Chapter 7

Multilingual Conversational ASR–Challenge 2025

In this chapter, we describe our participation in the Multilingual Automatic Speech Recognition (ASR) competition organized by Nexdata.ai https://www.nexdata.ai/competition/mlc-slm.

For speech dialogue systems to work well, we need real-world conversation data. This data captures the natural flow of speech, including pauses, interruptions, when speakers talk over each other, and different styles of speaking. Such data is especially hard to get when dealing with multiple languages. This competition aim to help to solve this problem. The organizers provide a large, real-world dataset of conversations in multiple languages. The goal is to build speech language models (SLMs) that can work well in these multilingual conversational settings. These models can help build better AI systems that understand and respond naturally in real human conversations across different languages.

7.1 Task Description

This competition has two tasks. Both are about building speech language models that work with multiple languages. But we have only participated in the Task I.

Task I: Multilingual Conversational Speech Recognition Goal:

The goal of this task is to build a speech recognition system that can accurately transcribe conversations in multiple languages. The dataset gives clear information about where each person's speech starts and ends (oracle segmentation) and who is speaking (speaker labels). The main focus is to produce the most accurate transcription of these multilingual conversations. The performance of the system is evaluated using Word Error Rate (see Equation 4.1) or Character Error Rate (see Equation 4.2).

Dataset Description

The Training Set includes the 11 languages: English, French, German, Italian, Portuguese, Spanish, Japanese, Korean, Russian, Thai, and Vietnamese. Each recording has two people having a natural conversation about random topics. Recordings were made in quiet indoor settings. The dataset includes segmentation and speaker labels. The dataset contains about 1500 hours of multilingual conversational speech:

- 500 hours of English recordings, covering a variety of accents such as British, American, Australian, Indian, and Philippine English.
- Around 100 hours for each of the other languages in the dataset.

Language	Hours
English	500
Russian	100
Vietnamese	100
Korean	100
French	100
German	100
Italian	100
Portuguese	100
Spanish	100
Japanese	100
Thai	100

Figure 7.1: *Dataset Description*

The development set is similar to the training set but smaller, with about 4 hours of data per language. It is used for testing and fine-tuning models during development. The evaluation set is used for scoring model performance:

- Eval 1 (Task I) provides segmentation (where speech starts and ends) and speaker labels. Results are evaluated using Word Error Rate or Character Error Rate.
- For Japanese, Thai, and Korean, the output text (hypotheses) must have spaces between each character. For other languages, the output text format stays the same.

Now we will discuss about our approach which we have implemented to solve this challenge.

7.2 Approach 1: Model Architecture for Multilingual ASR Challenge

In this section, we explain our approach for the Multilingual ASR Challenge. We used a multilingual speech model [9] as the base model. This model is trained to give speech representations for 11 languages. On top of this base model, we built a custom decoder to suit our task.

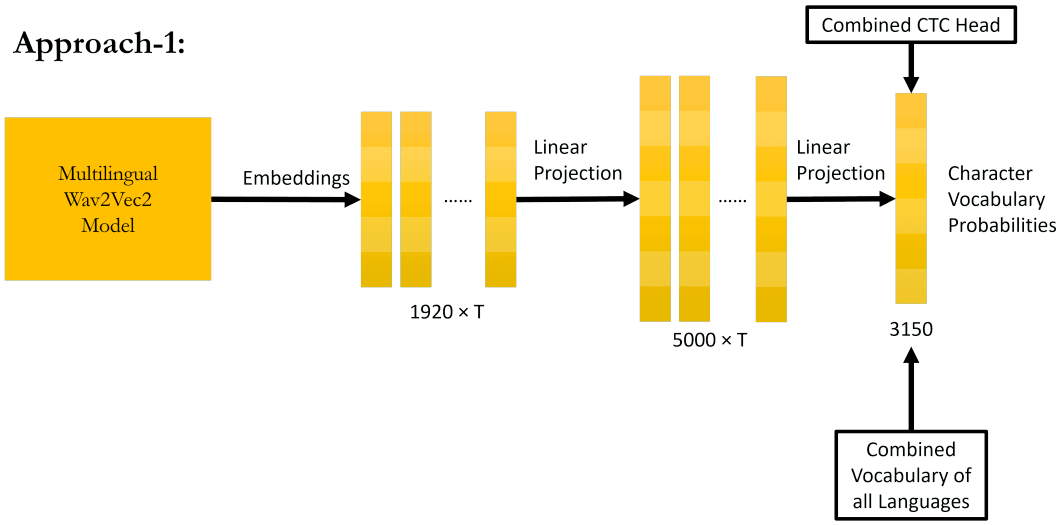

Figure 7.2: *Model Architecture For Approach 1*

Introduction to wav2vec2-XLS-R The wav2vec2-XLS-R model [9] is a large and powerful speech representation model developed by Facebook AI (now Meta AI). It belongs to the wav2vec 2.0 (for more details see the literature survey 3.1) family, which is designed to process raw audio and convert it into useful features for tasks like automatic speech recognition (ASR).

