Industrial Training Report on ASR-FairBench: Measuring and Benchmarking Equity Across Speech Recognition Systems

Submitted by:

Satyam Rahangdale

Roll Number: 22CH10062

Department of Chemical Engineering

Indian Institute of Technology, Kharagpur

Internship Role:

Full Stack Developer

Project Title : ASR-FairBench

Supervisor: Prof. Animesh Mukherjee, Professor, Department of CSE, IIT Kharagpur

Organization:

Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur

Duration of Training:

May 3, 2025, to June 28, 2025

Acknowledgement

I would like to express my profound gratitude to my supervisor, **Prof. Animesh Mukherjee**, Professor in the Department of Computer Science and Engineering at IIT Kharagpur, for his invaluable guidance, unwavering support, and insightful mentorship throughout my industrial training. His expertise and vision were the driving forces behind the ASR-FairBench project, and his encouragement was instrumental in my professional and personal growth during this period.

My sincere thanks go to my project collaborators. I am especially grateful to **Anand Rai**, Research Scholar in the Department of CSE, for his direct mentorship and technical guidance. His deep knowledge of the subject matter and willingness to share it were crucial for navigating the complexities of the project. I also extend my appreciation to **Utkarsh Anand**, a fellow undergraduate student, for his contribution in writing python code for ASR metrics calculation, for his partnership and collaborative spirit. Our discussions and joint problem-solving sessions were both productive and enjoyable.

I am also deeply thankful to the **Complex Network Research Group (CNeRG)** at IIT Kharagpur for providing a stimulating and collaborative research environment. The intellectual rigor and spirit of inquiry within the group were incredibly motivating.

This internship was a transformative experience, and it would not have been possible without the collective support of this exceptional team.

Table of Contents

1. Introduction

1.1. The Proliferation of Automatic Speech Recognition (ASR)

In the contemporary technological landscape, Automatic Speech Recognition (ASR) systems have transitioned from a niche research area to a ubiquitous and transformative technology. These systems have fundamentally revolutionized human-computer interaction, powering a vast array of applications that are now deeply integrated into daily life.[1] From the convenience of voice assistants like Alexa and Siri to the critical functionality of real-time transcription services for meetings and media, ASR technology has become an indispensable tool.[1] Its applications span across diverse domains, including messenger apps, interactive chatbots, and in-car command systems, making digital services more accessible and efficient for millions of users worldwide.[1] The ability to convert spoken language into machine-readable text with increasing accuracy has unlocked new paradigms of interaction and information access.

1.2. The Problem of Algorithmic Bias in ASR Systems

Despite the remarkable advancements in overall ASR accuracy, a significant and pressing challenge has emerged: the issue of algorithmic bias and fairness. State-of-the-art ASR systems often exhibit significant performance disparities across diverse demographic groups.[1] Factors such as a speaker's accent, gender, age, or linguistic background can lead to substantially different levels of accuracy, raising serious concerns about equity and inclusivity in technology.[1]

The root of this problem often lies in the evaluation methodologies and the data used for training and benchmarking. Traditional ASR leaderboards and evaluation frameworks have historically focused almost exclusively on aggregate accuracy metrics, with Word Error Rate (WER) being the predominant standard.[1] This narrow focus on a single performance number masks the underlying inequities. Many widely-used ASR models are benchmarked on public datasets that lack sufficient demographic diversity, which inadvertently perpetuates and amplifies biases against underrepresented populations.[1] Consequently, an ASR model

optimized solely for a low aggregate WER might perform exceptionally well for a majority demographic but fail significantly for speakers with varied accents or dialects, thereby creating a digital divide.[1]

1.3. Project Objective: Introducing ASR-FairBench

To address this critical gap in ASR evaluation, my industrial training project was centered on the co-development of **ASR-FairBench**. The primary objective of this project was to design, develop, and deploy an open, real-time benchmarking platform dedicated to evaluating ASR models based on a holistic framework that encompasses both accuracy and fairness.[1]

ASR-FairBench moves beyond simplistic accuracy metrics by leveraging Meta's Fair-Speech dataset, a rich public corpus containing utterances from participants with self-reported demographic information.[1] A core innovation of the project is the proposal and implementation of a novel metric, the

Fairness-Adjusted ASR Score (FAAS). This metric integrates the traditional WER with a statistically robust fairness score derived from a mixed-effects regression model, providing a single, comprehensive score that reflects a model's true performance across diverse user groups.[1] The platform's ultimate goal is to drive the development of more inclusive and responsible ASR technologies by providing researchers and developers with the tools to identify, measure, and mitigate biases in their models.

1.4. My Role and Contributions as a Full Stack Developer

My designated role in this project was that of a **Full Stack Developer**.[1] My responsibilities were comprehensive, covering the end-to-end lifecycle of the ASR-FairBench web platform, from initial design to final deployment. I was tasked with translating the complex research concepts and statistical models conceived by the team into a tangible, functional, and user-friendly web application.

Specifically, my contributions included:

- **Frontend Development:** I designed and implemented the entire client-side application using the **React.js** framework. This involved creating an interactive user interface for model submission, dynamic and insightful data visualizations for the audit results using libraries like Plotly.js, and a real-time leaderboard for model comparison.[1]

- **Backend Development:** I developed the server-side logic and RESTful API using **Python** and the **Flask** micro-framework. This backend handles user requests, orchestrates the ASR model inference process, executes the complex FAAS calculation, and manages the data for the leaderboard.[1]
- **Deployment and MLOps:** I was responsible for deploying the full-stack application as a containerized service on **Hugging Face Spaces**. This involved writing a Dockerfile for the Flask backend to ensure a reproducible environment and configuring the deployment pipeline for both the frontend and backend components.[1]

My work was not simply to implement a pre-defined specification but to build the very instrument through which our novel research on the FAAS metric could be tested, validated, and shared with the global research community. The ASR-FairBench platform itself stands as a primary research artifact, and my engineering efforts were central to bringing this research to life. This unique position at the intersection of academic research and practical software engineering defined my internship experience.

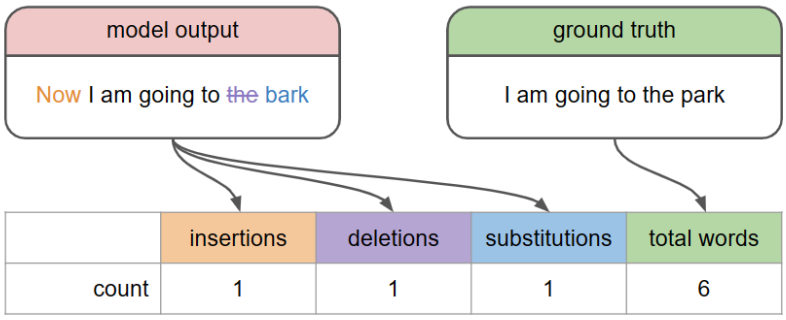
2. ASR Performance and Fairness Metrics

A core component of my work on ASR-FairBench involved a deep engagement with the metrics used to evaluate ASR systems. This required not only understanding traditional measures of performance but also implementing the novel fairness framework that defines our platform.

2.1. Traditional Accuracy Metrics: A Deep Dive

Before a more equitable evaluation framework can be appreciated, it is essential to understand the industry-standard metrics that have historically dominated the field.

