
# Symptom Analyzer

Presented By :
Saurabh Lakhanpal, Evan Gowans, Melanie Liao

# Project Break Down


## Data Preparation & Resources

- Prepared structured data by generating synthetic patients based on the Disease
- Symptom Knowledge Base (DSKB). Ensured clinical realism by simulating partial symptom presence and adding controlled noise for variability.

## Model Development & Storage

- Built and trained a dense neural network using TensorFlow.
- Saved the model, scaler, and feature encodings to deploy predictions in real-time.

## Backend API Development

- Developed a Flask API connected to a PostgreSQL database.
- Serves symptoms, diseases, and model predictions through structured endpoints.

## Frontend Web Application

- Created an interactive web app using HTML, CSS, and JavaScript.
- Users can search symptoms, analyze results, and view top disease matches with probabilities.

# Project Objective

**Problem Statement**

Patients often turn to generic online searches for health concerns, leading to anxiety and misinformation.

There is a need for a reliable tool that can predict diseases based on symptom input, reducing uncertainty and improving user trust.

**Project Objective**

Develop a machine learning-powered web application that:

- Accepts symptoms from users.
- Predicts the top 3 most likely diseases with probability scores.
- Provides related symptoms to refine and improve predictions.
- Offers a more accurate alternative to broad, unreliable online symptom searches.

# Dataset Overview

Dataset:

- Disease Symptom Knowledge Base (DSKB)

Content:

- 149 diseases
- 407 symptoms
- 5,000 synthetic mock patient records

Format:

- CSV files
- Web Scraping
- Symptoms hotcoded into binary columns (1 = symptom present, 0 = absent)

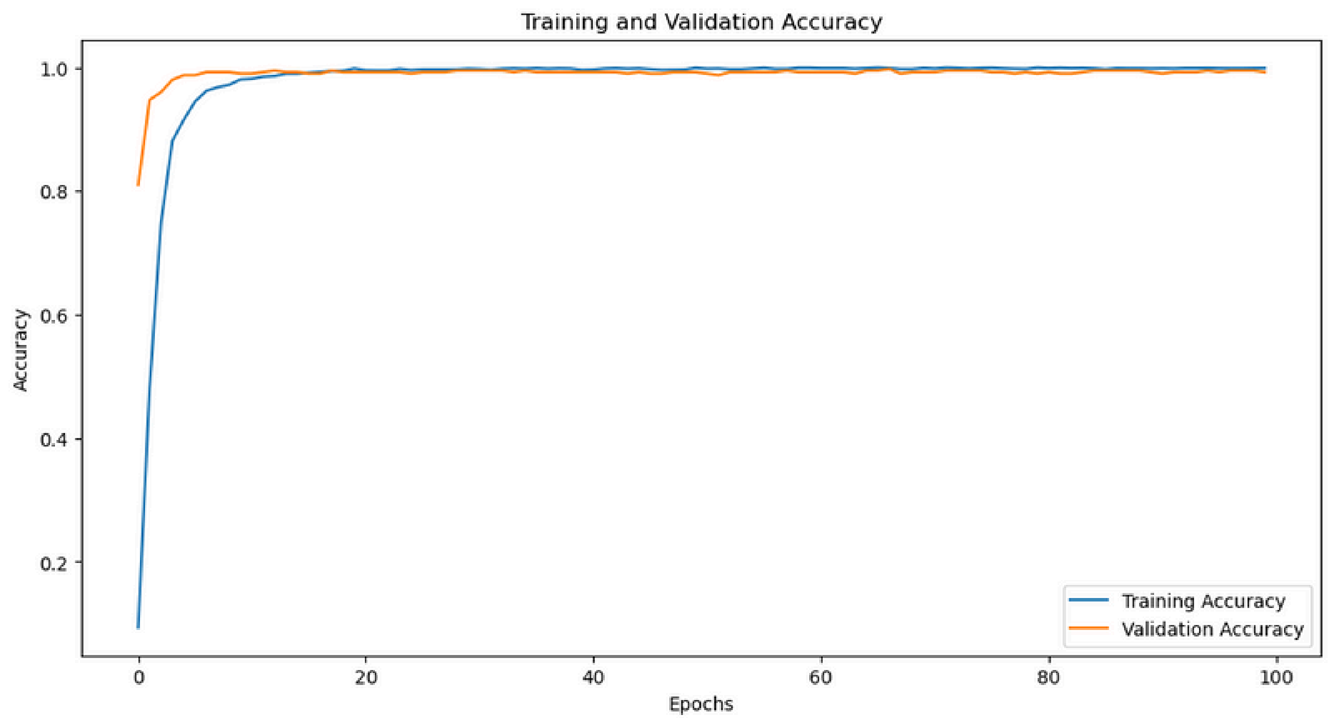
Why DSKB?