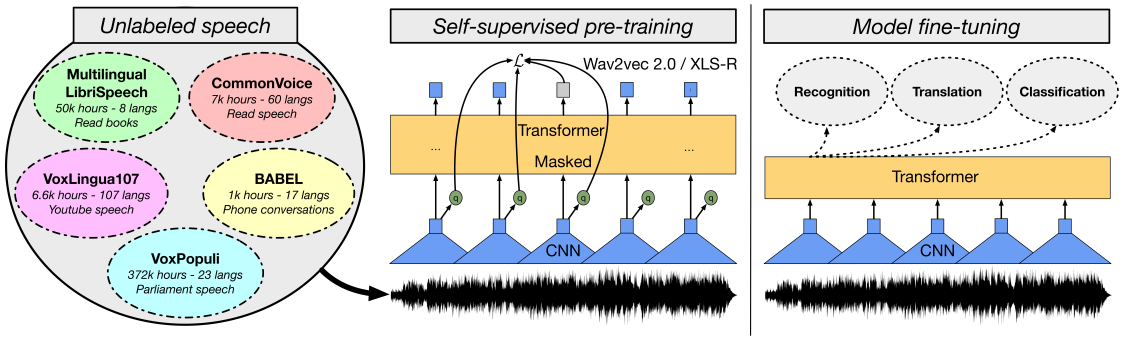

Figure 7.3: *Wav2Vec2-XLS-R Model*

The wav2vec2-XLS-R model is shown in the Figure 7.3 What makes XLS-R special is that

it is trained on speech data from many different languages (XLS-R stands for Cross-Lingual Speech Representation). This means the model learns patterns in speech that are common across languages. As a result, it works well for multilingual tasks, where the system has to understand and transcribe speech from different languages.

As shown in Figure 7.2, we first take the embeddings from the base model. These embeddings have a size of 1920 dimensions for each time step (each time step is about 25 milliseconds). We then project these embeddings to 5000 dimensions to help the model learn more complex patterns. After this, we add a CTC head — this is a simple linear projection layer that maps the 5000-dimensional vectors to a probability distribution over the combined vocabulary.

The model is trained using CTC Loss (Connectionist Temporal Classification Loss) [15]. For more details, see Appendix A.6. CTC Loss is commonly used for automatic speech recognition.

Combined Vocabulary

To train this model, we need to tokenize both the audio and its corresponding text. The audio tokenization is handled by the built-in tokenizer of the base model [9]. For text tokenization, we created our own vocabulary. Here's how we built the combined vocabulary:

- We combined all the text data from the different languages.
- We separated each character from this combined text and created a dictionary of unique characters along with their frequencies.
- We removed characters that appeared less than 10 times.

After this process, we ended up with the vocabulary of about 3150 tokens, which includes unique characters from all languages and some special tokens.

7.3 Data Management & Training Strategy

In this section, we explain how we managed the large data set for training this large model as we discussed in the previous Section 7.2. The total size of the training data set was around 300 GB, making it challenging to handle. If we tried to train the model on the entire dataset at once, it would have taken a very long time to complete even one epoch (one full pass through the data).

On top of this, we had limited GPU resources on the Param-Rudra cluster provided by the IIT Bombay, so we had to plan our training strategy carefully to make the best use of the available resources. To manage this, we decided to:

- Divide the large dataset into smaller parts (chunks of about 5,000 or 2500 samples each as per requirements). This made the data easier to load, process, and train on.
- We have ensure that each chunk have equal balance of classes — for example, a mix of different languages — so that the model could learn properly at each step.

- Train the model on these smaller chunks one after another, instead of the entire dataset at once. For example, we would train on samples between 145,000 and 147,000, then move to the next window.
- After training on each chunk, we saved a checkpoint of the model, so we could continue from where we left off in the next run without losing progress.
- Using this strategy, we were able to train efficiently despite the limited GPU access. On average, training on all the subsets for one epoch took about one day to complete.

Component	Details
GPU Node	Param-rudra cluster — each node with **2× A100 GPUs (80GB)**
Batch Size	2 or 4
Epochs	1 per subset (due to GPU quota)
Base Model	facebook/wav2vec2-xls-r-2b
Frozen Transformer Layers	0, 2, 5 First Layers to be Freeze
Learning Rate	9e-6, 5e-06, 1e-5
Vocabulary	Custom multilingual vocab (JSON, 3k+ tokens)
Loss Tracking	Saved per run to .csv (example: loss_145000_147000.csv)
Checkpointing	Incremental save/load via .pt files across runs

Figure 7.4: *Configuration For Training Model Using Approach 1*

The above Table 7.4 presents the key configuration used in our Approach 1 model configuration for the multilingual ASR challenge. It includes:

- Each GPU handled a batch size of 2 or 4 sample, given the large model and memory constraints.
- We have used different-different learning rate according to the number of froze layers in the model.
- Checkpoint is saved for the each chunk and used this checkpoint for the next chunk.