2.1.1. Word Error Rate (WER): Definition and Calculation



$$\text{word error rate} = \frac{1 + 1 + 1}{6} = 0.5 = 50\%$$

Word Error Rate (WER) is the most common and widely accepted metric for measuring the performance of an ASR system.[2] It provides a quantitative measure of the errors present in a machine-generated transcript by comparing it against a ground-truth, human-verified reference transcript. The calculation is based on the Levenshtein distance, which determines the minimum number of edits required to change one sequence into another, but applied at the word level instead of the character level.[5]

The formula for WER is given by:

$$WER = \frac{S + D + I}{N}$$

where:

- **S** is the number of **Substitutions**, where a word in the reference is incorrectly replaced by another word in the hypothesis (e.g., reference "there" becomes hypothesis "bear").[2]
- **D** is the number of **Deletions**, where a word present in the reference is omitted from the hypothesis (e.g., reference "the black cat" becomes hypothesis "the cat").[2]
- **I** is the number of **Insertions**, where a word not present in the reference is added to the hypothesis (e.g., reference "the cat" becomes hypothesis "the black cat").[2]
- **N** is the total number of words in the reference transcript.[1]

A lower WER value signifies a more accurate ASR system, with a WER of 0% indicating a perfect transcription.[2] While indispensable for gauging overall accuracy, WER has significant limitations. It treats all errors equally, failing to distinguish between a minor error that doesn't change the meaning and a critical one that does.[5] Most importantly for our project, an aggregate WER score provides no insight into how those errors are distributed across different demographic groups, thus failing to capture systemic biases.[1]

2.1.2. Real-Time Factor (RTF): Measuring Computational Efficiency

While WER measures accuracy, the **Real-Time Factor (RTF)** measures the computational speed and efficiency of an ASR system. It is a critical metric for assessing the practical viability of a model, particularly for applications requiring live or interactive transcription.

RTF is calculated as the ratio of the time taken to process an audio file to the duration of the audio file itself:

$$RTF = \frac{Processing\ Time}{Audio\ Duration}$$

For example, if an ASR system takes 30 seconds to transcribe a 1-minute audio clip, its RTF is 0.5. An RTF value less than 1.0 is essential for any real-time application, as it indicates that the system can process the audio faster than it is being spoken, thus keeping up with the live input. An RTF greater than 1.0 means the system lags behind the speaker, leading to unacceptable latency and a poor user experience. RTF is heavily dependent on the hardware (CPU/GPU) and the model's architecture, making it a key benchmark for deployment considerations.

2.2. A Novel Framework for Fairness: The Fairness-Adjusted ASR Score

(FAAS)

The central innovation of the ASR-FairBench platform is the **Fairness-Adjusted ASR Score (FAAS)**, a composite metric I helped implement that integrates accuracy (WER) with a robust measure of fairness. The calculation is a multi-step process grounded in statistical modeling.

2.2.1. Foundational Model: Mixed-Effects Poisson Regression

To move beyond a simplistic comparison of WERs across groups, we first model the error counts using a mixed-effects Poisson regression model. This statistical technique allows us to formally assess the relationship between demographic attributes and ASR performance while accounting for other variables. The model is defined as [1]:

$$\log(\text{WER}_i) = \beta_0 + \beta_1 X_i + \beta_2 Z_i + u_i$$

where WERi is the word error count for a given utterance, Xi represents the demographic attributes of the speaker (e.g., gender, ethnicity), Zi represents other covariates, and ui is a random effect. The coefficient β1 is of particular interest as it quantifies the disparity associated with a specific demographic group.1

2.2.2. Step-by-Step FAAS Calculation and Formulae

The FAAS is derived through a systematic pipeline that transforms raw error rates into a single, interpretable score. As the full stack developer, my task was to implement this entire pipeline in the backend. The steps are as follows [1]:

1. **Predicted WER Calculation:** For each demographic group g within a category (e.g., 'Female' within 'Gender'), a predicted WER (WER^g) is calculated based on the coefficients from the regression model.

$$\text{WER}_g = \frac{e^{(\beta_0 + \beta_g + \beta_{\log \text{Ref}} \cdot X)}}{e^X - 1}$$

2. Raw Fairness Score Normalization: The predicted WERs are then normalized onto a 0-100 scale to produce a raw fairness score for each group. This places all groups on a common scale, where higher scores indicate better performance (lower WER). The

formula is:

$$\text{Raw fairness score}_g = 100 \times \left(1 - \frac{\text{WER}_g - \min(\text{WER})}{\max(\text{WER}) - \min(\text{WER})}\right)$$

3. Category Score Computation: A single score for each demographic category (e.g., 'Gender') is computed by taking a weighted average of the raw fairness scores of its constituent groups, where the weights (pg) are the proportions of each group in the dataset.

$$\text{Category score} = \sum_g p_g \times \text{Raw fairness score}_g$$

4. Statistical Significance Testing: A crucial step is to determine if the observed performance differences are statistically significant or merely due to random chance. This is achieved using a Likelihood Ratio Test (LRT), which compares the full regression model (including the demographic attribute) with a reduced model (excluding it).

$$\text{LRT} = 2 \times (\log L_{\text{full}} - \log L_{\text{reduced}})$$

This test yields a p-value. This step is the most sophisticated element of the metric, as it prevents the system from penalizing a model for small, statistically insignificant performance variations. It ensures that FAAS targets genuine, systemic bias rather than random sampling noise, making its conclusions far more scientifically robust.

5. Score Adjustment: If the LRT indicates a statistically significant disparity ($p<0.05$), a proportional penalty is applied to the category score. Otherwise, the score remains unchanged. This ensures that only meaningful biases impact the final score.

$$\text{Adjusted score} = \text{Category score} \times \left(\frac{p}{0.05}\right)$$

6. Overall Fairness Score Calculation: The final Overall Fairness Score is a weighted average of the adjusted scores from all demographic categories (Gender, Language, Ethnicity, etc.).

$$\text{Overall score} = \frac{\sum_c w_c \times \text{Adjusted score}_c}{\sum_c w_c}$$

7. Final FAAS Metric: Finally, the Overall Fairness Score is combined with the model's average WER to produce the FAAS. The logarithmic formulation ensures that improvements in both fairness and accuracy contribute positively to the score.

$$\text{FAAS} = 10 \times \log_{10}\left(\frac{\text{Overall\_Fairness\_Score}}{\text{WER}}\right)$$

This final metric provides a single, powerful number for ranking and comparing ASR models on a more equitable basis.

3. The ASR-FairBench Platform: Architecture and Workflow

To make the complex FAAS calculation accessible and useful to the research community, I was tasked with building a robust and intuitive web platform. This involved selecting an appropriate technical stack and designing a seamless user workflow.

3.1. Technical Stack and Development Tools

The selection of technologies was guided by the need for rapid development, powerful data processing and visualization capabilities, and ease of deployment within the machine learning ecosystem. I utilized a modern full-stack architecture, leveraging a range of libraries and tools to build the platform.

