Internship Report

Makers Lab – Tech Mahindra Internship Report

An Internship Report Submitted to:

Makers Lab – Tech Mahindra

Submitted by:

Krishna Lokhande

Harmanjit Singh

Under the supervision of

Fauzal K

(Period from 07/08/2025 to 18/11/2025)

Contents

Abstract:

AutoVaani is an AI-powered car voice assistant developed to provide a safe, intelligent, and hands-free driving experience through natural language interaction. The project leverages advanced speech recognition techniques and transformer-based large language models to understand user commands and respond conversationally. The system is designed using a modular backend architecture implemented in Python with Flask, enabling real-time voice-based interaction. AutoVaani focuses on stability, scalability, and efficient CPU-based execution, making it suitable for environments with limited computational resources. This project highlights the practical implementation of artificial intelligence, natural language processing, and speech technologies in the automotive domain**.**

Introduction:

The automotive industry is witnessing a significant transformation with the integration of artificial intelligence and smart technologies. Voice assistants have become an essential feature in modern vehicles, allowing drivers to interact with systems without manual intervention. However, many existing voice-controlled systems rely on predefined commands and lack conversational intelligence. AUToVaani aims to overcome these limitations by introducing an AI-driven assistant capable of understanding natural speech and generating context-aware responses. The project demonstrates how AI can enhance driver convenience, safety, and engagement by reducing distractions and improving accessibility.

Problem Statement:

Despite the availability of voice-controlled systems in vehicles, many suffer from limited understanding, poor accuracy, and rigid command structures. These systems often fail when faced with natural language variations, accents, or complex queries. Additionally, deploying advanced AI models in real-time environments presents challenges related to hardware constraints, response latency, and system stability. The problem addressed by AutoVaani is the development of an intelligent, conversational, and lightweight voice assistant that operates efficiently while providing accurate and meaningful responses in an automotive setting.

Objective:

The objectives of the AutoVaani project are:

- To design and develop an AI-based voice assistant for automotive use
- To enable seamless speech-to-text and text-to-speech interaction
- To implement a transformer-based language model for intelligent response generation
- To ensure system stability using CPU-based deployment

- To create a scalable and modular backend architecture

- To enhance user experience through natural and conversational interaction

Scope of Project:

The scope of AutoVaani is limited to software-level implementation and does not include direct hardware integration with vehicle systems. The project serves as a proof-of-concept for intelligent voice assistants in cars. It focuses on backend processing, AI model integration, and API-based communication. The system is designed to be extendable, allowing future integration with navigation systems, vehicle controls, and external APIs. The project also explores multilingual and conversational AI capabilities within defined constraints.

Agile methodology:

The project development followed the Agile methodology, emphasizing iterative development and continuous improvement. The work was divided into small development cycles, each focusing on a specific feature such as speech recognition, language model integration, or response handling. Regular testing and mentor feedback played a crucial role in refining system performance. This approach allowed flexibility in adapting to challenges and ensured that the project evolved efficiently toward its objectives.

Requirement Analysis:

Functional Requirements

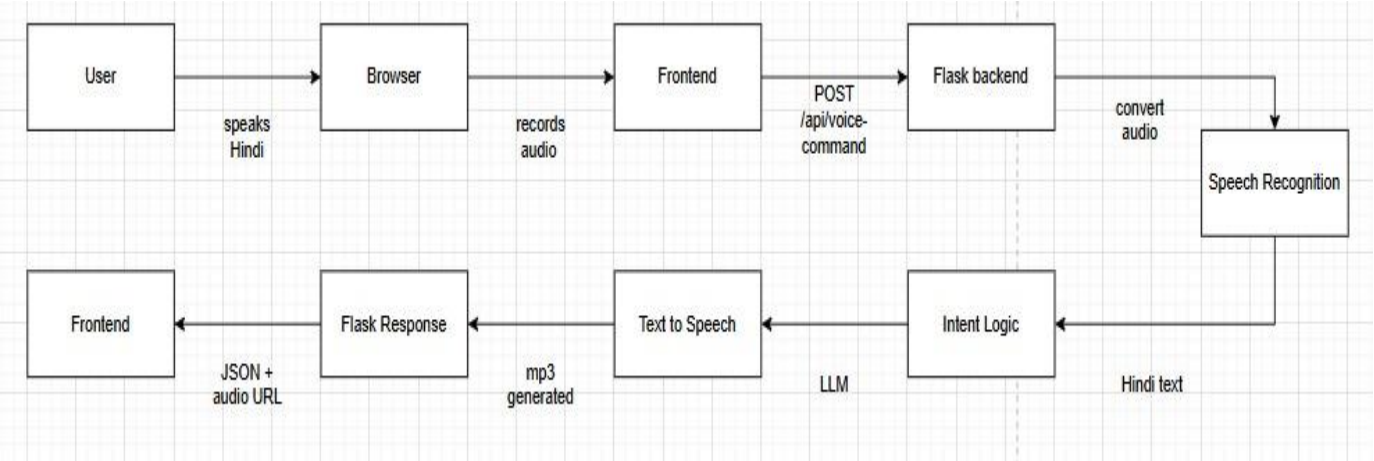
- Capture and process user voice input
- Convert audio input into text using speech recognition

- Interpret user intent using an AI language model
- Generate context-aware textual responses
- Convert text responses into speech output
- Handle API requests and responses efficiently

Non-Functional Requirements

- Low response latency for real-time interaction
- High system reliability and fault tolerance
- Compatibility with CPU-based environments
- Modular and maintainable code structure
- Secure handling of data and API endpoints

System Design:



The above diagram represents the end-to-end working architecture of AutoVaani, a Hindi voice-based AI assistant. The system enables a user to interact with the application using spoken Hindi commands, processes the audio using AI and NLP techniques on the backend, and returns an intelligent spoken response.

The architecture follows a client–server model, where the frontend handles user interaction and audio capture, while the Flask backend manages speech recognition, intent processing using an LLM, and text-to-speech conversion.