7.4 Loss & WER Curve For each Epoch

In this section, we discuss the loss plots and Word Error Rate plots for each epoch. Since our model has over 2 billion parameters, we trained it using different configurations in each epoch to manage the complexity. Let's start with Epoch 1. As shown in Figure 7.5, the plots display the loss versus training steps and WER versus training steps. In this epoch, the first 5 transformer layers of the model were frozen. The model achieved an average WER of 0.8301 and an average loss of 2.8448.

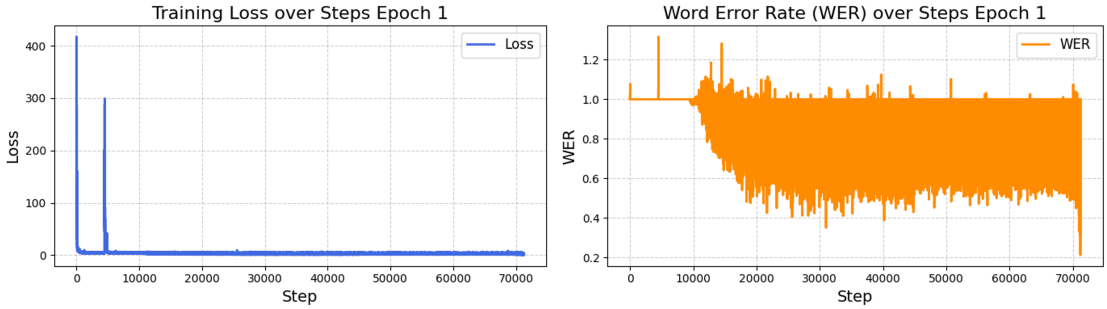

Figure 7.5: *Training Loss and WER with First 5 Transformer Layers Frozen, Epoch 1*

Next, in Epoch 2 (see Figure 7.6), we again kept the first 5 layers frozen. This time, the model showed better results with an average loss of 1.4219 and an average WER of 0.5717.

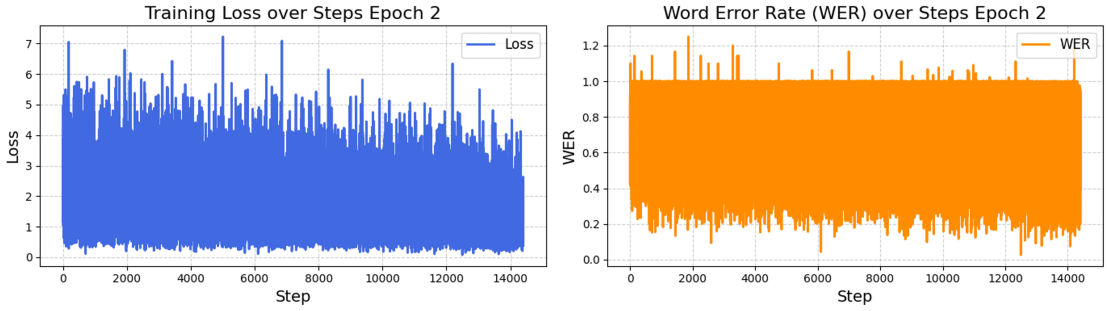

Figure 7.6: *Training Loss and WER with First 5 Transformer Layers Frozen, Epoch 2*

Next, in Epoch 3 (see Figure 7.7), Now we kept the first 2 layers frozen. This time, the model showed better results with an average loss of 1.2328 and an average WER of 0.5335.

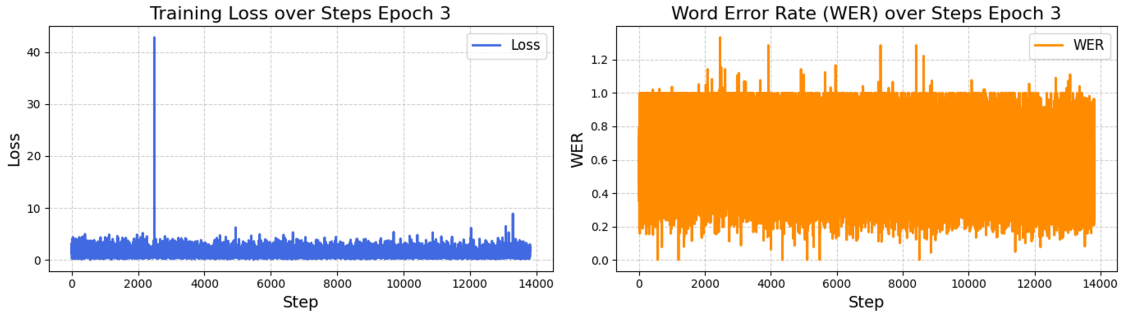

Figure 7.7: *Training Loss and WER with First 2 Transformer Layers Frozen, Epoch 3*

