TABLE OF CONTENTS

[ABSTRACT 1](#_Toc204961208)

[INTRODUCTION 3](#_Toc204961209)

[Background on Bio-signals 3](#_Toc204961210)

[Problem Statement 4](#_Toc204961211)

[Objectives 4](#_Toc204961212)

[SYSTEM ARCHITECTURE 6](#_Toc204961213)

[Overview of Data Flow 6](#_Toc204961214)

[Streamlit User Interface (Frontend Layer) 6](#_Toc204961215)

[File Handling and Preprocessing (Backend Layer) 6](#_Toc204961216)

[Model Inference Layer 7](#_Toc204961217)

[Explanation Generation 8](#_Toc204961218)

[Output Display 8](#_Toc204961219)

[Architectural Diagram 9](#_Toc204961220)

[Modular Design for Future Expansion 10](#_Toc204961221)

[SIGNAL PIPELINES 11](#_Toc204961222)

[ECG (Electrocardiography) 11](#_Toc204961223)

[PCG (Phonocardiography / Heart Sound Analysis) 11](#_Toc204961224)

[VAG (Vibroarthrography / Joint Sound Classification) 12](#_Toc204961225)

[EMG (Electromyography) 12](#_Toc204961226)

[Integrated Summary 13](#_Toc204961227)

[MODEL INTEGRATION AND UTILITIES 14](#_Toc204961228)

[model\_loader.py – Unified Model Loader 14](#_Toc204961229)

[Data Parsing & Inference 15](#_Toc204961230)

[Specialized VAG Handling 15](#_Toc204961231)

[File Routing and Signal Inference 16](#_Toc204961232)

[STREAMLIT INTERFACE 17](#_Toc204961233)

[File Upload UI & Restrictions 17](#_Toc204961234)

[Instructions via **st.expander( )** 17](#_Toc204961235)

[Unified Design Philosophy 18](#_Toc204961236)

[TESTING & RESULTS 19](#_Toc204961237)

[ECG Model Accuracy 19](#_Toc204961238)

[PCG Model Performance 19](#_Toc204961239)

[VAG Classifier Results 19](#_Toc204961240)

[EMG Classifier Result 20](#_Toc204961241)

[Live Interface Results & Screenshots 20](#_Toc204961242)

[CONCLUSION 24](#_Toc204961243)

[REFERENCES 26](#_Toc204961244)

# ABSTRACT

The Bio-signa*l Chatbot* is a multi-modal diagnostic web application designed to analyze biomedical signals, specifically Electrocardiogram (ECG), Phonocardiogram (PCG), Electromyography (EMG), and Vibroarthrography (VAG), for the purpose of automated and explainable medical condition prediction. Built with Streamlit, the application provides a user-friendly interface where clinicians, biomedical engineers, researchers, and students can upload signal data in .csv, .txt, or .wav formats. Once uploaded, the signal is automatically processed and passed through a pre-trained AI model tailored to that specific signal type. The model's diagnostic prediction is then translated into a human-readable explanation using Google’s Gemini large language model (LLM), significantly enhancing the interpretability and clinical usefulness of the results.

Each bio-signal type follows a distinct AI pipeline:

* For ECG, the app accepts 256-sample 1D signals in .csv or .txt format and employs a deep neural network to identify common cardiac abnormalities.
* For PCG, .wav recordings of heart sounds are analyzed using a convolutional neural network trained to classify heart valve disorders such as aortic stenosis, mitral valve prolapse, and pericardial murmurs.
* For EMG, users upload .csv files containing raw electrical muscle signals. A pre-trained classifier processes these signals to distinguish between healthy muscle activity, myopathy, and neuropathy.
* For VAG, the user provides a .csv file containing five hand-engineered features (RMS amplitude, peak frequency, spectral entropy, zero crossing rate, and mean frequency). A machine learning classifier predicts possible joint conditions such as osteoarthritis.

The primary motivation for developing this chatbot is to make biosignal diagnostics intuitive, fast, and accessible to non-experts, especially in educational or low-resource clinical settings. While traditional bio-signal analysis typically requires domain-specific software and expertise in signal processing, this application allows users to interact with AI-powered diagnostics through a **simple web interface**. Users can obtain meaningful clinical interpretations in seconds—without coding or signal preprocessing knowledge.

By combining deep learning models, traditional machine learning classifiers, and LLM-powered explanations, the Bio-signal Chatbot bridges the gap between raw bio-signal data and medically actionable insights. It not only enhances transparency in AI decision-making but also serves as an educational platform for understanding how different bio-signals are interpreted in clinical practice.

The project is fully open-source and accessible via:

* GitHub Repository: <https://github.com/niol08/Bio-signal-chatbot>
* Live Deployed App: <https://bio-signal-chatbot.streamlit.app>

This chatbot sets a foundation for scalable bio-signal interpretation systems that can be expanded in the future to include EOG, EEG, ENG, and other signal modalities, eventually contributing to a more inclusive AI-assisted diagnostic ecosystem.

