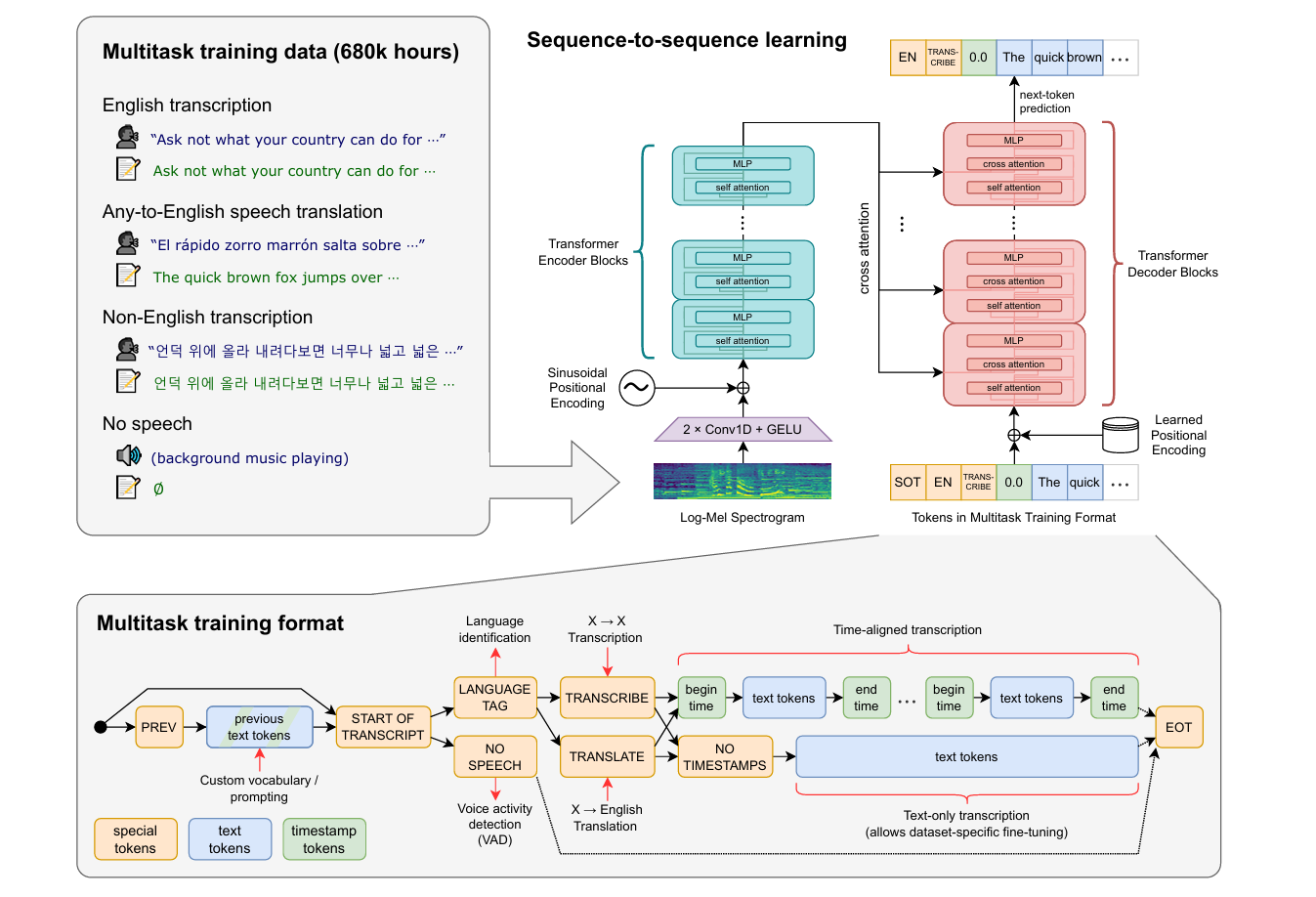
**WHISPER:**

Whisper is a powerful general-purpose speech recognition model developed by OpenAI and released in 2022. Whisper was trained on a large and diverse set of multilingual and multitask data collected from the web. Moreover, Whisper supports speech-to-text tasks in nearly 100 languages and can also perform speech translation into English, making it one of the most versatile open-source ASR systems available.