Next, let's look at Epoch 4 (see Figure 7.8). In this epoch, all layers of the model is set to be trainable, unlike the earlier epochs where some layers were frozen. The plot shows results only up to 4000 training steps, which means that the training for this epoch was not completed. This is because making all layers trainable significantly increased the time needed to finish this epoch.

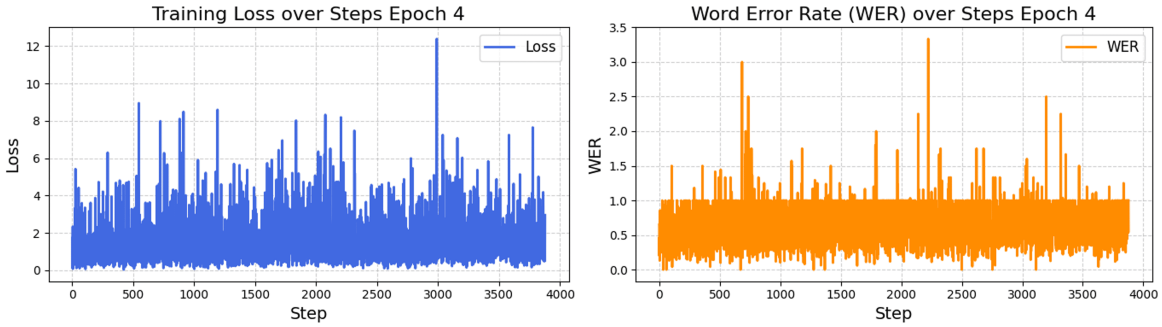

Figure 7.8: *Training Loss and Word Error Rate with Fully Trainable Transformer Layers*

To manage the large size of the model (which has over 2 billion parameters), we also changed how we handled numerical precision. At first, we were training the model using float32 precision, but this required too much GPU memory and we could not fit the entire model into the 80 GB GPU. So, we switched to float16 precision, which is known as mixed precision training. This helped reduce the GPU memory usage and made it possible to continue training without running out of memory.

In Epoch 4, the model achieved an average loss of 1.5440 and an average WER of 0.6350. However, since the training was incomplete, these results do not fully reflect the potential performance of the model in this configuration. Next, we will discuss the performance of these various trained models on the development data set .

7.5 Approach 1: Performance of Model on the Development Dataset

The Word Error Rate Plot, as shown in Figure 7.9, presents WER for each language across different model configurations (Prediction 0, Prediction 2, and Prediction 5). WER measures how many words were predicted incorrectly. Lower WER values mean better results.

- Prediction 0 (Blue Bar) refers to the model where all layers are trainable (no layers are frozen).
- Prediction 2 (Orange Bar) refers to the model where the first 2 layers are frozen during training.
- Prediction 5 (Green Bar) refers to the model where the first 5 layers are frozen.

