**Project Report: Bhaiya Bot**

**By:** Tanya Tarun & Kunal Kumar Singh

**Date:** August 6, 2025

**1. Introduction**

**1.1. Project Overview**

Bhaiya Bot is an intelligent, web-based AI voice tutor designed to provide real-time, personalized academic assistance. It leverages a powerful Large Language Model (LLM) to answer student queries across a vast spectrum of academic levels, from nursery to the highly competitive JEE/NEET preparation. The application features a dual-input system (voice and text) and delivers spoken answers, creating an intuitive and conversational learning environment.

**1.2. Inspiration & Problem Statement**

The inspiration for this project is deeply personal, stemming from observing my brother's rigorous preparation for the NEET medical entrance exam. He, like millions of other students, often encountered specific doubts outside of coaching hours. The available resources provided generic, textbook definitions, whereas he needed a nuanced explanation tailored to his level of understanding. This highlighted a critical gap in the education ecosystem: the lack of an instantly accessible, adaptive, and patient tutor. Bhaiya Bot was conceived to fill this gap, acting as a virtual "Bhaiya" (older brother) who is always available to clear doubts in a way that promotes true understanding over rote memorization.

**2. Objectives**

The primary objectives for the Bhaiya Bot project were:

* To create a seamless, voice-first user interface that makes asking questions natural and effortless.
* To develop a system where the AI's responses are dynamically tailored to the user's selected academic level.
* To build a full-stack application with a scalable Python backend and a visually impressive, modern frontend.
* To deploy the application on the cloud, making it publicly accessible and demonstrating a complete development-to-production workflow.

**3. System Architecture**

Bhaiya Bot is built on a classic client-server architecture.

**3.1. Backend Architecture (Flask Server)**

The backend is a REST API built with the **Flask** framework in Python.

* **app.py**: The main application file that defines the API endpoints (/api/ask) and serves the frontend.
* **llm\_handler.py**: This module contains the core logic for interacting with the Google Gemini API. It takes the user's query and selected level, formats it using a template, sends it to the AI, and processes the response.
* **prompt\_templates.py**: This file is crucial for the bot's adaptability. It stores a dictionary of "personas" that instruct the AI on how to behave (e.g., as a patient teacher for a 1st grader or an expert professor for a JEE aspirant).
* **Google Gemini API**: The gemini-1.5-flash model is used as the AI brain, chosen for its optimal balance of speed and response quality.

**3.2. Frontend Architecture (Web Interface)**

The frontend is a single-page application built with standard web technologies.

* **HTML**: Provides the fundamental structure of the page.
* **Tailwind CSS**: A utility-first CSS framework used to rapidly build the professional, responsive, and visually appealing dark-themed UI with "glassmorphism" effects.
* **JavaScript (Vanilla)**: Manages all client-side logic, including DOM manipulation, event handling, and state management.
* **Web Speech API**: This browser-native API is the cornerstone of the voice interaction, with SpeechRecognition for speech-to-text and SpeechSynthesis for text-to-speech.
* **Lottie**: A library used to render the fluid and engaging robot avatar animation.

**3.3. Data Flow**

1. User selects an academic level on the frontend.
2. User asks a question via voice or text.
3. JavaScript captures the question and the selected level.
4. A POST request is sent to the backend's /api/ask endpoint with the question and level in a JSON payload.
5. The Flask server receives the request and passes it to the llm\_handler.
6. The llm\_handler selects the appropriate prompt, formats the query, and sends it to the Google Gemini API.
7. The Gemini API generates a response and sends it back to the Flask server.
8. The Flask server sends the answer back to the frontend.
9. JavaScript displays the answer on the screen and uses the Web Speech API to speak it aloud.

**4. Challenges Faced & Solutions**

The development process during this hackathon was a valuable lesson in real-world problem-solving.

1. **API Rate Limiting:** The most significant hurdle was hitting the daily free-tier quota on the Google Gemini API due to extensive testing. This caused the live application to fail.
   * **Solution:** We quickly diagnosed the 429 Quota Exceeded error by checking the Render logs. The immediate fix was to generate a new API key in a new Google Cloud project, which provided a fresh daily quota, and update it in the Render environment variables.
2. **Cross-Origin Resource Sharing (CORS):** Initially, the frontend and backend were separate, leading to the infamous Failed to fetch error. The browser's security policy was blocking communication.
   * **Solution:** After trying less effective methods, we re-architected the application. We modified the Flask server (app.py) to serve the index.html file itself, unifying the origin of the app and the API and completely bypassing any CORS issues.
3. **Deployment Environment Errors:** Our first deployment attempts failed due to environment-specific package issues (pywin32) and metadata generation errors.
   * **Solution:** We learned that a pip freeze from a Windows machine is not portable. We solved this by creating a clean, minimal requirements.txt file containing only the essential cross-platform libraries, which led to a successful deployment.

**5. Conclusion & Future Scope**

Bhaiya Bot successfully achieves the goal of creating a functional, impressive, and user-friendly AI voice tutor. It demonstrates a complete full-stack development cycle, from ideation and implementation to successfully navigating complex deployment challenges.

The project has immense potential for future expansion:

* **User Accounts:** To save chat history and track learning progress.
* **Subject Selection:** Allowing users to choose a specific subject for more focused answers.
* **Image Recognition (OCR):** Implementing a feature to analyze and solve problems from textbook images.
* **Fine-Tuning:** Training a custom model on specific educational datasets to further improve answer accuracy and relevance.
* **Custom Database**