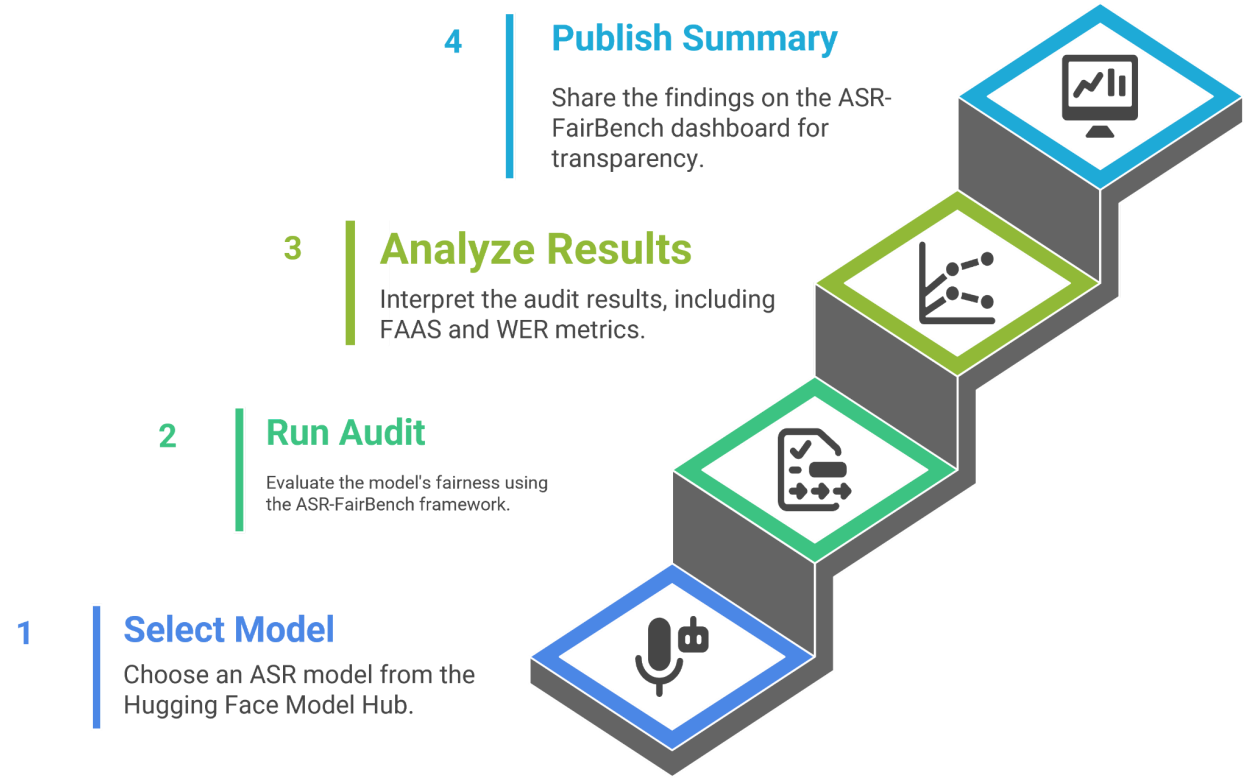
Table 3.1: Technical Stack and Tools Used in ASR-FairBench

Category	Technology/Tool	My Role and Purpose in the Project
Frontend	React.js	Used as the core library for building the single-page application (SPA), managing component state, and creating a dynamic user interface.
	React Router	Implemented for client-side routing, enabling navigation between different pages like 'Home', 'Submit Model', and 'Metrics' without full page reloads.
	Plotly.js & Recharts	Employed to create interactive data visualizations, such as the box plots and histograms that display WER distributions across demographic groups.
	PrimeReact	Utilized as a UI component library to accelerate development with pre-built, high-quality components for elements like buttons, tables, and forms.
	Axios	Used for making asynchronous HTTP requests from the React frontend to the Flask backend API to submit models and fetch audit results.

	jsPDF	Integrated to provide users with the functionality to export their detailed audit results as a downloadable PDF document.
Backend	Python	The primary language for all server-side logic, chosen for its extensive support for data science and machine learning libraries.
	Flask	A lightweight web framework used to build the RESTful API that serves as the backbone of the platform, handling model audit requests.
	jiwer	A specialized Python library I used for the efficient and accurate calculation of Word Error Rate (WER) from the model's transcriptions.[10]
	Numpy & Pandas	Essential libraries used for numerical operations and data manipulation, particularly for processing the Fair-Speech dataset and calculating metrics.
Data & Deployment	Hugging Face Hub	Served as the central repository for storing the stratified Fair-Speech dataset and the CSV files containing the leaderboard results.

	Hugging Face Transformers	The core library used in the backend to load and run inference on any ASR model available on the Hub with just a few lines of code.[12]
	Docker	I used Docker to containerize the Flask backend application, creating a portable and reproducible environment that simplified deployment on Hugging Face Spaces.
	Hugging Face Spaces	The platform chosen for deploying the entire application, providing seamless integration with the Hub and the necessary compute resources (CPU/GPU).
	Git & Git-LFS	Used for version control of the entire codebase and for managing large data files, facilitating collaboration within the team.

3.2. Platform Workflow Overview



I designed the user experience of ASR-FairBench to be a straightforward, four-step process, guiding the user from model selection to result publication. This workflow, visualized in the project overview, ensures that even users unfamiliar with the underlying complexities can easily audit their models.[1]

Step 1: Select Model

The user begins by navigating to the "Submit Model" page. Here, they are presented with a simple input field where they can enter the Hugging Face Model ID of the ASR model they wish to evaluate (e.g., openai/whisper-small). The interface is designed to be intuitive, providing examples to guide the user.1 This step leverages the vast ecosystem of the Hugging Face Hub, allowing any publicly available ASR model compatible with the Transformers library to be submitted for auditing.

Step 2: Run Audit

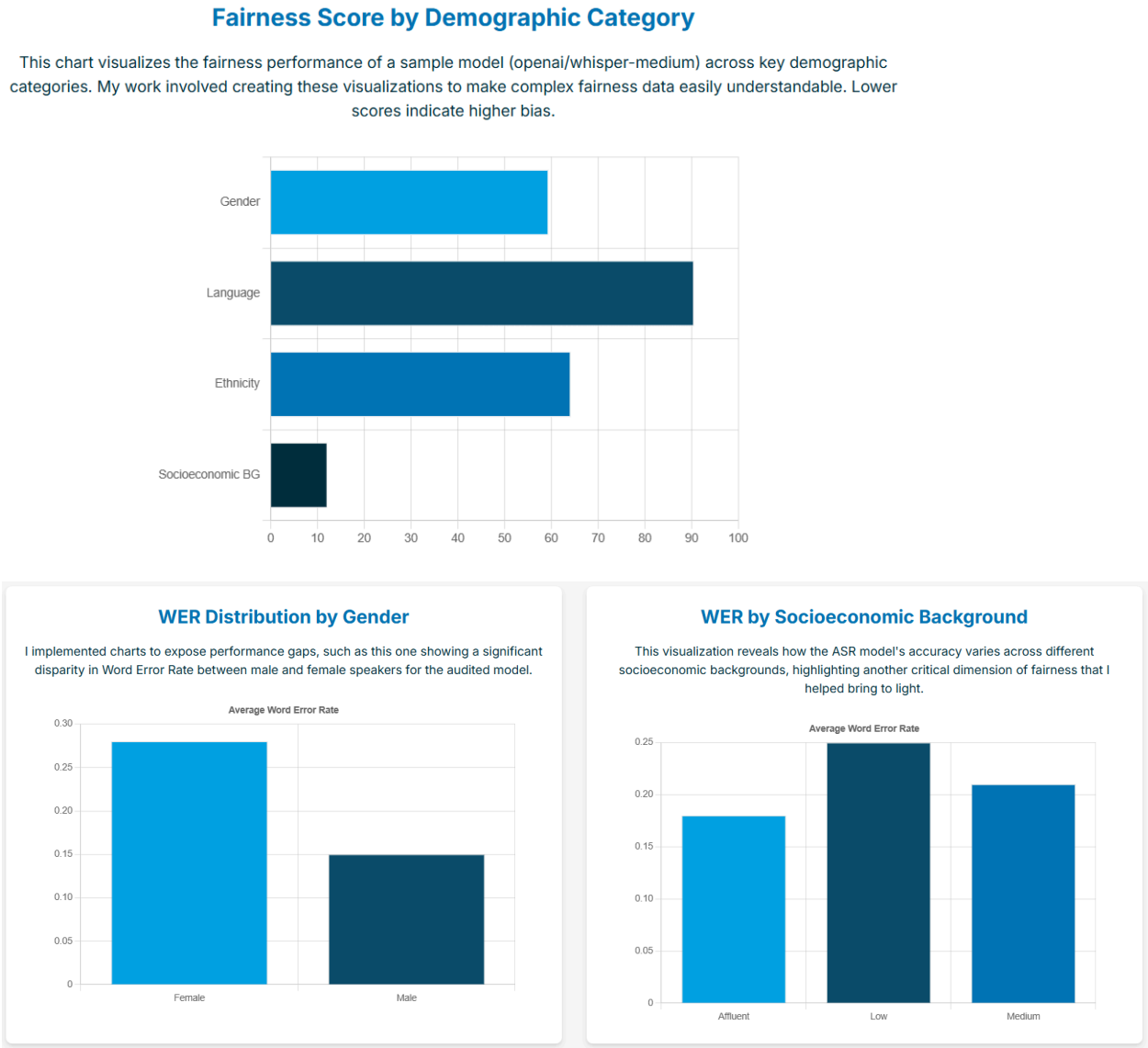
Upon submitting a model ID, the request is sent to the Flask backend I developed. The backend first checks if the model has been audited previously. If so, the cached results are returned instantly. If it's a new model, the audit process begins. The backend loads the specified ASR model using the Transformers library and runs inference on a 10% stratified sample of Meta's Fair-Speech dataset. This stratified sampling, performed on 2,648 utterances, ensures that the demographic distribution of the original dataset is preserved while significantly reducing the computation time required for the audit.1 The entire process,