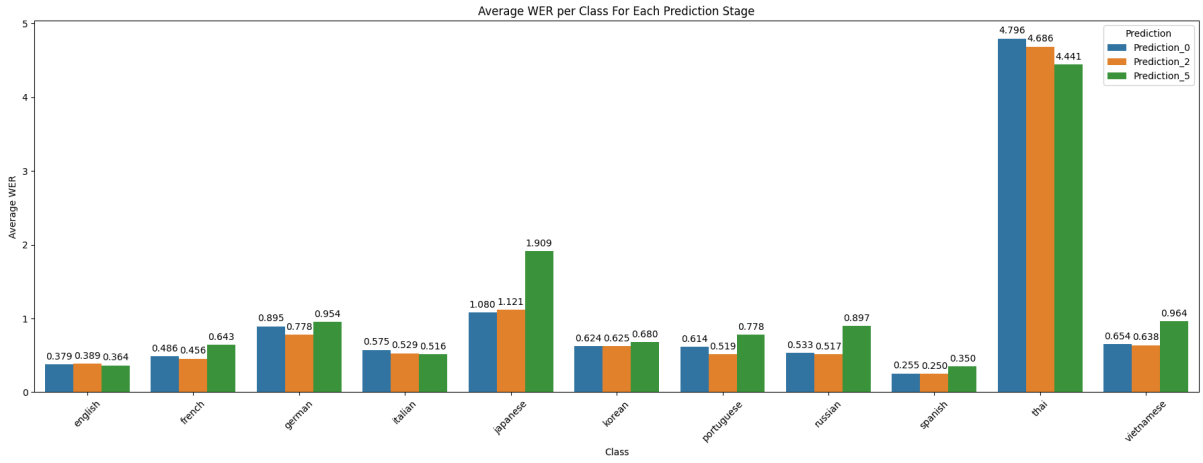

Figure 7.9: *Word Error Rate (WER) Across All Models For Each Language*

1. English: The WER stayed quite stable across predictions — around 0.38 at Prediction 0, 0.39 at Prediction 2, and 0.36 at Prediction 5.
2. French: The WER started at 0.49, improved a little at Prediction 2 (0.46), but got worse at Prediction 5 (0.64).
3. German: Similar pattern — better at Prediction 2 (0.78) than Prediction 0 (0.90), but higher again at Prediction 5 (0.95).
4. Japanese: WER increased over predictions — 1.08 at Prediction 0, 1.12 at Prediction 2, and 1.91 at Prediction 5, showing the model struggled more at later stages.
5. Thai: The WER was very high across all predictions — 4.80 at Prediction 0, 4.69 at Prediction 2, and 4.44 at Prediction 5, so Thai remained the most challenging language for the model.

6. Other languages like Korean, Spanish, Italian, Russian, etc., had smaller changes across prediction stages, with some small improvements or slight increases in WER.

The Character Error Rate (CER) Plot, as shown in Figure 7.10, presents the CER for each language across different model configurations (Prediction 0, Prediction 2, and Prediction 5).

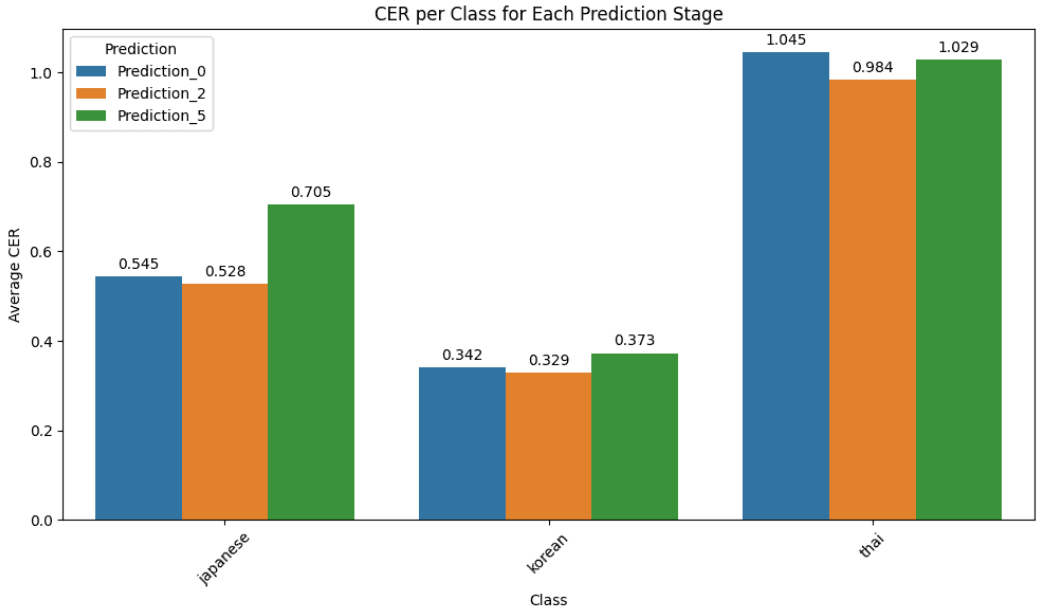

Figure 7.10: *Character Error Rate (CER) Across All Models For Each Language*

CER measures how many characters in the predicted text are incorrect compared to the actual text. A lower CER indicates better model performance.

1. Japanese: The CER was around 0.54 at Prediction 0, improved slightly to 0.53 at Prediction 2, but increased to 0.71 at Prediction 5. This shows that the model's performance varied depending on how many layers were frozen.

2. Korean: The CER remained low across all stages — about 0.34 at Prediction 0, 0.33 at Prediction 2, and slightly higher at 0.37 at Prediction 5.

3. Thai: The CER stayed high at all stages — 1.05 at Prediction 0, 0.98 at Prediction 2, and 1.03 at Prediction 5, indicating that the model consistently found Thai challenging across configurations.

7.6 Approach 2: Model Architecture for Multilingual ASR Challenge

In this approach, we aimed to build a better model compared to our previous setup. To do this, we introduced a multi-head CTC architecture, where we created separate CTC heads for each language on top of a shared base model 7.2 same as we have used in the Approach 1.