Whisper is based on a sequence-to-sequence Transformer architecture. The model comprises two main components: an encoder and a decoder. The encoder is responsible for processing the audio input, which is first converted into a sequence of log-Mel spectrogram features. These features are passed through multiple layers of Transformer blocks consisting of multi-head self-attention and feedforward layers. The encoder outputs a high-dimensional representation of the audio input, capturing both local and global acoustic patterns. The decoder, on the other hand, is autoregressive and generates the transcription (or translation) token by token. It uses causal (left-to-right) self-attention to condition on previously generated tokens and cross-attention to incorporate information from the encoder outputs. The decoder operates over a fixed vocabulary of around 50,000 tokens that include characters, timestamps, language identifiers, and special markers like "start of transcript".

Before being fed into the model, audio undergoes a carefully designed preprocessing pipeline. The audio files are resampled to 16 kHz and transformed into an 80-channel log-Mel spectrogram on overlapping windows (25 ms in length with a 10 ms stride). Whisper processes audio in 30-second chunks, which allows the model to handle long-form audio recordings in a computationally efficient way.

Whisper was trained using 680,000 hours of supervised audio-transcription pairs, making it one of the largest ASR training sets ever constructed. Of this, approximately 117,000 hours are in English, while the remaining 563,000 hours consist of audio in other languages paired with their English translations. This not only teaches the model to transcribe but also to translate speech to English. The model was trained using a cross-entropy loss objective to maximize the likelihood of correct token sequences.

****

**Figure 1: Overview of the Whisper model architecture**



|  |  |
| --- | --- |
| **Table 1: Whisper model sizes** | **Table 2: WER on common voice 9 dataset** |