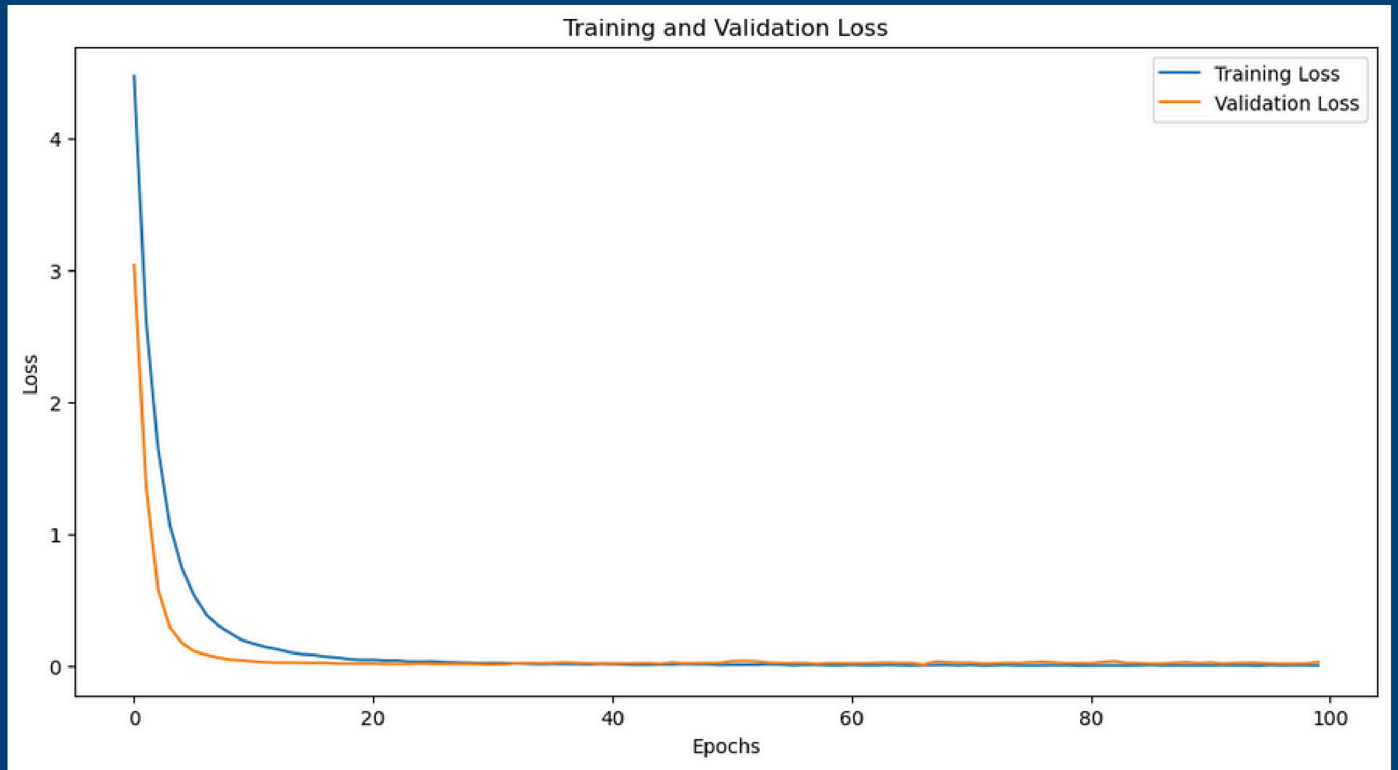
Provides structured, rich labeled data ideal for supervised machine learning.

```
Number of rows in synthetic dataset: 5000
```

```
Database disease_symptom_db dropped successfully.
Database disease_symptom_db created successfully.
SQL schema file executed successfully.
Data for diseases_tb uploaded successfully. Total rows: 133
Data for symptoms_tb uploaded successfully. Total rows: 406
Data for disease_symptom_tb uploaded successfully. Total rows: 1906
```