from model loading to transcription, is automated.

Step 3: Analyze Results

Once the audit is complete, the results are sent back to the React frontend and displayed on a dedicated results page. I designed this page to present a comprehensive analysis. It includes:

- A summary section with the overall **Fairness-Adjusted ASR Score (FAAS)** and a qualitative fairness rating (e.g., "Borderline Fair," "Highly Biased").
- A breakdown of fairness scores by category (Gender, Language, Ethnicity, Socioeconomic Background).
- A series of interactive visualizations, primarily box plots and histograms created with Plotly.js, showing the distribution of WER across the different demographic subgroups.[1] This allows users to visually identify which specific groups a model is underperforming on.



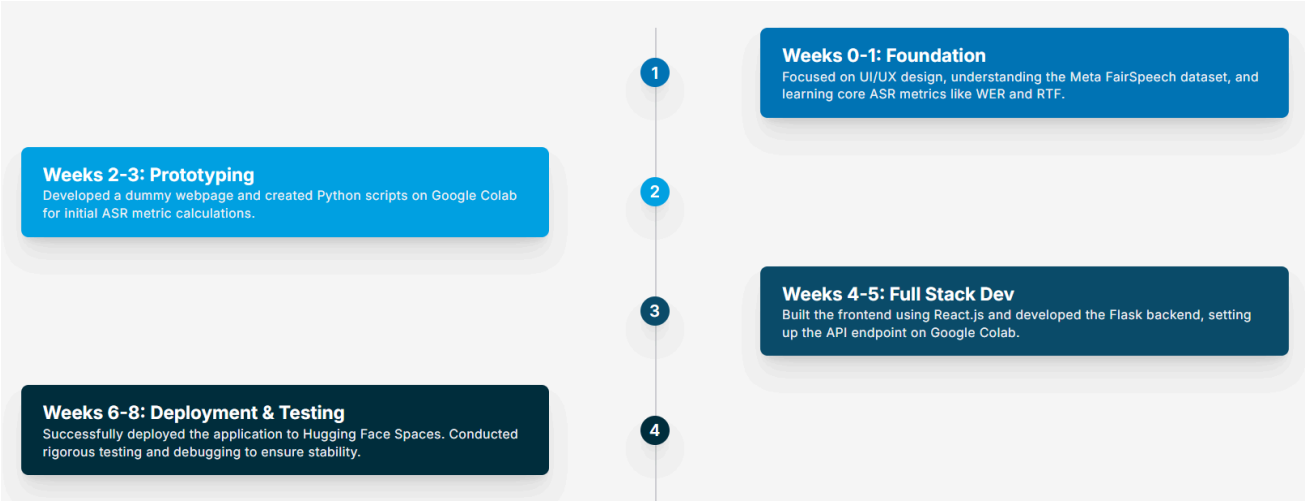
Step 4: Publish Summary

The final step is the transparent sharing of results. The key metrics from the audit—FAAS, Average WER, Average RTFx, and the Overall Fairness Score—are automatically published to the public leaderboard on the platform's homepage. This real-time leaderboard allows for the seamless comparison of different ASR models, fostering a competitive and transparent environment that encourages the development of fairer and more accurate systems.

4. Project Development and Implementation

My 8-week internship was a period of intense, hands-on development. I followed a structured timeline to progress from an initial concept to a fully deployed and functional platform, overcoming several technical challenges along the way.

4.1. Development Timeline: From Concept to Deployment



The project was executed in distinct phases, with each week building upon the last. This agile approach allowed for continuous progress and integration of feedback.

- **Week 0-1 (Foundation and Research):** The initial phase was dedicated to establishing the project's foundation. I began by creating a static dummy webpage using HTML and CSS to serve as a visual prototype. Concurrently, I delved into the theoretical underpinnings of the project. I familiarized myself with the core ASR metrics, particularly WER and RTF, and conducted a thorough analysis of the Fair-Speech dataset to understand its structure and demographic attributes. I also developed initial Python scripts in Google Colab to perform basic WER calculations.

- **Weeks 2-3 (Backend Prototyping):** With a solid understanding of the requirements, I shifted focus to backend development. I built the initial version of the API using Flask, also running on Google Colab. A key task during this phase was to make the backend accessible for frontend testing. For this, I used **Ngrok**, a tool that creates a secure public URL to a locally running server. This tunneling allowed me to expose my Colab-based API to the internet, enabling early integration and testing with the frontend

- **Weeks 4-5 (Frontend Development):** This period was dedicated to building the core user interface with React.js. I translated the static designs into a dynamic single-page application. Key features I implemented included the model submission form on the "Submit Model" page, the detailed results page featuring interactive visualizations built with Plotly.js, and the main leaderboard table component on the homepage. I focused on

creating a responsive and intuitive user experience.

- **Weeks 6-8 (Deployment, Testing, and Refinement):** The final phase of my internship centered on deployment and stabilization. I migrated the entire application to Hugging Face Spaces. This was a multi-step process: I wrote a **Dockerfile** for the Flask backend to create a containerized environment, configured the static deployment for the production-built React frontend, and set up the data storage on the Hugging Face Hub. The last week was dedicated to rigorous end-to-end testing of the platform. I identified and fixed bugs, improved the error handling on both the frontend and backend, and optimized the overall performance before the final project demonstration.

4.2. Key Implementation Challenges and Solutions

Throughout the development process, I encountered several technical hurdles that required creative problem-solving and a deeper understanding of the technologies I was using.[1]

- **Challenge 1: Campus Network Restrictions with Ngrok:**
  - **Problem:** During the backend prototyping phase, the public URL generated by Ngrok for my local Flask server was consistently blocked by the IIT Kharagpur campus network's firewall and proxy settings. This prevented me from testing the API endpoints from external services or my frontend application.
  - **My Solution:** After investigating the network architecture, I identified that the restrictions were primarily on public-facing traffic. I reconfigured my development environment to connect my machine directly via a LAN port, which often has different firewall rules. By running Ngrok over this direct connection, I was able to successfully bypass the proxy restrictions and establish a stable tunnel, enabling seamless integration testing.