1. User Interaction (Voice Input)
    - The process begins when the user speaks a command in Hindi.
    - This could be a query such as navigation assistance, system control, or general information.
    - The interaction is completely voice-driven, making the system hands-free and user-friendly.
2. Browser (Audio Capture)
    - The browser acts as the first interface between the user and the system.

- Using browser-based APIs (such as Web Audio or MediaRecorder), the user’s voice is recorded in real time.
- The recorded audio is temporarily stored and prepared for transmission to the frontend application.

3. Frontend (Client Application)
    - The frontend receives the recorded audio from the browser.
    - It acts as a communication bridge between the browser and the backend.
    - The audio file is sent to the backend using an HTTP POST request
    - This request includes the recorded audio data for further processing.
4. Flask Backend (API Layer)
    - The Flask backend serves as the core processing unit of the system.
    - It receives the audio input from the frontend and coordinates all backend operations.
    - Flask ensures:
        - Request handling
        - Audio file management
        - Integration with AI models
        - Response generation in a structured JSON format
5. Speech Recognition (Speech-to-Text)
    - The received audio is forwarded to the Speech Recognition module.
    - This module converts the Hindi speech audio into Hindi text using Speech-to-Text (STT) techniques.
    - The output of this step is a clean textual representation of the user’s spoken command.
6. Intent Logic Processing

- The recognized Hindi text is passed to the Intent Logic module.
- This module is responsible for:
    - Understanding the user’s intent
    - Classifying the command (e.g., control, query, assistance)
    - Deciding how the system should respond
- It uses rule-based logic combined with AI reasoning.

7. LLM (Large Language Model)
    - The LLM (Large Language Model) processes the intent and context.
    - It generates a natural, human-like response in Hindi.
    - The LLM ensures:
        - Context awareness
        - Meaningful and accurate replies
        - Conversational tone suitable for a voice assistant
8. Text-to-Speech Conversion
    - The generated Hindi text response is sent to the Text-to-Speech (TTS) module.
    - This module converts the text into spoken audio.
    - The output is an MP3 audio file, making the response audible to the user.
9. Flask Response Handling
    - Once the MP3 file is generated, the Flask backend prepares the final response.
    - The backend sends:
        - A JSON response
        - An audio file URL pointing to the generated speech
    - This structured response ensures easy handling on the frontend.
10. Frontend Playback (Voice Output)

- The frontend receives the JSON response from Flask.
- It extracts the audio URL and plays the MP3 file automatically.
- The user hears the AI assistant's spoken response in Hindi, completing the interaction cycle.

Codes:

app.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import speech_recognition as sr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import os
import tempfile
import subprocess

# ---------------- Flask Setup ----------------

app = Flask(__name__)
CORS(app)

# ---------------- Model Setup ----------------

print("\nLoading Hindi car assistant model...", flush=True)

MODEL_PATH = "Qwen/Qwen2.5-0.5B-Instruct"

# ✅ Force CPU for stability
device = "cpu"
dtype = torch.float32

print(f"Using device: {device}", flush=True)

tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)

model = AutoModelForCausalLM.from_pretrained(
```

```
    MODEL_PATH,
    torch_dtype=dtype
).to(device)

pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    device=-1
)

alpaca_prompt_template = """You are a helpful car voice assistant.
Reply briefly in Hindi.

User command:
{}

Assistant response:
"""

# ---------------- Helpers ----------------

def ai_car_response(user_command: str) -> str:
    prompt = alpaca_prompt_template.format(user_command)

    outputs = pipe(
        prompt,
        max_new_tokens=80,
        do_sample=True,
        temperature=0.7,
        top_p=0.9
    )

    text = outputs[0]["generated_text"]
    return text.split("Assistant response:")[-1].strip()


def car_action_logic(user_command: str):
    if "गर्मी" in user_command:
        return "ठीक है, मैं एसी चालू कर रहा हूँ।"
    if "ठंड" in user_command:
        return "ठीक है, मैं एसी बंद कर रहा हूँ।"
    if "खिड़की" in user_command and "खोल" in user_command:
        return "ठीक है, मैं खिड़की खोल रहा हूँ।"
    if "खिड़की" in user_command and "बंद" in user_command:
        return "ठीक है, मैं खिड़की बंद कर रहा हूँ।"
    return None
```

```python
# ---------------- API ----------------

@app.route("/api/voice-command", methods=["POST"])
def handle_voice_command():
    print("\n📥 Request received", flush=True)

    webm_path = None
    wav_path = None

    try:
        if "audio" not in request.files:
            return jsonify({"error": "No audio file provided"}), 400

        audio_file = request.files["audio"]
        recognizer = sr.Recognizer()

        # Save WebM audio
        with tempfile.NamedTemporaryFile(delete=False, suffix=".webm") as tmp:
            audio_file.save(tmp.name)
            webm_path = tmp.name

        wav_path = webm_path.replace(".webm", ".wav")

        print("🔄 Converting audio to WAV", flush=True)

        # Convert WebM → WAV (clean + louder)
        subprocess.run(
            [
                "ffmpeg", "-y",
                "-i", webm_path,
                "-ac", "1",
                "-ar", "16000",
                "-af", "silenceremove=1:0:-50dB,volume=2",
                wav_path
            ],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL,
            check=True
        )

        # Speech Recognition
        with sr.AudioFile(wav_path) as source:
            recognizer.adjust_for_ambient_noise(source, duration=0.6)
            audio_data = recognizer.record(source)

        print("🟢 Calling Google Speech API", flush=True)

        user_command_text = recognizer.recognize_google(
```

```
            audio_data, language="hi-IN"
        )

        print("🧠 User said:", user_command_text, flush=True)

        # Decide response
        rule_response = car_action_logic(user_command_text)

        if rule_response:
            car_response = rule_response
            print("⚙️ Rule-based response used", flush=True)
        else:
            print("🤖 Using LLM fallback", flush=True)
            car_response = ai_car_response(user_command_text)

        return jsonify({
            "user_command": user_command_text,
            "car_response": car_response
        })

    except sr.UnknownValueError:
        return jsonify({
            "error": "मैं समझ नहीं पाया। कृपया धीरे और साफ़ बोलें।"
        }), 400

    except sr.RequestError as e:
        print("❌ Google Speech API error:", e, flush=True)
        return jsonify({
            "error": "Speech service unavailable"
        }), 503

    except Exception as e:
        print("🔥 ERROR:", repr(e), flush=True)
        return jsonify({"error": str(e)}), 500

    finally:
        for f in [webm_path, wav_path]:
            if f and os.path.exists(f):
                os.remove(f)
        print("🧹 Temp files cleaned", flush=True)