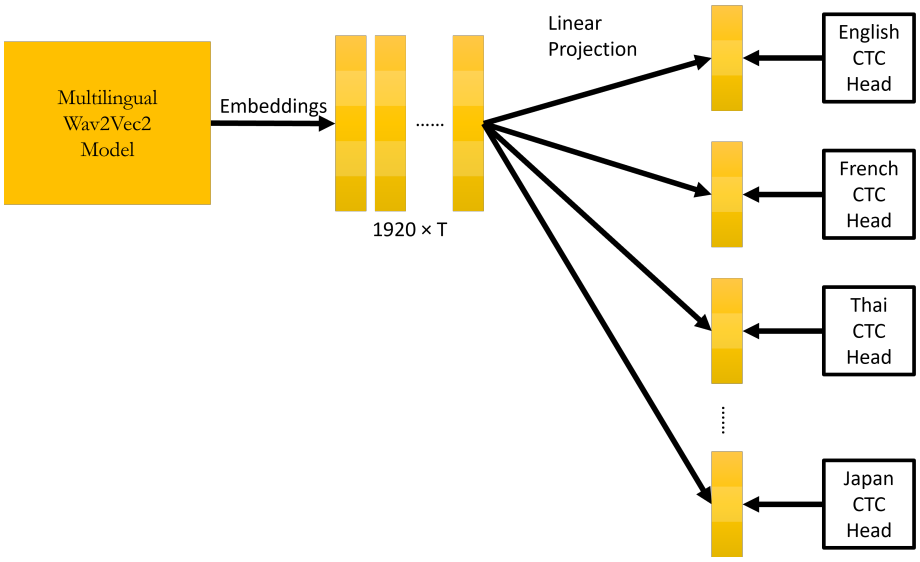

Figure 7.11: *Complete Data Transformation Pipeline*

As shown in Figure 7.11, the model consists of:

- A common base model that processes the input speech and generates shared speech embeddings for all languages.
- On top of this base model, we added multiple linear projection layers (CTC heads) — one for each language.
- Each CTC head is a linear layer that maps the shared embeddings produced by the base model to the vocabulary of its specific language. This means language-specific CTC head converts these features into predictions such as characters for that particular language.
- During training, depending on the language of the input in the batch, only the corresponding CTC head is activated.

In this model, each CTC head generates a probability distribution over the vocabulary of its corresponding language. This means that each head has its own vocabulary that matches the language it is responsible for. Now we will see how we make language ID dictionary and tokenizer dictionary.

Language ID and Tokenizer Mapping for Multi-Head CTC Model

For the model to choose the correct CTC head during training and prediction, it is very important that the data class prepares the batches properly. Each batch must contain:

- The audio features (input to the model).
- The language ID (to help the model select the right CTC head).

To make this work correctly, it's not enough to just pass the audio. We also need to ensure that the right tokenizer is used for each language. The tokenizer converts text into numerical form that the model understands (such as converting characters into IDs).

To manage this, we created two important dictionaries:

1. **lang2id dictionary** — This maps each language to a unique ID (or index). For example, English might be ID 0, French might be ID 1, and so on. This ID is used by the model to select the correct CTC head during training and evaluation. The mapping is shown in Figure 7.12.
2. **tokenizer dictionary** — This maps each language to its specific tokenizer. The tokenizer is responsible for converting the output text to the format expected for that language. As shown in Figure 7.13, each language has its own tokenizer entry. The order and keys of this dictionary match exactly with the lang2id dictionary, ensuring that the model can correctly pair each language with its tokenizer.

```
lang2id = {
            "english" :  0,
            "korean" :   1,
            "japanese" : 2,
            "italian" :  3,
            "russian" :  4,
            "portuguese":5,
            "spanish" :  6,
            "vietnamese":7,
            "french" :   8,
            "thai" :     9,
            "german" :   10
            }
```

Figure 7.12: *Language to ID Dictionary Map*

Both of these dictionaries are essential for the data processing pipeline. They ensure that the correct language ID is passed along with the audio features. The correct tokenizer is used to process the labels (target text) for each language. The model can dynamically select the right CTC head during training and inference. This design ensures smooth training and evaluation of the multi-head CTC model on multilingual data.

```
tokenizers = {
                "english": eng_tokenizer,    #0
                "korean": kor_tokenizer,     #1
                "japanese": jap_tokenizer,   #2
                "italian": ita_tokenizer,    #3
                "russian": rus_tokenizer,    #4
                "portuguese": por_tokenizer,#5
                "spanish": spa_tokenizer,    #6
                "vietnamese": vie_tokenizer,#7
                "french": fre_tokenizer,     #8
                "thai": tha_tokenizer,       #9
                "german": ger_tokenizer     #10
            }
```

Figure 7.13: *Language ID to Tokenizer Dictionary Map*

Multi-Head CTC Forward-Backward Logic

In our model, we use a multi-head CTC (Connectionist Temporal Classification) setup where we have separate CTC heads for different languages. This means that each language has its own CTC layer for making predictions.