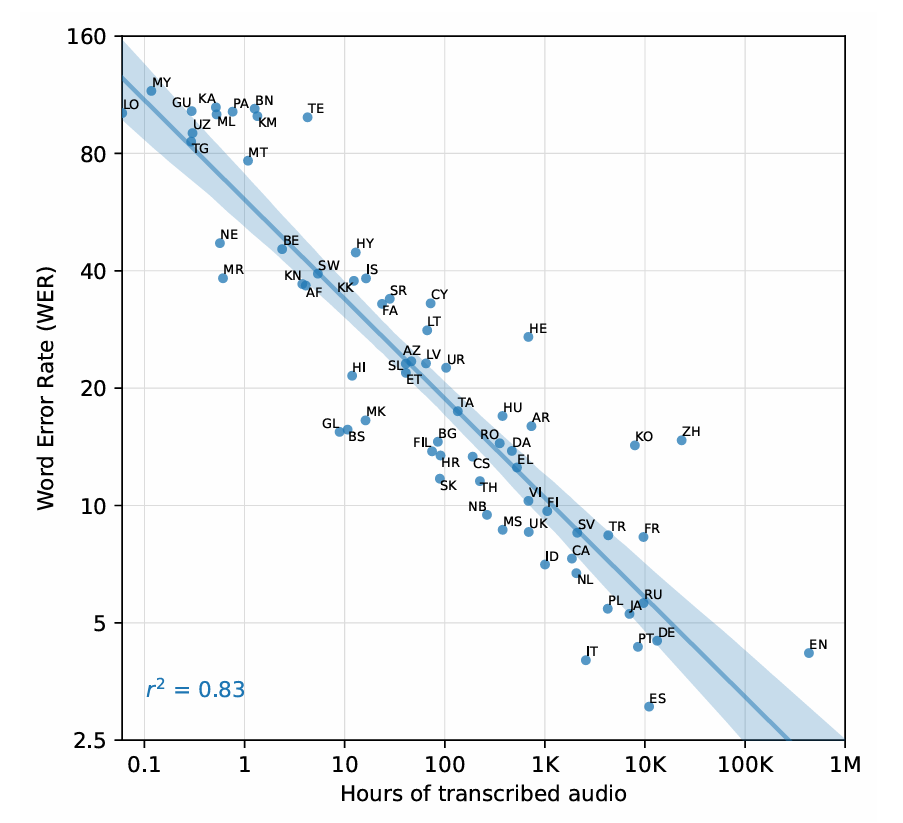
****



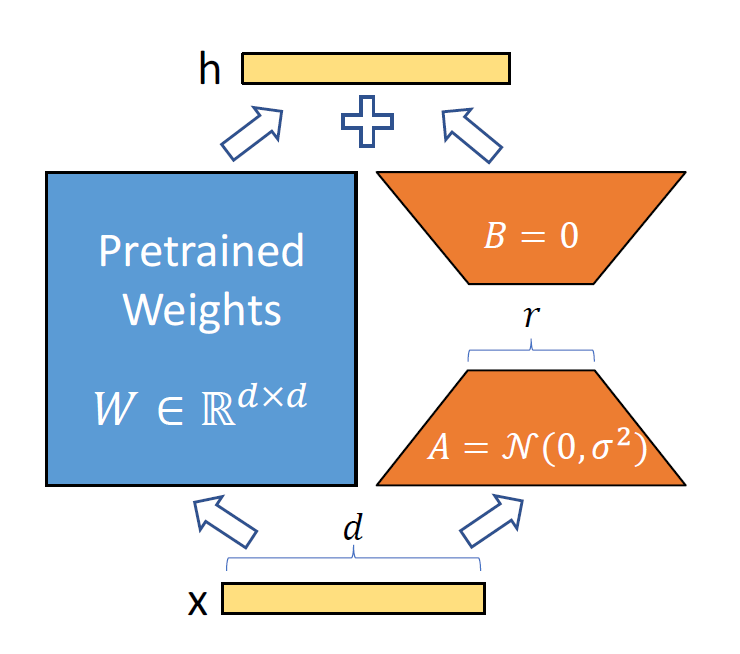
Figure 2: Whisper trained on 12 hours of Hindi audio

**What is PEFT?**

[1] Full fine-tuning of transformer-based Pretrained Language Models (PLMs) involves training the entire model, including all layers and parameters, on a specific downstream task using task-specific data. Initially, PLMs are trained on large-scale datasets with unsupervised learning objectives like language modeling or masked language modeling, to learn general language representations. However, these PLMs may not perform optimally when applied to specific tasks due to a lack of appropriate domain knowledge or lesser application specific training. Full fine-tuning provides an effective solution to address this limitation. During full fine-tuning, the PLM is initialized with pretrained weights and subsequently trained on task-specific data using techniques like backpropagation and gradient descent. All model parameters, including pretrained weights, are updated to minimize a task-specific loss that quantifies the disparity between predicted outputs and ground truth. In this way, full fine-tuning enables the model to learn task specific patterns and nuances from the labelled data, facilitating predictions or outputs tailored to the target tasks. Notably, full fine-tuning necessitates substantial computational resources and labelled data, as the model is trained from scratch for the specific target task. Moreover, as PLMs grow in size and with the advent of LLMs containing billions of parameters, full fine-tuning places even greater demands on computational resources. In contrast, PEFT methods aim to alleviate these requirements by selectively updating or modifying specific parts of the PLMs while still achieving performance comparable to full fine-tuning. Furthermore, full fine-tuning may give rise to overfitting when the task-specific dataset is small or when the PLMs are already well-suited to the target task.

Parameter-Efficient Fine-Tuning (PEFT) [1] helps mitigate the problem of catastrophic forgetting by preserving the pre-trained model’s original knowledge while enabling task-specific learning. In traditional fine-tuning, updating all the model's parameters for a new task often leads to overwriting the weights important for previous tasks, causing the model to forget earlier knowledge—a phenomenon known as catastrophic forgetting. PEFT addresses this by freezing the base model and introducing a small number of trainable parameters, such as low-rank adapters or prompt embeddings, specific to each task. This separation ensures that learning new tasks does not interfere with previously learned ones, making PEFT particularly effective in fine-tuning tasks. Thus, PEFT is a class of techniques designed to adapt large pre-trained models to downstream tasks without updating all of their parameters. Also, traditional fine-tuning methods require significant computational resources and memory, as they involve adjusting the entire model. In contrast, PEFT methods introduce a small number of additional parameters—such as low-rank adapters, prompts, or side modules—while keeping the vast majority of the model frozen. This approach greatly reduces training costs, enables fast adaptation to new tasks, and allows for efficient deployment, especially in low-resource settings. Techniques like LoRA (Low-Rank Adaptation), Prefix Tuning, and Adapters exemplify the PEFT paradigm and have demonstrated competitive or superior performance compared to full fine-tuning in many NLP applications.

**LoRA (Low Rank Adaptation):**



***Figure : Low-Rank Adaptation (LoRA) of Pretrained Weights via Randomized Parameter Injection [2]***

Low-Rank Adaptation (LoRA) [2] is an efficient fine-tuning technique designed to adapt large pretrained models using significantly fewer parameters. Instead of updating all the weights of a model, LoRA introduces low-rank trainable matrices into existing layers while keeping the original pretrained weights frozen. This is achieved by decomposing the weight update into two smaller matrices — one initialized from a normal distribution and the other often set to zero — and applying their product as a low-rank update to the original weights. By doing so, LoRA dramatically reduces memory and computational requirements during training, making it highly suitable for fine-tuning large models like Whisper on domain-specific tasks without needing to retrain or store the full model.