# ---------------- Run ----------------

if __name__ == "__main__":
    app.run(
        host="0.0.0.0",
        port=5001,
        debug=False,
```

```
        use_reloader=False
    )
```

Index.html

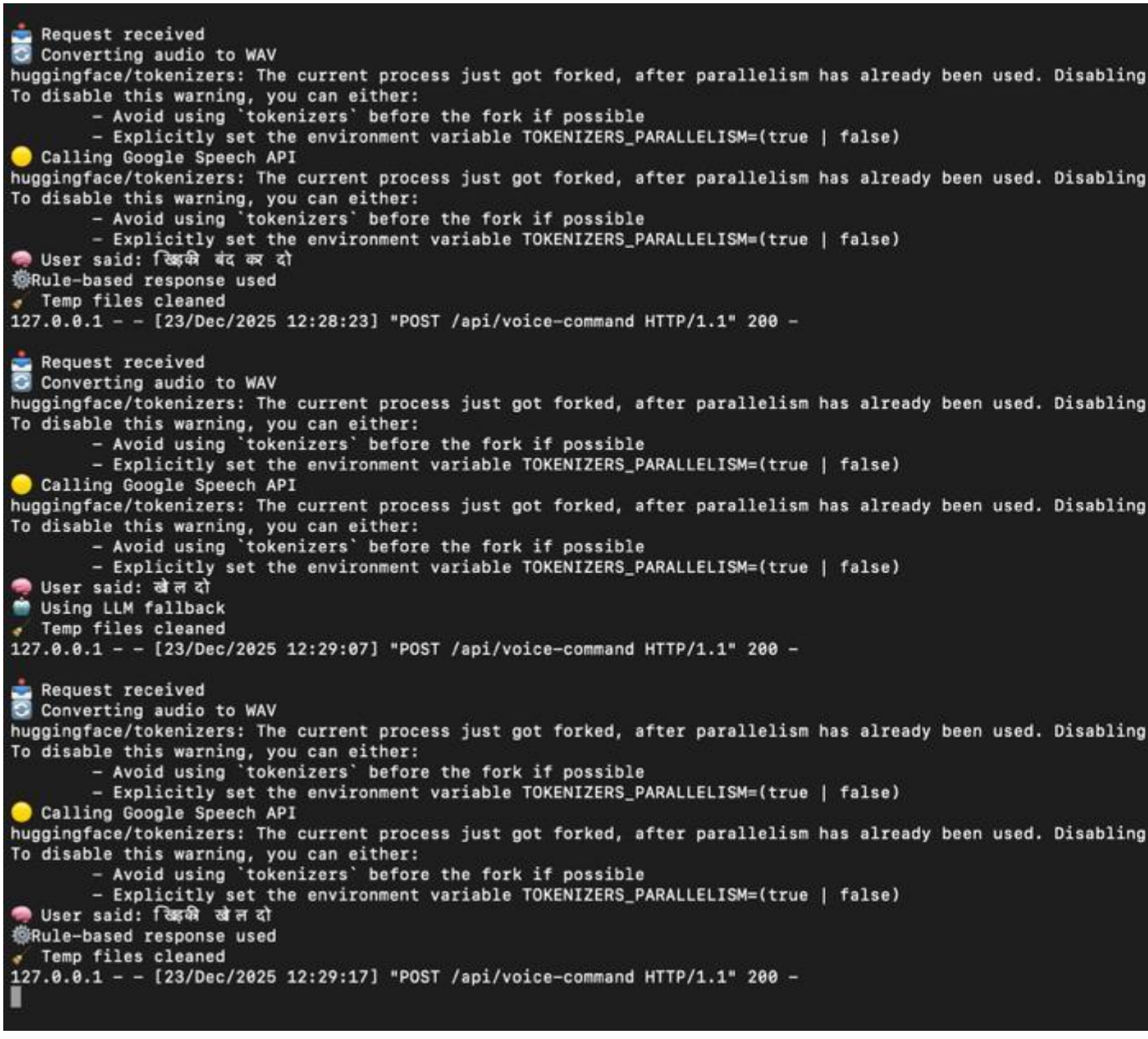
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hindi Car Voice Assistant</title>

    <style>
        body {
            font-family: Arial, sans-serif;
            background: #111;
            color: #fff;
            text-align: center;
            padding: 40px;
        }

        button {
            padding: 15px 25px;
            font-size: 18px;
            cursor: pointer;
            border: none;
            border-radius: 8px;
            background: #00c853;
            color: white;
        }

        button:disabled {
            background: #555;
        }

        .box {
            margin-top: 20px;
            padding: 15px;
            background: #222;
            border-radius: 8px;
            text-align: left;
            max-width: 600px;
            margin: auto;
        }

        .label {
            color: #00e5ff;
```