During the forward pass (when the model is making predictions and calculating the loss):

1. Each batch of data contains the input features (audio features) along with the language ID for each sample.
2. The model checks the language ID for each input in the batch.
3. Based on this language ID, the model selects the correct CTC head for that language.
4. Only the selected CTC head is used to compute the loss for that batch. The other CTC heads are ignored during this step.

During the backward pass (when the model updates its weights based on the loss):

1. The gradients (which are used to update the model parameters) flow only through the selected CTC head that was active during the forward pass.
2. The other CTC heads do not receive any gradient updates in that batch. This ensures that only the relevant part of the model is updated for each language.
3. This approach allows the model to train language-specific CTC heads effectively, while sharing the same base model for all languages.

7.7 Loss & Word Error Rate For Approach 2

We recorded the loss and WER (Word Error Rate) for each language in the batch, as well as the average WER for the batch, and saved them in a text file. A snapshot of this data is shown in Figure 7.14, which displays the loss and WER values up to epoch 1. From the results, we noticed that while the loss decreased quite well during training, the average WER and the WER for individual languages did not improve much they stayed almost the same that is around 1.

In this snapshot: wer-lang-0, wer-lang-1, ..., wer-lang-9 show the WER values for each specific language. Each number corresponds to a particular language in the model (for example, lang-0 might be English, lang-1 could be Japanese, etc.). There is empty spots (just commas without numbers), it means that language was not present in the batch at that step.

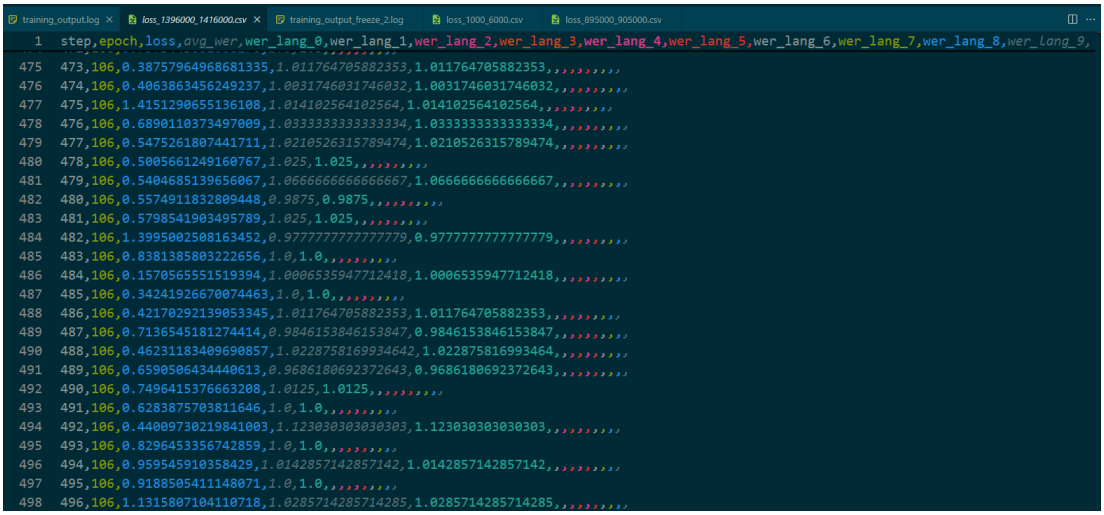

Figure 7.14: *Loss & Word Error Rate (WER) for Approach 2*

Overall, this approach with separate CTC heads for each language did not give good results. Our previous method, where we used a single combined CTC head for all languages, worked better and gave lower WER.

7.8 Leaderboard Results

We made submissions on the evaluation set provided by the challenge organizers using different models:

- When we submitted results from the model trained till epoch 2 with the first 5 layers frozen, we got an average WER of 0.53 on the evaluation set.
- When we used the model where the first 2 layers were frozen, the average WER was 0.39.
- For the model where no layers were frozen (0 layers frozen), the average WER was 0.41. Since this model hadn't completed its full training (all epochs), we believe it could achieve even better results if training continued.