A neural network contains many dense layers which perform matrix multiplication. The weight matrices in these layers typically have full-rank. When adapting to a specific task, the pre-trained language models have a low “intrinsic dimension” and can still learn efficiently despite a random projection to a smaller subspace. Inspired by this, it is hypothesized that the updates to the weights also have a low “intrinsic rank” during adaptation.

Instead of updating the large pre-trained weights W ∈ ℝ^{d×d}, LoRA **freezes W** and introduces **two low-rank trainable matrices**, A and B, such that:

ΔW=BA

So, during training, only A and B are updated while W remains fixed.

* **Pretrained Weights:**
  + W ∈ ℝ^{d×d} is the original weight matrix of a large model
  + It is not trained during LoRA fine-tuning — it is frozen.
* **Input and Output:**
  + x: Input vector to the layer of size d.
  + h: Output after processing via LoRA (added on top of the normal output).
* **Trainable Low-Rank Matrices:**
  + A ∈ ℝ^{r×d}: Randomly initialized (normal distribution), trainable matrix.
  + B ∈ ℝ^{d×r}: Initialized as zero, also trainable.
  + These two represent a low-rank decomposition with rank r (r << d).
* **Computation Path:**
  + Input x is passed through A → produces a lower-dimensional representation (r).
  + That is then passed through B → maps it back to the original dimension (d).
  + This output BAx is added to the output from the frozen W (i.e., Wx) as a residual connection:

h=Wx+BAx

Thus, LoRA (Low-Rank Adaptation) is particularly well-suited for fine-tuning large-scale models like Whisper-large (with 1.55 billion parameters) to improve performance on application-specific tasks such as transcription in a particular language or dialect. While Whisper-large is trained on a diverse, multilingual dataset, its generalized training may not fully capture the nuances, pronunciation patterns, or linguistic variations of a specific language—especially if it is underrepresented in the pretraining corpus. By fine-tuning with LoRA, the model can learn these language-specific patterns more effectively. LoRA introduces additional trainable low-rank matrices into select layers while keeping the pretrained weights fixed, allowing the model to adapt to new linguistic contexts without forgetting its foundational knowledge. This targeted adaptation not only enhances transcription accuracy in the target language but also preserves the robustness and fluency inherited from the large-scale pretraining. Thus, LoRA serves as a powerful mechanism to specialize Whisper-large for high-performance, language-specific speech recognition.

**Experimental setup:**

We are using common voice 21 (Hindi dataset) for finetuning using LoRA.

Train set: 9.38 hours

Test set: 4.73 hours

Train samples: 7563

Test samples: 3337

**WER before finetuning:**

WER on Train Set: 68.92%

WER on Test Set: 71.67%

**Steps in training:**

Training samples = 7563

Batch size = 8

Steps=7563/8=946

**Epochs**: 20

For 20 epochs, 20\*946= 18920 steps.

**Trainable parameters:**

trainable params: 15,728,640 || all params: 1,559,033,600 || trainable%: 1.0089

* Trainable params:15,728,640  
  These are the parameters being updated during training—introduced by LoRA (e.g., low-rank adapters in attention layers).
* All params: 1,559,033,600  
  This is the total number of parameters in the model, including both the frozen base model and any fine-tuned layers.
* Trainable %: 1.0089%  
  This shows that only ~1% of the model's total parameters are being fine-tuned, thanks to LoRA. The rest remain frozen.

**LORA Config:**

r=32: sets the **rank** of the LoRA adapter matrices A and B.

lora\_alpha=64: scaling factor for LoRA output.

target\_modules=["q\_proj", "v\_proj"]: injecting LoRA **only into specific submodules** of the model (likely self-attention components).  
bias="none": don't adapt bias terms.

lora\_dropout=0.05: adds dropout during adapter training.

**References:**

1. Xu, Lingling, et al. "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment." *arXiv preprint arXiv:2312.12148* (2023).
2. Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *ICLR* 1.2 (2022): 3.
3. Radford, Alec, et al. "Robust speech recognition via large-scale weak supervision." *International conference on machine learning*. PMLR, 2023.