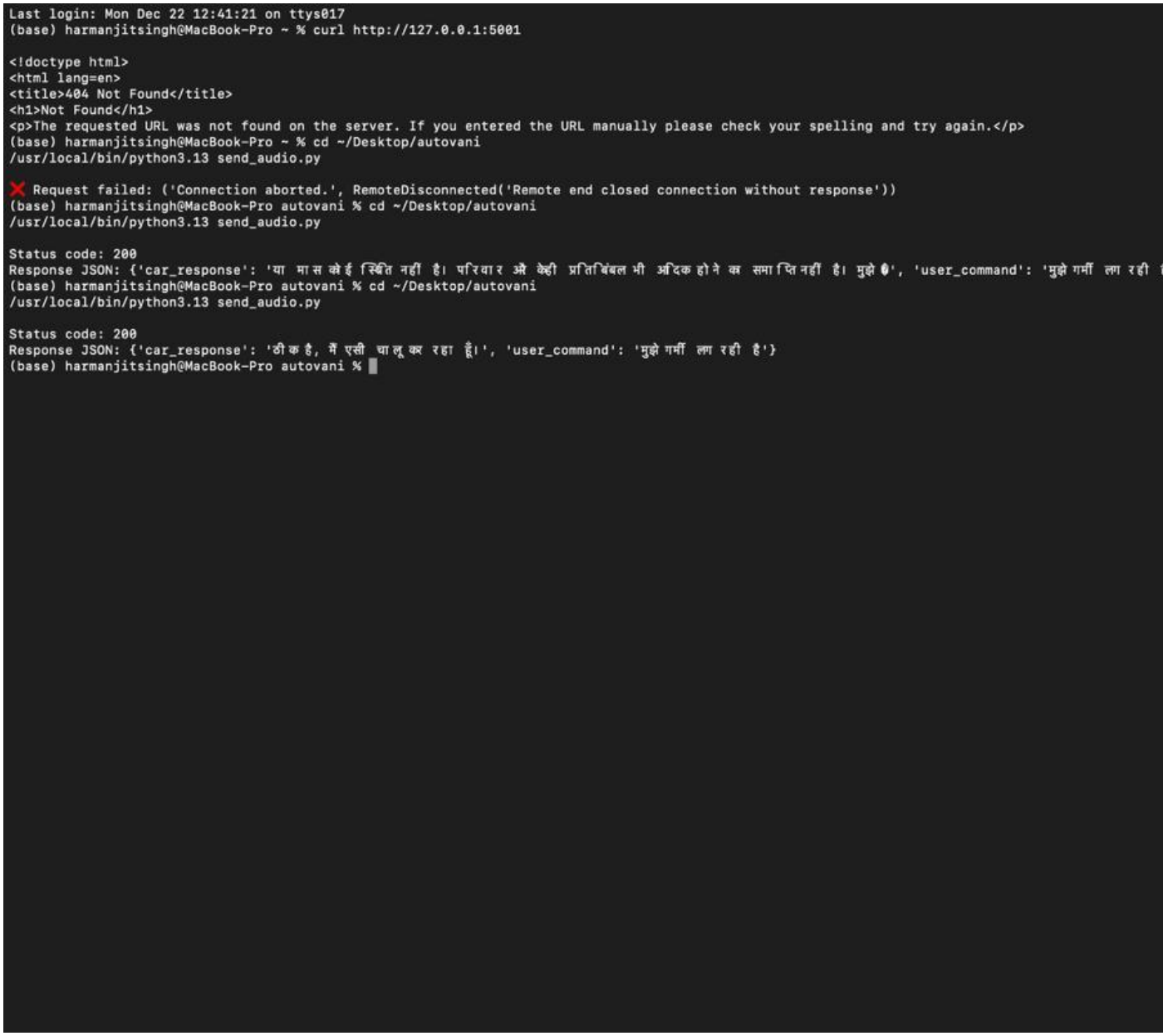
```
            font-weight: bold;
        }

        .hint {
            margin-top: 10px;
            color: #aaa;
            font-size: 14px;
        }
    </style>
</head>

<body>

<h1>🚗 Hindi Car Voice Assistant</h1>

<button id="recordBtn">🎙️ Start Speaking</button>
<p class="hint">Click → wait 1 second → speak slowly and clearly in Hindi</p>

<div class="box">
    <p><span class="label">You said:</span> <span id="userText">---</span></p>
    <p><span class="label">Car response:</span> <span id="carText">---
</span></p>
</div>

<audio id="carAudio" autoplay></audio>

<script>
let mediaRecorder;
let audioChunks = [];
let stream;

const recordBtn = document.getElementById("recordBtn");
const carAudio = document.getElementById("carAudio");

recordBtn.onclick = async () => {
    audioChunks = [];
    recordBtn.disabled = true;
    recordBtn.innerText = "🎙️ Listening...";

    // 🎤 Request microphone with proper constraints
    stream = await navigator.mediaDevices.getUserMedia({
        audio: {
            echoCancellation: true,
            noiseSuppression: true,
            autoGainControl: true
        }
    });
```

```
    // ✅ Force a stable mime type
    mediaRecorder = new MediaRecorder(stream, {
        mimeType: "audio/webm;codecs=opus"
    });

    mediaRecorder.start();

    mediaRecorder.ondataavailable = e => {
        if (e.data.size > 0) audioChunks.push(e.data);
    };

    // ✅ Give user enough time to speak
    setTimeout(() => {
        mediaRecorder.stop();
        recordBtn.innerText = "⏳ Processing...";
    }, 7000);

    mediaRecorder.onstop = async () => {
        // 🛑 Stop mic immediately
        stream.getTracks().forEach(track => track.stop());

        const audioBlob = new Blob(audioChunks, { type: "audio/webm" });

        const formData = new FormData();
        formData.append("audio", audioBlob, "voice.webm");

        try {
            const res = await fetch("http://127.0.0.1:5001/api/voice-command",
{
                method: "POST",
                body: formData
            });

            const data = await res.json();

            if (data.error) {
                alert(data.error);
            } else {
                document.getElementById("userText").innerText =
                    data.user_command || "—";

                document.getElementById("carText").innerText =
                    data.car_response || "—";

                // 🔊 Play car response audio (if available)
                if (data.audio_url) {
                    carAudio.src = "http://127.0.0.1:5001" + data.audio_url;
                    carAudio.play();
```

```
                }
            }

        } catch (err) {
            alert("Backend not reachable");
        }

        recordBtn.disabled = false;
        recordBtn.innerText = "🎙️ Start Speaking";
    };
};
</script>

</body>
</html>
```

