1. Problem Definition

RespiraSense is a comprehensive web application designed to revolutionize the healthcare experience for patients and doctors. It includes two distinct modules—patient and doctor interfaces—that enable seamless interaction between users. The patient side offers a personalized dashboard, secure file storage for medical reports, appointment booking with healthcare professionals, and access to an integrated AI-driven chatbot, RespiraBot.

RespiraBot is a symptom-checking chatbot designed to assist users in identifying potential health conditions based on their described symptoms. The problem lies in training a model to reliably classify symptoms and suggest possible conditions while avoiding irrelevant or non-medical queries. This requires fine-tuning an AI model to accurately respond to healthcare-related inquiries while rejecting non-medical queries with appropriate responses.

The objective is to fine-tune a pre-trained language model to achieve a healthcare-specific assistant capable of providing accurate, consistent, and domain-relevant outputs.

2. Methodology

2.1 Frontend

The RespiraSense web application was developed using the MERN stack. React.js and TypeScript were used for building an interactive and responsive user interface. The backend was built with Node.js and Express.js, with Multer handling secure file uploads. MongoDB served as the database for storing user data, appointments, and file references. The app features robust user authentication and a doctor-patient appointment management system. The integrated RespiraBot chatbot, fine-tuned with custom healthcare data, provides an AI-driven symptom-checking experience.

2.2 Integration of the OpenAI API Key

The OpenAI API was integrated into RespiraBot using Node.js and TypeScript for the backend, with the openai npm package enabling seamless communication. This integration allowed RespiraBot to generate accurate and context-aware medical responses. However, the initial version sometimes provided answers to irrelevant queries, highlighting the need for fine-tuning to improve its focus and ensure it only responds to healthcare-related questions.

2.3 Dataset Procurement

The dataset was sourced from Kaggle, containing a list of diseases and associated symptoms. Each row of the dataset represents a disease, while columns indicate specific symptoms. A value of 1 indicates the symptom is present for the disease, and 0 indicates its absence. Link:
https://www.kaggle.com/datasets/dhivyeshrk/diseases-and-symptoms-dataset

2.4 Dataset Pre-processing

Step 1: Converting the Original Dataset into a Textual Format

The original dataset from Kaggle was preprocessed using a Python script to create a CSV file in a format suitable for training. This was done by loading the dataset into a DataFrame, iterating over each row to extract the disease name and related symptoms, and constructing user input and assistant response pairs. If symptoms were present, the input was formatted as "I am suffering from [symptoms]," and the response was framed as a suggested disease diagnosis. If no symptoms were present, a default response was generated to indicate that a diagnosis could not be made. The processed data was then saved into a CSV file named converted_dataset.csv.

Step 2: Converting the CSV File to .jsonl Format
The CSV file (converted_dataset.csv) was converted into .jsonl format to be compatible with OpenAI's fine-tuning requirements. This step was achieved using a Python script that read the CSV, iterated over each row, and formatted it into a JSON structure with messages representing user and assistant interactions. Each interaction was saved as a separate line in the output file, train.jsonl.

Step 3: Generating Non-Medical Query Data
A synthetic dataset of 100 non-medical queries was generated using ChatGPT to include examples of user questions unrelated to healthcare. The responses for these inputs were crafted to ensure the model only accepts medical-related questions, with a standard rejection message: "I can only assist with healthcare-related queries. Please ask a medical question." This step helped train the model to differentiate between medical and non-medical queries and respond appropriately.

Step 4: Combining and Finalizing the Dataset
The medical query dataset and the non-medical rejection dataset were merged to create the final dataset, totaling approximately 6,000 examples. This comprehensive dataset was prepared to train and fine-tune the chatbot, RespiraBot, enabling it to respond accurately to healthcare-related questions and reject irrelevant ones.
This was then divided into training and validation data with an 90/20 split favoring the training data. This was because fine-tuning process is managed to optimize the model based on the training data alone, without explicit validation steps.

2.5 Fine Tuning the Model

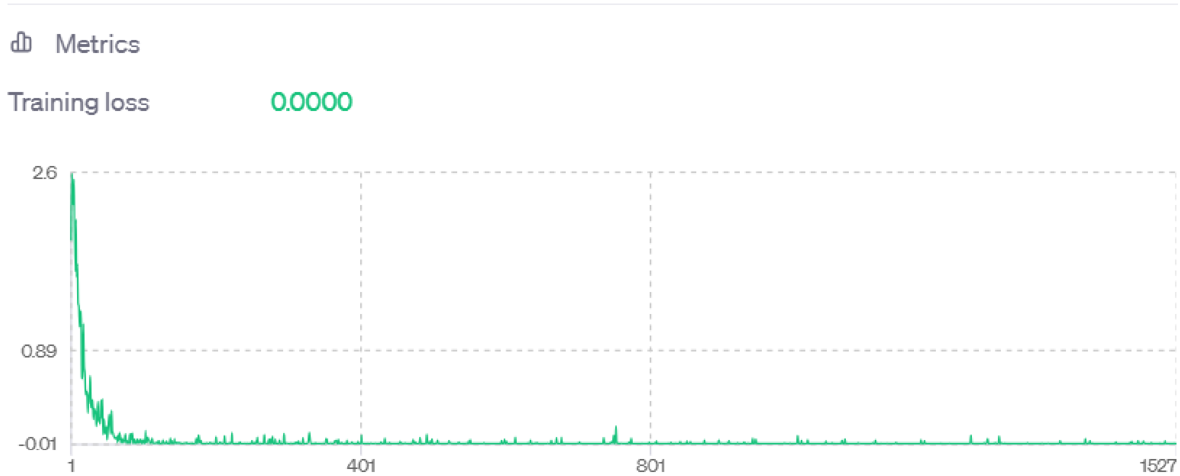
2.41 Uploading the Training Data
The dataset was prepared and converted into the .jsonl format, suitable for training. To fine-tune the model, the training data was uploaded to OpenAI using a Node.js script. The script utilized axios and form-data libraries to send an HTTP POST request to the OpenAI API's /files endpoint. The training file was specified with the purpose set to "fine-tune" to indicate its use for model fine-tuning.