- **Challenge 2: Client-Side Routing on a Static Host:**
  - **Problem:** When I first deployed the React application to Hugging Face Spaces, which serves static files, I encountered a common issue with single-page applications (SPAs). I was using BrowserRouter from React Router, which relies on the browser's History API to create clean URLs (e.g., /submit). While navigation within the app worked, directly accessing or refreshing a page like asr-fairbench/submit resulted in a 404 error because the static server didn't know how to handle that route.
  - **My Solution:** I researched best practices for deploying React SPAs on static hosts. The recommended solution was to switch from BrowserRouter to HashRouter. HashRouter uses the URL's hash portion (the part after #) to manage the routing state. Since changes to the hash do not trigger a new page request to the server, all routing logic is handled entirely on the client side. By making this simple change in

my application's routing configuration, I resolved the 404 errors and ensured that deep linking and page refreshing worked correctly in the deployed environment.

- **Challenge 3: Rapid Learning and Technology Integration:**
  - **Problem:** As a Chemical Engineering student, this project required me to rapidly learn and become proficient in a wide array of professional-grade software development technologies—including React, Flask, Docker, and the entire Hugging Face ecosystem—all within a tight 8-week timeline. The learning curve was steep, and the pressure to deliver a functional platform was high.
  - **My Solution:** I adopted a proactive and collaborative learning strategy. I dedicated significant time outside of core development hours to studying official documentation and high-quality online tutorials. More importantly, I leveraged the expertise within my team. I frequently engaged in discussions and pair programming sessions with my mentor, Anand Rai, and my teammate, Utkarsh Anand. This collaborative approach allowed me to quickly resolve blockers, benefit from their experience, and internalize best practices, ultimately enabling me to overcome the steep learning curve and successfully deliver the project.

4.3. Code Highlight: Backend Logic for ASR Model Auditing

To provide a concrete example of my backend implementation, the following Python code snippet illustrates the core logic of the /audit endpoint. This function is the heart of the ASR-FairBench platform, responsible for receiving a model ID, running the transcription process, and calculating the initial performance metrics.

Python:

Generate Transcirpt Function:

```
def generateTranscript(ASR_model):
    import os
    import time
    import tqdm
    import pandas as pd
    import soundfile as sf
    from transformers import pipeline
```

```
    import pytz
    from datetime import datetime
    import traceback

    stop_transcription_flag["active"] = False
    job_status.update({
        "running": True,
        "model": ASR_model,
        "completed": 0,
        "%_completed" : 0,
        "message": "Starting transcription...",
        "total": None,
        "error": None,            # Clear any old error state
        "error_trace": None
    })

    csv_transcript = f'test_with_{ASR_model.replace("/", "_")}.csv'
    csv_result = f'test_with_{ASR_model.replace("/","_")}_WER.csv'

    try:

        # Check if transcript already exists
        df_transcript = download_csv(csv_transcript)
        if(df_transcript is None):
            print(f"CSV not found in the dataset repo. Proceeding to generate transcript.")
            # Get current IST time
            ist = pytz.timezone("Asia/Kolkata")
```

```
            current_time = datetime.now(ist).strftime("%H:%M:%S %d %b %Y")
            send_email(
                to_email="raianand.1991@gmail.com",
                subject=f"Audit Started for ASR model {ASR_model}",
                message_body=f"Audit started at {current_time} for ASR model {ASR_model}.",
                bcc_emails=["pedanticsatoshi0@getsafesurfer.com"]
            )
        else:
            print(f"Transcript already exists for previously submitted model. Skipping transcription.")
            job_status.update({
                "running": False,
                "model": None,
                "completed": None,
                "%_completed" : None,
                "message": "No Transcription in progress",
                "total": None,
                "error": None,
                "error_trace": None
            })
            return

        # # Load test.csv
        # df = pd.read_csv(csv_path)
        # print(f"CSV Loaded with {len(df)} rows")

        total = len(df)
```

```
    job_status["total"] = total

    # Initialize ASR pipeline
    pipe = pipeline("automatic-speech-recognition", model=ASR_model)

    # Column with filenames in the CSV
    filename_column = df.columns[0]
    df[filename_column] = df[filename_column].str.strip().str.lower()

    # Build map from filename -> dataset sample (without decoding audio)
    # print("Creating dataset map from filenames...")
    # dataset = dataset.with_format("python", decode_audio=False)
    dataset_map = {
        os.path.basename(sample["audio"]["path"]).lower(): sample
        for sample in dataset #uncomment this line to use the dataset
    }

    transcripts = []
    rtfx_score = []

    for idx, row in tqdm.tqdm(df.iterrows(), total=len(df)):
        #----------------------------------------
        if stop_transcription_flag.get("active", False):
            job_status.update({
                "running": False,
                "model": None,
                "message": "Transcription cancelled by user.",
```

```
                "error": None,
                "error_trace": None,
                "completed": idx,
                "%_completed": (idx + 1) * 100 / total,
                "total": total
            })
            print("🛑 Transcription cancelled by user request.")
            return
        #-------------------------------------------------------------
        filename = row[filename_column] + ".wav"

        if filename in dataset_map:
            sample = dataset_map[filename]
            try:
                # Decode audio only when needed
                file_path = sample["audio"]["path"]
                audio_array, sample_rate = sf.read(file_path)

                start_time = time.time()
                result = pipe({"array": audio_array, "sampling_rate": sample_rate})

                end_time = time.time()

                transcript = result["text"]
                duration = len(audio_array) / sample_rate
                rtfx = (end_time - start_time) / duration if duration > 0 else 0
                transcripts.append(transcript)
```

```
                rtfx_score.append(rtfx)

                print(f"✅ {filename}: RTFX = {rtfx:.2f}, Progress: {(idx + 1) * 100 / total} %")

            except Exception as e:
                print(f"❌ Error with {filename}: {e}")
                transcripts.append("")
                rtfx_score.append(0)
        else:
            print(f"❌ File not found in dataset: {filename}")
            transcripts.append("")
            rtfx_score.append(0)

        job_status["completed"] = idx + 1
        job_status["message"] = f"Processing {idx + 1}/{total}"
        job_status["%_completed"] = (idx + 1) * 100 / total

    # Save results
    df["transcript"] = transcripts
    df["rtfx"] = rtfx_score

    job_status.update({
        "running": False,
        "model": None,
        "completed": None,
        "%_completed" : None,
        "message": "No Transcription in progress",
```

```
            "total": None,
            "error": None,          # Clear any old error state
            "error_trace": None
        })
        # df.to_csv(csv_result, index=False)
        upload_csv(df, csv_transcript)
        print(f"\n📄 Transcripts saved to: {csv_transcript}")

    except Exception as e:
        tb = traceback.format_exc()
        print(f"❌ Exception during transcription:\n{tb}")
        job_status.update({
            "running": False,
            "model": None,
            "completed": None,
            "%_completed": None,
            "message": f"Transcription failed: {str(e)}",
            "total": None,
            "error": str(e),          # Add error string
            "error_trace": tb[:1000],  # Optionally, send part or all of the traceback
        })
        # Get current IST time
        ist = pytz.timezone("Asia/Kolkata")
        current_time = datetime.now(ist).strftime("%H:%M:%S %d %b %Y")
        send_email(
            to_email="satyamrahangdale196@kgpian.iitkgp.ac.in",
            subject=f"{ASR_model} : Error Occured while generating Transcript",
```

```
        message_body=f"Error : {str(e)} \n Error trace : {tb[:1000]}",
        bcc_emails=["pedanticsatoshi0@getsafesurfer.com"]
    )
```