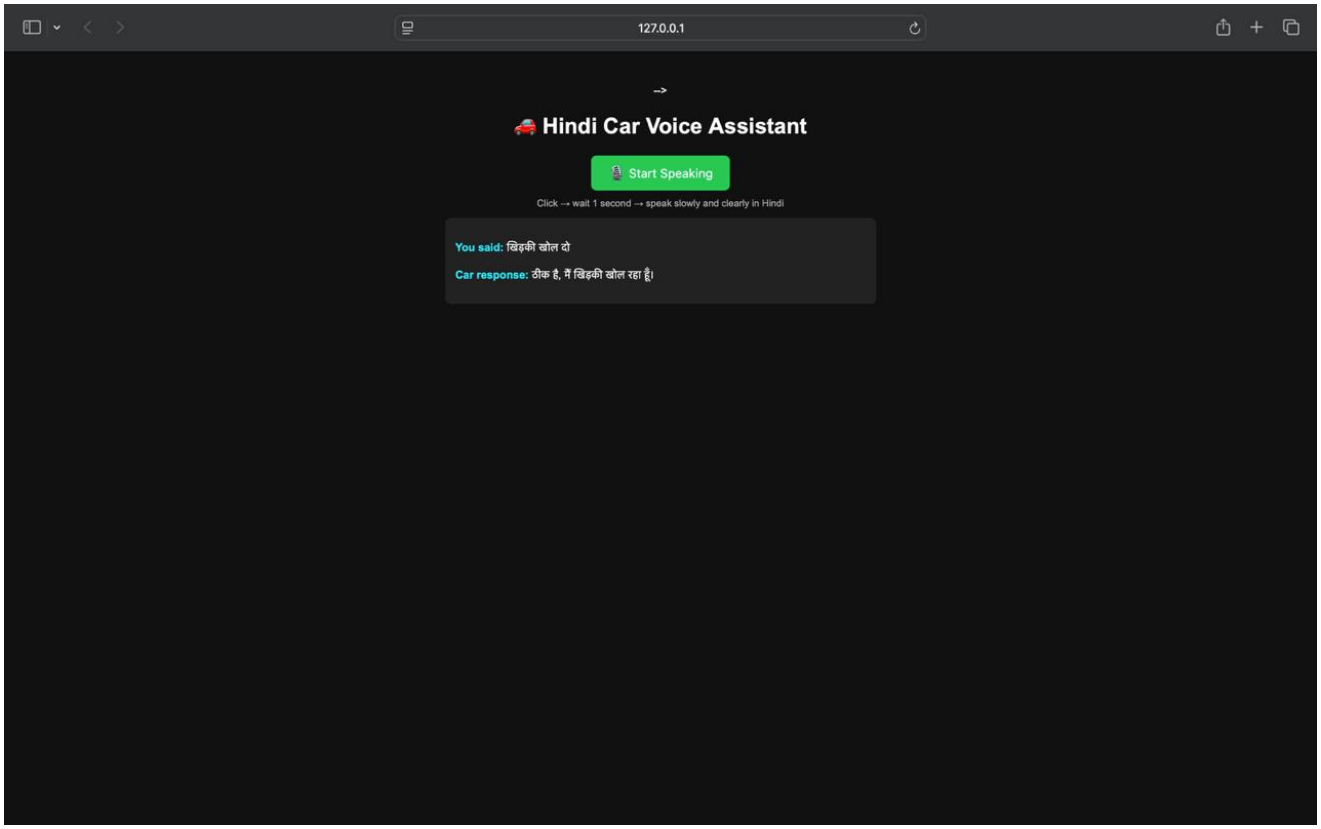
Implementation:

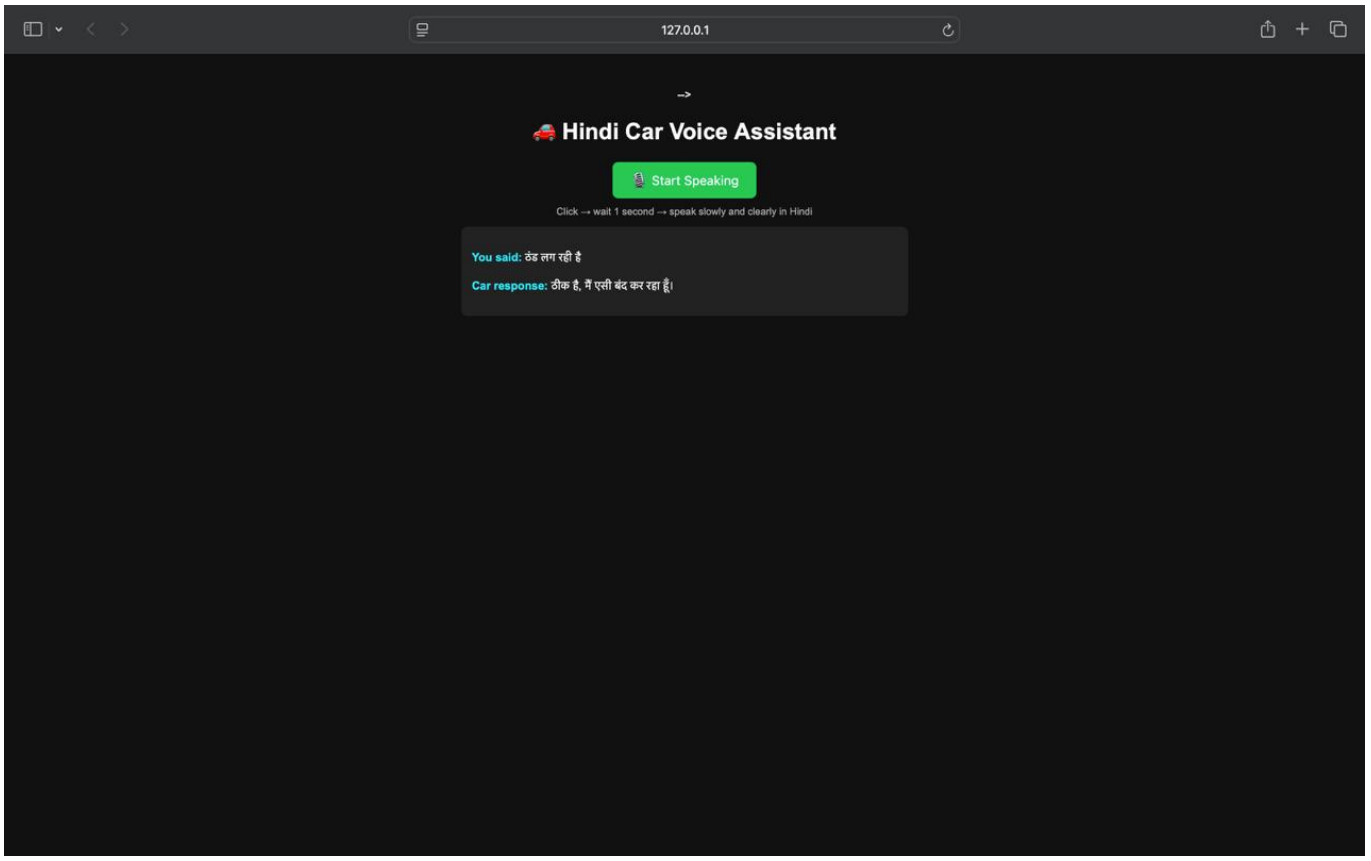


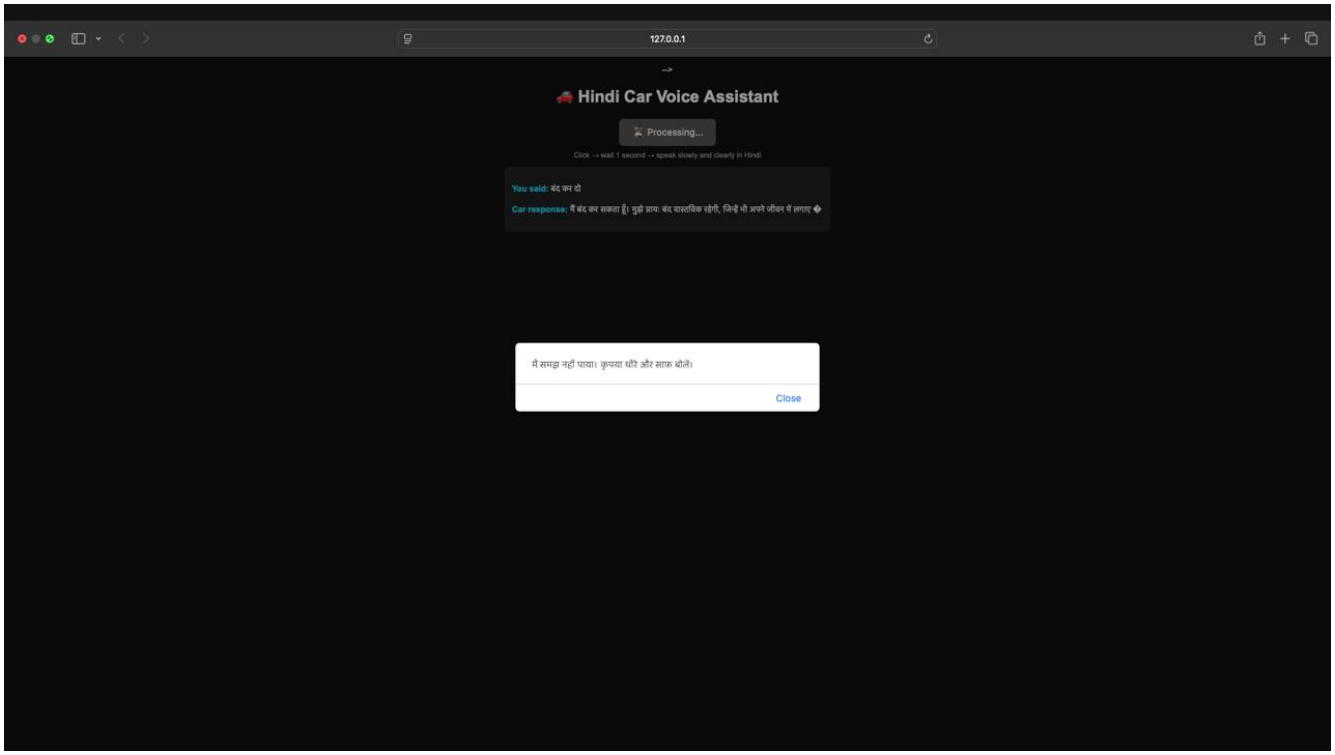

Last login: Mon Dec 22 12:41:21 on ttys017
(base) harmanjitsingh@MacBook-Pro ~ % curl http://127.0.0.1:5001
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.
(base) harmanjitsingh@MacBook-Pro ~ % cd ~/Desktop/autovani
/usr/local/bin/python3.13 send_audio.py
Request failed: ('Connection aborted.', RemoteDisconnected('Remote end closed connection without response'))
(base) harmanjitsingh@MacBook-Pro autovani % cd ~/Desktop/autovani
/usr/local/bin/python3.13 send_audio.py
Status code: 200
Response JSON: {'car_response': 'या मा स के ई स्थिति नहीं है। परिवार औ केही प्रतिबंबल भी अधिक हो ने क समा प्ति नहीं है। मुझे', 'user_command': 'मुझे गर्मी लग रही
(base) harmanjitsingh@MacBook-Pro autovani % cd ~/Desktop/autovani
/usr/local/bin/python3.13 send_audio.py
Status code: 200
Response JSON: {'car_response': 'ठीक है, मैं एसी चालू कर रहा हूँ।', 'user_command': 'मुझे गर्मी लग रही है'}
(base) harmanjitsingh@MacBook-Pro autovani %