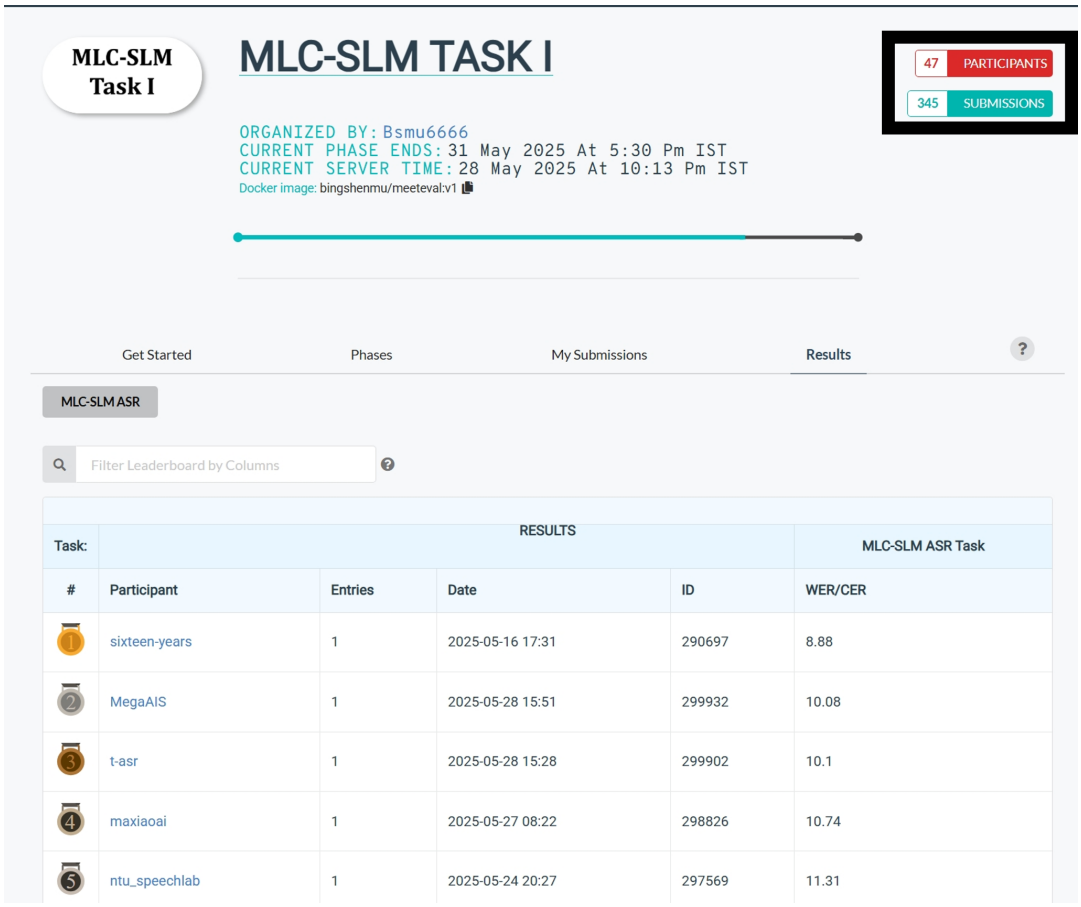

Figure 7.15: *Leader Board Result*

As shown in the leaderboard, there were a total of 47 participating teams. Our team achieved a rank of 23rd with a score (WER) of 0.39 and our team name is Speech-ABS (see Figure 7.16).

6	ntu_speechlab	1	2025-05-29 12:01	300477	10.84
7	llm-t	1	2025-05-30 08:12	300958	11.27
8	seewo	1	2025-05-30 14:03	301135	11.57
9	maybe	1	2025-05-30 14:52	301174	11.76
10	daominhtri	1	2025-05-20 08:32	293280	11.94
11	fosafer	1	2025-05-28 13:52	299831	12.26
12	zgzmeng	1	2025-05-24 13:49	297420	12.87
13	eloquence	1	2025-05-18 20:05	292446	15.15
14	High Accuracy	1	2025-05-26 19:02	298598	16.63
15	reopenai	1	2025-05-20 10:00	293324	17.45
16	but_mlc	1	2025-05-22 20:57	296517	19.69
17	mlcslm-baseline	1	2025-05-16 17:31	290698	20.17
18	voicecode	1	2025-05-27 11:41	298904	20.82
19	sigma	1	2025-05-21 11:28	295546	27.84
20	auroralab	1	2025-05-20 14:57	293520	28.55
21	pachira	1	2025-05-23 12:27	296843	31.07
22	funspeech	1	2025-05-27 11:35	298896	33.56
23	Speech-ABS	1	2025-05-29 15:52	300560	39.06

Figure 7.16: *Leader Board Result*

Since we wanted to try a new approach that didn't exist before, we chose not to use the standard ASR models that are already available. If we had used those existing models directly for all languages, we could have achieved better scores more easily.

The main difference between Approach 1 and Approach 2 is:

- In Approach 1, the model can transcribe speech to text without needing any language ID information.
- In Approach 2, the model requires the language ID as input in order to transcribe the speech.