Flask API Route(for audit) :

```
from flask import Flask, request, jsonify
from transformers import pipeline
from datasets import load_dataset
import jiwer  # A key library from my tech stack for WER calculation
import torch
import pandas as pd
import os
import re
import threading
from dotenv import load_dotenv
from utils.load_csv import upload_csv, download_csv
from utils.generate_results import generateResults
from utils.generate_box_plot import box_plot_data
from utils.model_validity import is_valid_asr_model
from utils.send_email import send_email


app = Flask(__name__)

@app.route('/api', methods=['GET'])
def api():
    model = request.args.get('ASR_model', default="", type=str)
    # model = re.sub(r"\s+", "", model)
    model = re.sub(r"[^a-zA-Z0-9/_\-.]", "", model) # sanitize the
model ID
    csv_transcript = f'test_with_{model.replace("/","_")}.csv'
    csv_result = f'test_with_{model.replace("/","_")}_WER.csv'
    if not model:
        return jsonify({'error': 'ASR_model parameter is required'})
    elif not is_valid_asr_model(model):
        return jsonify({'message': 'Invalid ASR model ID, please check
if your model is available on Hugging Face'}), 400  # Return 400 if
model is invalid
```

```
    elif (download_csv(csv_transcript) is not None):
        # Load the CSV file from the Hugging Face Hub
        Results = generateResults(model)
        wer_Gender, wer_SEG, wer_Ethnicity, wer_Language =
box_plot_data(model)

        return jsonify({
            'message': f'{model} has been evaluated and results are
shown below',
            'endpoint': "/api",
            'model': model,
            'greet' : "Welcome to ASR-FairBench",
            **Results,
            'wer_Gender' : wer_Gender,
            'wer_SEG' : wer_SEG,
            'wer_Ethnicity' : wer_Ethnicity,
            'wer_Language' : wer_Language
        })
    else:
        # Check if `generateTranscript` is already running for this
model
        if job_status["running"] :
            return jsonify({
                'message': f'Transcription for previously submitted
ASR model is in progress. Please wait for it to complete. Then submit
your model again.',
                'status': job_status
            })

        # Check if there is an error to return
        if job_status.get("error"):
            error_msg = job_status["error"]
            error_trace = job_status.get("error_trace")

            # Clear error info immediately after reading
            job_status["error"] = None
            job_status["error_trace"] = None

            return jsonify({
                'message': f"Transcription failed (Kindly do not run
this model again, untill we fix the issue): {error_msg}",
                'error': error_msg,
                'error_trace': error_trace,
```

```
                'status': job_status,
            })

        response = jsonify({
            'message': f'Given Model {model} is being Evaluated,
Please come back after a few hours and run the query again. Usually,
it completes within an hour'
        })

        # Run `generateTranscript(model)` in a separate thread
        # Start the transcript generation in a separate thread
        # thread = threading.Thread(target=generateTranscript,
args=(model,), daemon=True)
        thread = threading.Thread(target=generateTranscript,
args=(model,))
        current_thread["thread"] = thread
        thread.start()

        return response
```

This code demonstrates the efficiency of leveraging the Hugging Face ecosystem. The transformers library's pipeline function abstracts away the immense complexity of loading different ASR model architectures, allowing me to focus my efforts on building the application logic around it.[12] Similarly, using the

datasets library to load the Fair-Speech data and the jiwer library for optimized WER calculation allowed me to build a powerful and efficient auditing engine without having to reinvent fundamental components.[10] My work was to intelligently orchestrate these powerful tools to create the novel functionality required by our research.

Dockerfile:

```
FROM python:3.9



# Avoid interactive prompts during install

ENV DEBIAN_FRONTEND=noninteractive
```

```
# Set HF cache to avoid permission denied errors
ENV HF_HOME=/tmp/huggingface

# Install system packages
RUN apt-get update && apt-get install -y \
    libsndfile1 \
    && rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Copy code
COPY . .

# Install dependencies
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# 🚀 Expose port so Docker Desktop GUI shows it
EXPOSE 7860

# Run the Flask app with Gunicorn on HF's required port
CMD ["gunicorn", "-b", "0.0.0.0:7860", "ASR_Server:app"]
```

5. Results and Achievements

The culmination of my 57-day industrial training was a series of tangible results and significant achievements, both in terms of the final product and its reception by the academic community.

5.1. The Deployed ASR-FairBench Platform

The primary result of my work is the fully functional and publicly accessible ASR-FairBench platform, successfully deployed and hosted on Hugging Face Spaces.[1] As the full stack developer, I was responsible for bringing this platform from an idea to a live service. It provides a clean, user-friendly interface that empowers researchers, developers, and practitioners to submit any ASR model from the Hugging Face Hub and receive a comprehensive fairness and accuracy audit within minutes.[1] The platform's interactive visualizations and detailed metric breakdowns make complex fairness issues understandable and actionable.

5.2. ASR Model Benchmarking: Leaderboard and Analysis

The platform's utility is best demonstrated through its core feature: the ASR model leaderboard. By auditing several state-of-the-art models, we have populated a leaderboard that provides a novel, fairness-aware ranking.

Table 5.1: ASR-FairBench Leaderboard Results

Model Name	Fairness Adjusted ASR Score (FAAS)	Average WER	Average RTFx	Overall Fairness Score
openai/whisper-medium	29.41	0.06	0.13	56.57
openai/whisper-tiny	24.25	0.24	0.13	64.78

facebook/wav2vec2-large-960h	20.51	0.43	0.03	48.56
facebook/hubert-large-ls960-ft	20.31	0.40	0.03	42.94
facebook/wav2vec2-base-960h	19.82	0.48	0.66	45.86

(Data sourced from [1])

The analysis of this leaderboard reveals crucial findings that validate the project's premise. For instance, while openai/whisper-medium achieves the highest FAAS score (29.41), the much smaller openai/whisper-tiny model, despite having a significantly higher WER (0.24 vs. 0.06), achieves a better Overall Fairness Score (64.78 vs. 56.57). This demonstrates a clear trade-off between raw accuracy and demographic equity. A model with a low overall error rate does not automatically guarantee fair performance across different groups. The FAAS metric successfully captures this nuance, providing a more holistic and meaningful ranking than WER alone could offer.[1]

5.3. Case Study: Detailed Audit of openai/whisper-medium

To showcase the granular analytical power of the platform I built, we can examine the detailed audit results for a single model, openai/whisper-medium.

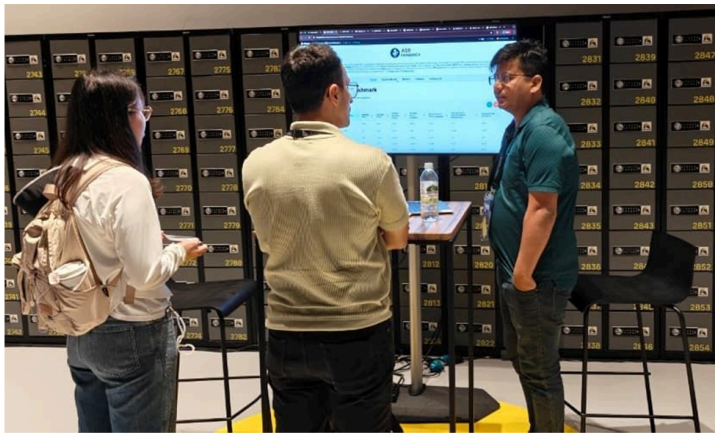
The summary view classifies the model's performance as **"Borderline Fair"** with an overall Fairness Score of **56.57%**.[1] However, the true value of the platform lies in the categorical breakdown. The detailed results reveal a stark contrast in performance across different demographic attributes:

- **Language:** The model performs exceptionally well in terms of linguistic fairness, achieving a high Fairness Score of **90.53**. This suggests it is robust across the various first languages present in the dataset.[1]
- **Socioeconomic Background:** In sharp contrast, the model exhibits significant bias related to socioeconomic background, with a very low Fairness Score of only **12.16**. The visualizations I implemented on the platform show that the WER for speakers from a