2.42 Creating the Fine-Tuning Job
Once the training file was uploaded successfully, a fine-tuning job was created through the OpenAI API. The script made a POST request to the /fine_tuning/jobs endpoint, specifying the uploaded training file and the target model (gpt-4o-mini-2024-07-18). This step initiated the fine-tuning process, customizing the pre-trained model to focus on healthcare-related interactions and reject non-relevant queries. The job's progress was monitored and verified using the OpenAI dashboard's Playground to ensure the model's responses aligned with expectations.

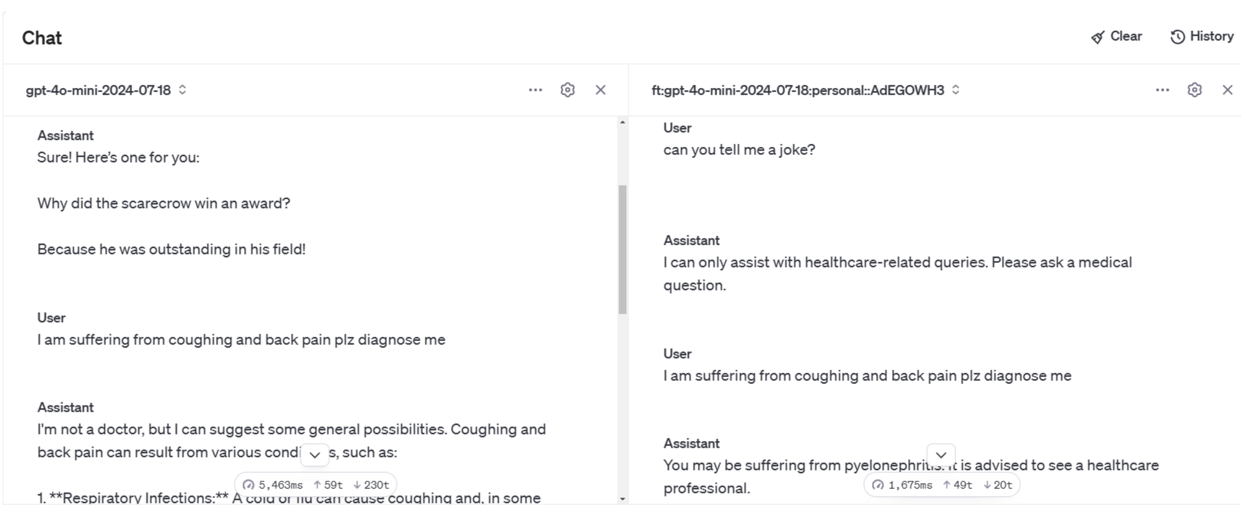
2.43 Integration of the Fine Tuned Model
The fine-tuned model was integrated into the RespiraSense MERN app by configuring the Node.js backend to interact with the OpenAI API. API endpoints were set up to send user input to the model, process the responses, and deliver accurate medical advice through the RespiraBot chatbot. This integration ensured that the app provided relevant healthcare guidance while filtering out non-medical queries, enhancing user interactions for both patients and doctors.

3. Results

The fine-tuning of the RespiraBot model was successful, using gpt-4o-mini-2024-07-18 as the base model. With 831,201 tokens trained over three epochs and a batch size of 11, the model achieved a training loss of 0.0000, showing effective learning. The final model, ft:gpt-4o-mini-2024-07-18:personal::AdEGOWH3, was tested and confirmed to deliver accurate medical responses and properly reject unrelated queries as in the image showing example comparison. This improved the chatbot's performance in the RespiraSense app, enhancing user interactions for both patients and doctors.



A comparison of the simple base model Gpt-4o-mini and the finetuned model are given below. These were obtained by running the prompts in the OpenAI fine tuned model 'Playground' environment:



4. Conclusion

The RespiraSense app, powered by a fine-tuned AI model, effectively combines accurate medical guidance with user-centric features such as file storage, dashboards, and appointment booking. This integration enhanced the chatbot's ability to provide reliable health information while managing irrelevant queries seamlessly.