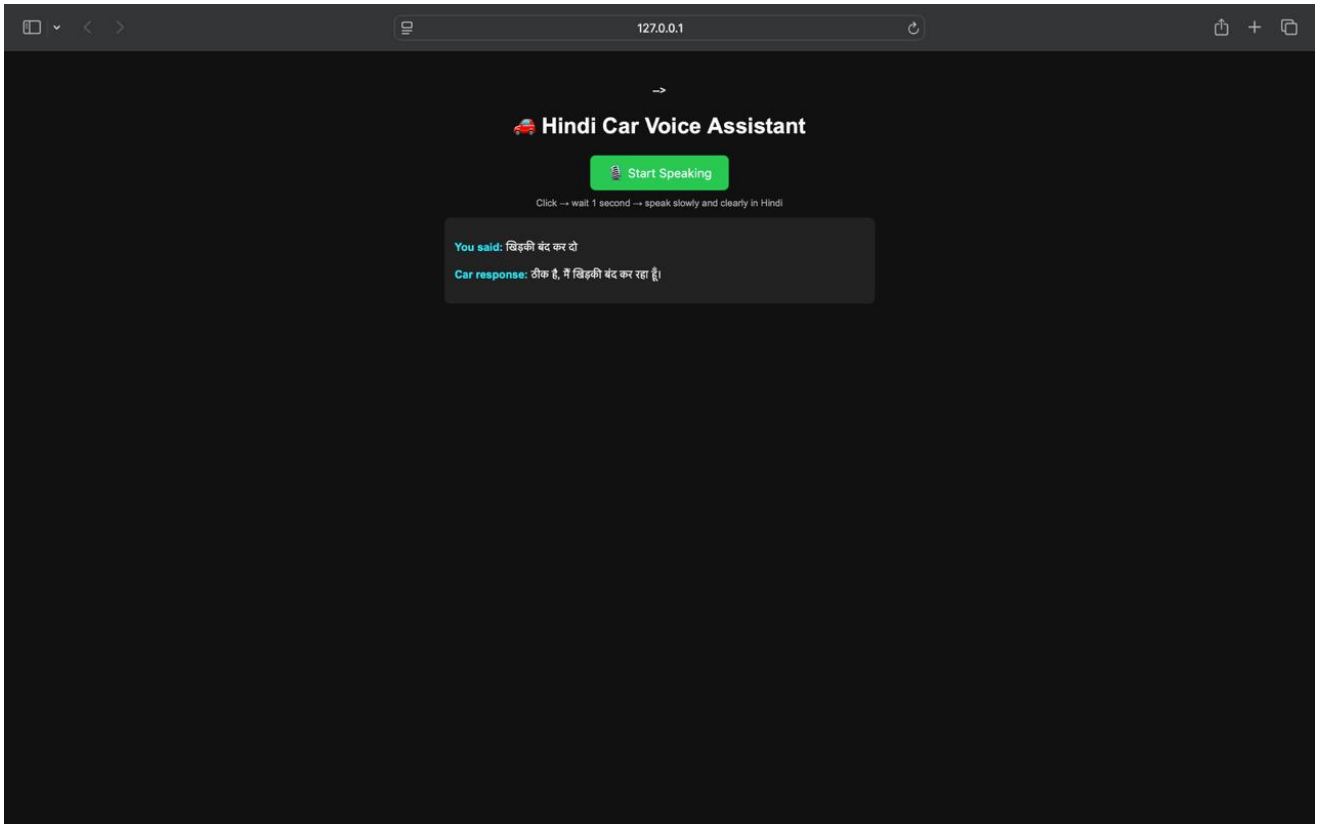
```
^C%                                                    (base) harmanjitsingh@MacBook-
/usr/local/bin/python3.13 app2.py

Skipping import of cpp extensions due to incompatible torch version 2.6.0 for torchao version 0.14.1

Loading Hindi car assistant model...
Using device: cpu
`torch_dtype` is deprecated! Use `dtype` instead!
Device set to use cpu
 * Serving Flask app 'app2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.1.5:5001
Press CTRL+C to quit

📩 Request received
🔄 Converting audio to WAV
🟡 Calling Google Speech API
🧠 User said: मुझे गर्मी लग रही है
⚙️Rule-based response used
🧹 Temp files cleaned
127.0.0.1 - - [23/Dec/2025 12:14:59] "POST /api/voice-command HTTP/1.1" 200 -

📩 Request received
🔄 Converting audio to WAV
🟡 Calling Google Speech API
🧠 User said: ठंड लग रही है
⚙️Rule-based response used
🧹 Temp files cleaned
127.0.0.1 - - [23/Dec/2025 12:16:43] "POST /api/voice-command HTTP/1.1" 200 -
```

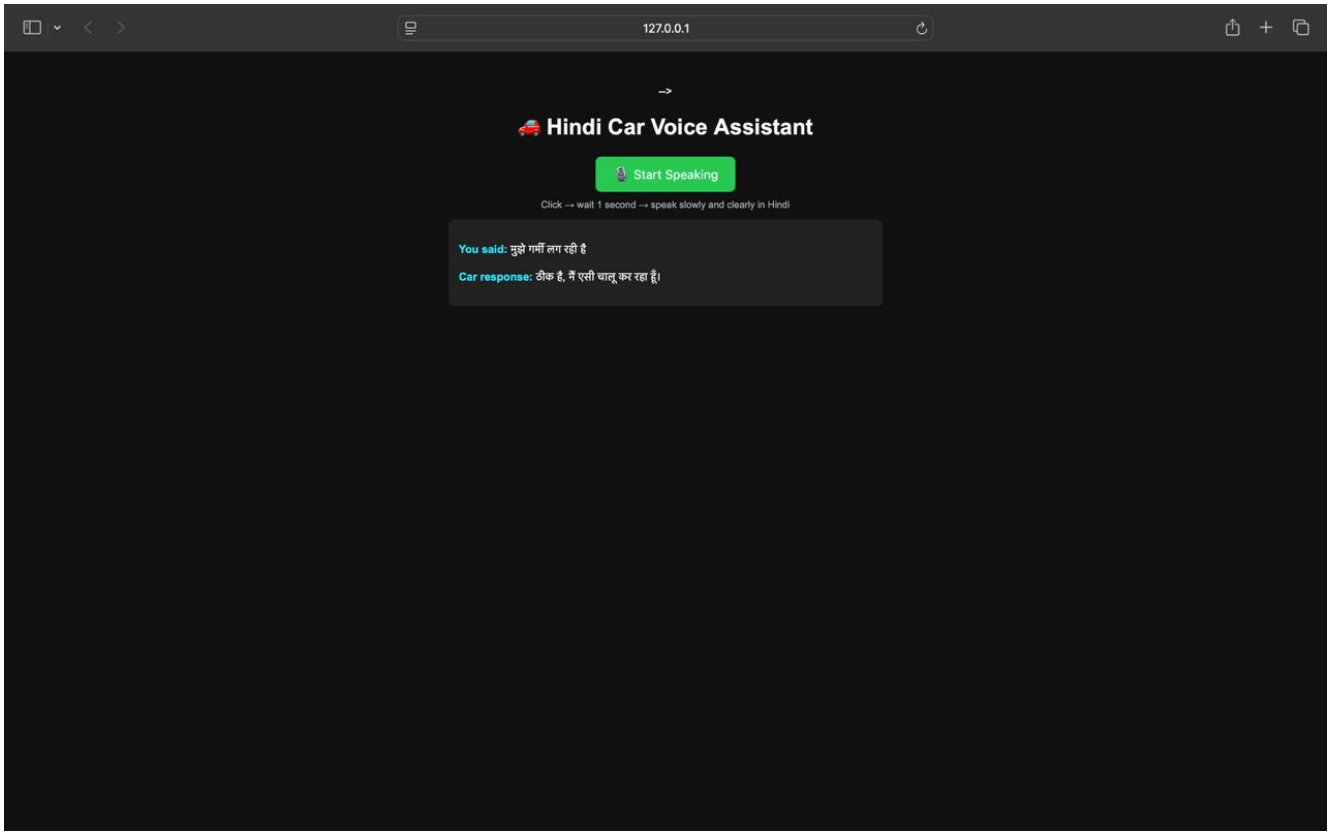
```
^C%                                                    (base) harmanjitsingh@Mac
/usr/local/bin/python3.13 app2.py

Skipping import of cpp extensions due to incompatible torch version 2.6.0 for torchao version 0.14.1

Loading Hindi car assistant model...
Using device: cpu
`torch_dtype` is deprecated! Use `dtype` instead!
Device set to use cpu
 * Serving Flask app 'app2'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI se
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.1.5:5001
Press CTRL+C to quit

📩 Request received
🔄 Converting audio to WAV
🟡 Calling Google Speech API
🧠 User said: मुझे गर्मी लग रही है
⚙️Rule-based response used
🧹 Temp files cleaned
127.0.0.1 - - [23/Dec/2025 12:14:59] "POST /api/voice-command HTTP/1.1" 200 -
```