# Technologies & ML Architecture




**Technologies Used**

- Machine Learning: TensorFlow, Scikit-learn
- Data Handling: Pandas, NumPy
- Backend: Flask, SQLAlchemy, PostgreSQL
- Frontend: HTML, CSS, JavaScript
- Dev Tools: Google Colab, GitHub, AWS (planned deployment)

**Machine Learning Model Architecture**

- Model Type: Dense Neural Network (Multiclass Classification)
- Structure:
  - Input: 407 symptom features
  - 3 Hidden Layers (ReLU activation, Batch Normalization, Dropout)
  - Output Layer: Softmax activation (149 disease classes)
- Training:
  - Loss: Categorical Crossentropy
  - Optimizer: Adam
  - Data: 5,000 synthetic patient rows
- Model Performance:
  - Training Accuracy: 99.88%
  - Validation Accuracy: 99.25%
  - Training Loss: 0.0063
  - Validation Loss: 0.0315

# Synthetic Data Generation

**Objective**: Generate artificial data mimicking real-world patient charts for machine learning and testing.
**Outcome**: A 5,000-row synthetic dataset with diverse disease-symptom profiles, ready for algorithm training and testing.

## Step 1: Simulating Patients

- Created mock charts linking diseases to their symptoms.
- Ensured variability by assigning 33%+ of symptoms to each disease.

## Step 2: Introducing Noise

- Created mock charts linking diseases to their symptoms. Ensured variability by assigning 33%+ of symptoms to each disease.

## Step 3: Why Noise Matters

- Mimics variability in clinical data
- Builds robust models for real-world applications.

## Step 4: Real-World Alignment

- Reflects thresholds used in diagnostic frameworks (DSM-5: ~30–50% of symptoms; ACR for lupus: ~36–40%)

# Demo

## Features of the Web App

**Symptom Search:**
- Autocomplete search bar linked to a database of 407 symptoms.

**Disease Prediction:**
- Predicts the top 3 diseases with probability scores based on selected symptoms.

**Related Symptom Suggestions:**
- Suggests additional symptoms for refining predictions.

**Reanalyze and Refine:**
- Users can add new symptoms and instantly update their results.

**Lightweight Frontend:**
- Built with pure HTML, CSS, and JavaScript.
- Fast, responsive, and mobile-friendly design without heavy frameworks.

## Workflow Overview

**Step 1: Data Preparation (ETL)**
- Cleaned and loaded disease-symptom data into a PostgreSQL database.

**Step 2: Synthetic Patient Generation**
- Created 5,000 mock patients with ≥33% disease symptoms and controlled noise.

**Step 3: Model Training**
- Trained a dense neural network on synthetic patient data.

**Step 4: Model Export**
- Saved trained model (.h5) and preprocessing tools (.pkl).

**Step 5: Backend API (Flask)**
- Built API endpoints to serve real-time predictions from the model.

**Step 6: Frontend Integration**
- Web app connects user symptom input to API and displays top 3 disease predictions.


# Challenges Faced

- Managing imbalanced data (some diseases have fewer symptoms & ranking system).
- Choosing model complexity (not overfitting on mock patients).
- Cross-environment compatibility (TensorFlow versions, Python versions).
- Backend and frontend integration (CORS, API security).

## Key Learnings

- Building and serving a deep learning model in a real-world app.
- Handling full-stack architecture (frontend, backend, model).
- Team collaboration with Google Colab.
- Importance of user-centered design for health-related apps.

## Next Steps

- Train on real patient data (if available) for even better accuracy.
- Add ranking for better predictions
- Deploy to cloud server (AWS/GCP) for public use.
- Add user accounts and prediction history saving.


# Thank You