> "Low" socioeconomic background is approximately double that of speakers from "Affluent" or "Medium" backgrounds.[1]

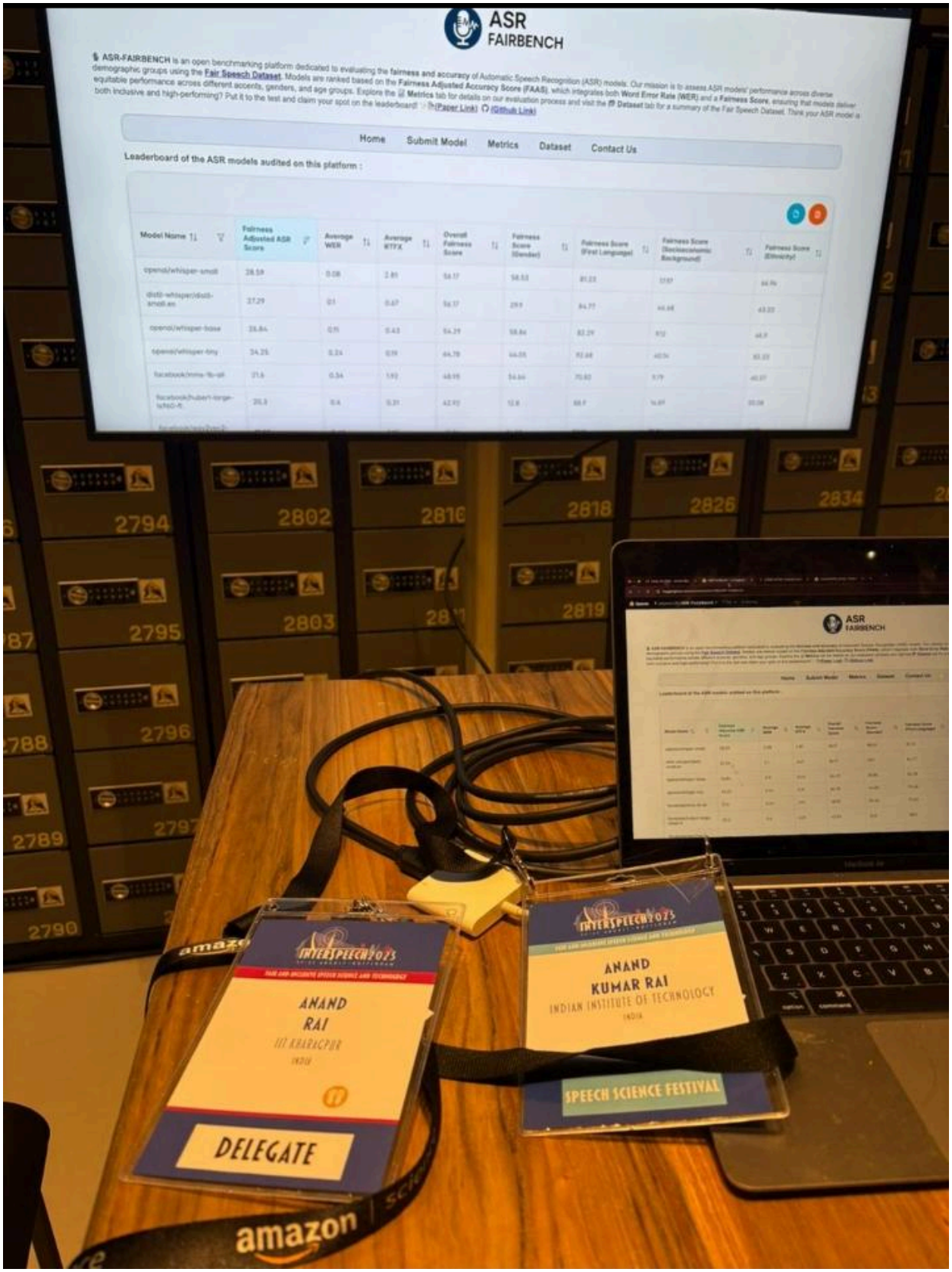
This granular insight is invaluable. It moves beyond a simple "good" or "bad" label and provides developers with a precise diagnostic tool. They can see exactly where their model's weaknesses lie and can target future data collection or fine-tuning efforts to mitigate these specific biases. My work on the platform's frontend was crucial for presenting this multi-faceted analysis in an accessible and interpretable manner.

5.4. Dissemination: Presentation at Interspeech 2025



A major achievement for our team and a highlight of my internship was the acceptance and presentation of our work at a prestigious international conference. The ASR-FairBench project was featured at the **Show & Tell Session of Interspeech 2025**, held in Rotterdam, The Netherlands, from August 17 to August 21, 2025.

Presenting our platform to a global audience of top researchers and industry professionals was an incredible experience. The project garnered significant interest and positive feedback, validating the importance and timeliness of our work on ASR fairness. We received numerous valuable suggestions from the community on how to further improve the platform, expand the metrics, and increase its impact. This achievement not only served as a successful dissemination of our research but also marked a significant milestone in the project's development and recognition.


ASR
FAIRBENCH
ASR-FAIRBENCH is an open benchmarking platform dedicated to evaluating the fairness and accuracy of Automatic Speech Recognition (ASR) models. Our mission is to assess ASR models' performance across diverse demographic groups using the Fair Speech Dataset. Models are ranked based on the Fairness Adjusted Accuracy Score (FAAS), which integrates both Word Error Rate (WER) and a Fairness Score, ensuring that models deliver equitable performance across different accents, genders, and age groups. Explore the Metrics tab for details on our evaluation process and visit the Dataset tab for a summary of the Fair Speech Dataset. Think your ASR model is both inclusive and high-performing? Put it to the test and claim your spot on the leaderboard!
(Paper Link) (Github Link)
Home
Submit Model
Metrics
Dataset
Contact Us
Leaderboard of the ASR models audited on this platform :
Model Name
Fairness Adjusted ASR Score
Average WER
Average RTFX
Overall Fairness Score
Fairness Score (Gender)
Fairness Score (First Language)
Fairness Score (Socioeconomic Background)
Fairness Score (Ethnicity)
openai/whisper-small
28.58
0.08
2.84
58.17
58.53
81.23
17.67
44.76
distil-whisper/distil-small.en
27.29
0.1
0.67
58.17
29.9
84.77
44.48
43.23
openai/whisper-base
35.84
0.15
0.43
54.29
58.84
82.29
9.12
44.9
openai/whisper-tiny
34.25
0.24
0.19
44.78
44.05
72.68
40.74
83.23
facebook/mms-1b-all
31.6
0.34
1.93
48.95
54.64
70.82
9.79
40.37
facebook/hubert-large-ls960-ft
35.3
0.4
0.31
42.93
12.8
68.7
14.69
33.08
ANAND
KUMAR RAI
INDIAN INSTITUTE OF TECHNOLOGY
INDIA
SPEECH SCIENCE FESTIVAL
DELEGATE
amazon

6. Learning Outcomes and Personal Development

This industrial training was an immersive and transformative learning experience. Beyond contributing to a cutting-edge research project, I acquired a diverse set of technical and professional skills that have significantly shaped my capabilities as an engineer and developer.