127.0.0.1
🚗 Hindi Car Voice Assistant
Start Speaking
Click → wait 1 second → speak slowly and clearly in Hindi
You said: खिड़की खोल दो
Car response: ठीक है, मैं खिड़की खोल रहा हूँ।


127.0.0.1
🚗 Hindi Car Voice Assistant
Start Speaking
Click → wait 1 second → speak slowly and clearly in Hindi
You said: ठंड लग रही है
Car response: ठीक है, मैं एसी बंद कर रहा हूँ।


127.0.0.1
Hindi Car Voice Assistant
Processing...
Click → wait 1 second → speak slowly and clearly in Hindi
You said: बंद कर दो
Car response: मैं बंद कर सकता हूँ। पुछो प्रायः बंद वास्तविक रहेगी, जिन्हें भी अपने जीवन में लगाए
मैं समझ नहीं पाया। कृपया धीरे और साफ बोलें।
Close


127.0.0.1
Hindi Car Voice Assistant
Start Speaking
Click → wait 1 second → speak slowly and clearly in Hindi
You said: खिड़की बंद कर दो
Car response: ठीक है, मैं खिड़की बंद कर रहा हूँ।



The implementation of AutoVaani is carried out using Python due to its strong ecosystem for AI and backend development. Flask is used to develop RESTful APIs that handle incoming audio data and return responses. The SpeechRecognition library processes real-time voice input, while the Hugging Face Transformers library loads and executes the Qwen instruction-tuned language model. The model is configured to run on CPU to ensure stability and broader compatibility. Proper exception handling, temporary file management, and resource optimization techniques are implemented to enhance system performance.

Tools & Technologies:

- Python: Core programming language
- Flask: Backend web framework
- Hugging Face Transformers: Language model integration
- PyTorch: Model execution and tensor operations
- SpeechRecognition: Speech-to-text conversion
- Flask-CORS: Cross-origin request handling
- Git & GitHub: Version control and collaboration
- VS Code / Google Colab: Development environment

Testing:

Testing was conducted at various levels to ensure accuracy and reliability. Unit testing focused on validating individual modules such as speech recognition and text generation. Integration testing ensured smooth data flow between system components. Real-world testing involved multiple voice inputs with varying speech patterns to evaluate performance. The system was also tested for edge cases such as unclear audio input, silence, and network interruptions.

Deployment:

The AutoVaani backend is deployed as a Flask application that can run locally or on cloud-based servers. The language model is loaded during application startup to minimize runtime delays. The application supports cross-origin requests, enabling seamless integration with frontend interfaces. Deployment considerations include performance optimization, ease of setup, and scalability.

Maintenance:

Maintenance of AutoVaani is simplified due to its modular architecture. Updates to individual components such as the AI model or speech engine can be performed without affecting the entire system. Regular monitoring and logging help identify performance issues. Maintenance activities also include dependency updates, performance tuning, and system security checks.

Advantages:

- Enables hands-free vehicle interaction
- Improves driving safety by reducing distractions
- Provides natural and conversational responses
- Efficient CPU-based execution
- Scalable and modular system design

Limitations:

- Accuracy depends on speech clarity and background noise
- CPU-based inference may introduce latency
- Requires internet access for model loading
- Limited to software-level vehicle interaction

Future Enhancements:

- Integration with navigation and vehicle control systems
- Support for additional languages and accents
- Offline AI model optimization
- User personalization and voice profiles
- Integration with IoT and smart vehicle ecosystems

Conclusion:

AutoVaani demonstrates the effective application of artificial intelligence in the automotive domain by delivering an intelligent and conversational voice assistant. The project successfully integrates speech processing and transformer-based language models into a scalable backend system. It highlights the feasibility of deploying AI-powered solutions even in resource-constrained environments. Overall, this project has provided valuable hands-on experience in AI development, system design, and real-world problem solving.

References:

- Hugging Face Transformers Documentation
- Flask Official Documentation
- PyTorch Documentation
- SpeechRecognition Library Documentation
- Research articles on AI-based automotive voice assistants