is tasked with the following steps to achieve this goal:

1. Reviewing the literature on speaker counting.
2. Implementing a data simulator that creates overlapping speech signals by sampling clean data from a large dataset (e.g. [librispeech-clean-100](https://www.openslr.org/12/)) and adding noise and reverberation from the [open-rir dataset](http://www.openslr.org/28) with a specified probability (e.g. 0.5).
3. Implementing and testing at least two models for speaker counting, such as x-vectors or ECAPA-TDNN (or any other model proposed by the students), with input being a 1-2 second speech segment.
4. Implementing the inference stage where the model can process long recordings by chunking them into 1-2 second segments and making a decision on the number of speakers present in each segment. The output should be in the form of a text file indicating the start and end times of each segment and the decision made by the model. For instance:

0.00 1.00 0 (no speech)

1.00 5.50 1 (1 speaker)

5.50 6.55 2 (two speakers)

6.55 10.34 1 (1 speaker)

Where each line contains begin\_second, end\_second, classifier decision.

1. Integrating the code into the main SpeechBrain project and making the best model available on the SpeechBrain repository for use by others. Develop and interface for inference purposes, similar to the one available for the [VAD](https://huggingface.co/speechbrain/vad-crdnn-libriparty).