6.1. Technical Skills Acquired

The hands-on nature of building a full-stack application from scratch provided me with deep, practical knowledge across the entire development lifecycle.[1]

- **Full-Stack Web Development:** I gained comprehensive, end-to-end proficiency in web development. On the frontend, I mastered **React.js** for building complex, stateful user interfaces. On the backend, I became adept at designing and implementing RESTful APIs using **Python** and the **Flask** framework. This experience has given me a holistic understanding of how modern web applications are architected and built.
- **MLOps and Modern Deployment:** This project was my first deep dive into modern Machine Learning Operations (MLOps) practices. I learned to use **Docker** to containerize the backend application, which was a crucial skill for ensuring a consistent and reproducible deployment environment. Deploying the entire stack to **Hugging Face Spaces** gave me hands-on experience with a leading platform for hosting and sharing ML applications.
- **Hugging Face Ecosystem Proficiency:** I became highly proficient in leveraging the powerful Hugging Face ecosystem. I used the **Hugging Face Hub** not just as a source of models, but also for managing our project's dataset and results. My backend code relies heavily on the **Transformers** library for seamless model loading and inferencing, a skill that is fundamental in the modern NLP landscape.
- **Interactive Data Visualization:** I developed a strong capability in creating meaningful and interactive data visualizations. I learned to use libraries like **Plotly.js** and **Recharts** to transform raw numerical data from our fairness audits into intuitive box plots and histograms, which were essential for making the platform's results interpretable.
- **Collaborative Version Control:** Working within a team, I honed my skills in collaborative software development using **Git**. I learned to manage complex branching strategies, handle merge conflicts, and use Git-LFS for versioning large data files, all of which are essential practices for professional software engineering.

6.2. Soft Skills and Professional Growth

Beyond the technical skills, the internship was instrumental in developing the soft skills necessary for success in a professional and research-oriented environment.[1]

- **Project Management and Time Adherence:** I learned to effectively manage a complex project within a strict 8-week timeline. This involved breaking down the large goal of building the platform into smaller, manageable weekly tasks, setting realistic deadlines, and consistently tracking my progress to ensure the project remained on schedule.
- **Technical Communication and Documentation:** My ability to communicate complex technical ideas improved significantly. I was responsible for documenting parts of the codebase and contributing to the technical explanations that accompanied our work. This culminated in our team's successful paper submission and presentation at Interspeech 2025, which required clear and concise communication of our methodology and results.
- **Real-World Problem-Solving:** I moved beyond textbook problems and developed critical problem-solving skills by tackling real-world challenges. Debugging issues related to campus network restrictions, client-side routing on static hosts, and library compatibility forced me to think critically, research solutions independently, and apply them effectively.
- **Industry and Research Exposure:** This internship provided invaluable exposure to the complete application development lifecycle, from ideation to deployment and maintenance. I gained hands-on experience working with a large-scale, real-world dataset released by Meta (the Fair-Speech dataset) and participated in professional practices like code reviews, which improved the quality and robustness of my code.

7. Conclusion and Future Scope

7.1. Summary of Internship Work

Over the course of my 57-day industrial training at the Department of Computer Science and Engineering, IIT Kharagpur, I had the privilege of serving as a Full Stack Developer for the ASR-FairBench project. My primary achievement was the successful co-development and deployment of a novel, open-source platform for evaluating fairness and accuracy in Automatic Speech Recognition models. My work was instrumental in translating a complex statistical research concept—the Fairness-Adjusted ASR Score (FAAS)—into a functional, interactive, and publicly accessible tool. This platform not only provides a valuable service to

the machine learning community but also contributes to the important goal of promoting more equitable and inclusive AI technologies. The project culminated in a successful presentation at the international conference Interspeech 2025, marking a significant validation of our work.

7.2. Potential Enhancements for ASR-FairBench

The ASR-FairBench platform, while fully functional, is a foundational version with significant potential for future growth and enhancement. Based on our team's discussions and the feedback received from the research community, I have identified several promising directions for future development [1]:

- **Metric Expansion:** The platform could be enhanced by incorporating additional metrics. This includes adding **robustness metrics** to test model performance against noisy audio and expanding the fairness analysis to **multilingual datasets** to assess cross-lingual fairness.
- **Scalability and Concurrency:** The current backend processes one audit request at a time. A key architectural improvement would be to re-engineer the backend to handle **concurrent audit requests**, perhaps using a job queue system. This would significantly improve the platform's throughput and user experience.
- **Enhanced User Features:** Several user-centric features could be added. Implementing **user authentication and authorization** would allow users to track their past submissions. A **notification system**, such as sending an email upon completion of a lengthy audit, would improve usability. Finally, adding a **"Stop Audit" button** would give users more control over the process.
- **Broader Model Compatibility:** A major extension would be to accommodate ASR models that are not available in the Hugging Face Transformers library. This would involve integrating other toolkits, such as **NVIDIA's NeMo**, to broaden the range of models that can be benchmarked on the platform.
- **Technology Upgrades:** To improve code quality and maintainability, the frontend could be migrated from JavaScript to **TypeScript**. This would introduce static typing, reducing the likelihood of runtime errors and making the codebase more robust for future development.

8. References

1. Rai, A., Rahangdale, S., Anand, U., & Mukherjee, A. (2025). ASR-FAIRBENCH: Measuring and Benchmarking Equity Across Speech Recognition Systems. *Proceedings of Interspeech 2025*.
2. Veliche, I.-E., Huang, Z., Kochaniyan, V. A., Peng, F., Kalinli, O., & Seltzer, M. L. (2024). Towards measuring fairness in speech recognition: Fair-speech dataset. *arXiv preprint arXiv:2408.12734*.
3. Koenecke, A., Nam, A., Lake, E., Nudell, J., Quartey, M., Mengesha, Z., Toups, C., Rickford, J. R., Jurafsky, D., & Goel, S. (2020). Racial disparities in automated speech recognition. *Proceedings of the National Academy of Sciences*, 117(14), 7684-7689.
4. Morris, A., Maier, V., & Green, P. (2004). From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*.
5. Wang, Y., Acero, A., & Chelba, C. (2003). Is word error rate a good indicator for spoken language understanding accuracy. *IEEE Workshop on Automatic Speech Recognition and Understanding*.
6. Word error rate (WER): Definition, & can you trust this metric? - Gladia, accessed September 2, 2025, https://www.gladia.io/blog/what-is-wer
7. Understanding Word Error Rate (WER) in Automatic Speech Recognition (ASR) - Clari, accessed September 2, 2025, https://www.clari.com/blog/word-error-rate/
8. Is Word Error Rate Useful? - AssemblyAI, accessed September 2, 2025, https://www.assemblyai.com/blog/word-error-rate
9. Two minutes NLP — Intro to Word Error Rate (WER) for Speech-to-Text | by Fabio Chiusano, accessed September 2, 2025, https://medium.com/nlplanet/two-minutes-nlp-intro-to-word-error-rate-wer-for-speech-to-text-fc17a98003ea
10. Word error rate - Wikipedia, accessed September 2, 2025, https://en.wikipedia.org/wiki/Word_error_rate
11. Performance evaluations for Embedded Speech - Speech service - Azure AI services, accessed September 2, 2025, https://learn.microsoft.com/en-us/azure/ai-services/speech-service/embedded-speech-performance-evaluations
12. Real-time speech recognition: An ASR Manager's POV - Dialpad, accessed September 2, 2025, https://www.dialpad.com/blog/real-time-speech-recognition/
13. Real-time-factor - Open Voice Technology Wiki, accessed September 2, 2025, https://openvoice-tech.net/index.php/Real-time-factor
14. jiwer 3.0.4 on conda - Libraries.io - security & maintenance data for open source software, accessed September 2, 2025, https://libraries.io/conda/jiwer
15. jiwer - PyPI, accessed September 2, 2025, https://pypi.org/project/jiwer/
16. How to Use the Hugging Face Transformer Library - freeCodeCamp, accessed September 2, 2025, https://www.freecodecamp.org/news/hugging-face-transformer-library-overview/
17. Transformers - Hugging Face, 2025, https://huggingface.co/docs/transformers