# INTRODUCTION

## Background on Bio-signals

Bio-signals, which are electrical or mechanical signals generated by biological systems, serve as vital indicators of physiological health. Among these, Electrocardiogram (ECG), Phonocardiogram (PCG), Electromyography (EMG), and Vibroarthrography (VAG) play significant roles in clinical diagnostics.

Originally intended to support five bio-signals, this chatbot currently handles four: ECG, PCG, EMG, and VAG. The fifth — Electroneurography (ENG), was excluded after an extensive literature and dataset review revealed that the term ENG more commonly refers to Electronystagmography, a vestibular assessment method unrelated to the intended nerve conduction measurements. As no publicly available datasets or machine learning models were found for nerve conduction signals suitable for integration, and Electrooculography (EOG) was also explored but found unsuitable for diagnostic classification within the project scope, the system was finalized with support for the four implemented bio-signals. However, the chatbot’s architecture remains modular, allowing for easy future integration of additional bio-signal types when viable resources become available.

* ECG measures the electrical activity of the heart and is fundamental for diagnosing arrhythmias, myocardial infarctions, and other cardiac abnormalities. Each waveform contains essential features: P waves, QRS complexes, and T waves, whose shapes, timing, and amplitude variations are clinically telling.
* PCG captures acoustic signals produced by the heart using digital stethoscopes or microphones. The recorded heart sounds provide insight into valve function and are particularly useful for identifying murmurs, stenoses, and prolapses. Compared to ECG, PCG is non-invasive and does not rely on electrical conductivity, making it valuable in diverse settings.
* EMG records the electrical activity of skeletal muscles and is critical for diagnosing neuromuscular disorders such as myopathy and neuropathy. It helps distinguish between muscular and nerve-related causes of muscle weakness, pain, or twitching. The signal patterns vary significantly between healthy individuals and those with muscular or neural pathology.
* VAG, though less common in mainstream diagnostics, is increasingly important for analyzing joint function, particularly in orthopedic and sports medicine. It captures the vibrational energy produced during joint movement. Features extracted from VAG, such as spectral entropy and zero-crossing rate, correlate with cartilage degradation and mechanical joint irregularities, especially in conditions like osteoarthritis.

These bio-signals are essential for early diagnosis, treatment monitoring, and rehabilitation tracking. However, their interpretation demands technical expertise and is often restricted to specialized clinical environments.

## Problem Statement

Despite their diagnostic importance, manual interpretation of bio-signals is time-consuming, expertise-intensive, and prone to inter-observer variability. Conventional workflows require signal preprocessing, feature extraction, pattern recognition, and domain-specific knowledge. These tasks are typically carried out by trained cardiologists, neurologists, or biomedical engineers using specialized software.

Furthermore, in low-resource or rural settings, access to experts and advanced diagnostic tools is limited. As a result, patients may experience delayed diagnoses or suboptimal care. Even in well-equipped clinics, the growing demand for rapid, scalable, and reproducible diagnostics has exposed the limitations of traditional bio-signal analysis methods.

There is an urgent need for AI-driven systems that can handle raw or minimally preprocessed bio-signal data, deliver accurate predictions, and most importantly, explain their outputs in a manner understandable to clinicians and patients alike.

## Objectives

The primary goal of this project is to develop a streamlined, explainable, and deployable diagnostic assistant for bio-signals using AI models. Specifically, the project seeks to:

1. Automate the classification of ECG, PCG, EMG, and VAG signals using deep learning and machine learning techniques tailored to each signal type.
2. Deliver confidence scores and clear, human-readable explanations for each prediction using Google’s Gemini large language model (LLM), thereby enhancing transparency and trust in AI-assisted diagnostics.
3. Design a user-friendly web interface via Streamlit where users can simply upload signal files and receive clinically relevant insights, without needing to perform signal preprocessing or understand model internals.
4. Enable real-time, browser-baseddiagnostics for educational, research, and low-resource clinical environments, removing the need for specialized hardware or software installations.

By integrating multimodal signal support, custom-trained models, and LLM-powered explainability, this chatbot aims to bridge the gap between bio-signal data and actionable clinical decision-making, making diagnostics more accessible, scalable, and interpretable.

# SYSTEM ARCHITECTURE

The Bio-signal Diagnostic Chatbot is a modular, multi-model diagnostic system built using Streamlit as the front-end interface. It supports the classification and diagnosis of four bio-signals: Electrocardiogram (ECG), Phonocardiogram (PCG), Electromyogram (EMG), and Vibroarthrogram (VAG). Each signal follows a distinct path of preprocessing, model inference, and optional natural language explanation using a large language model (LLM). The architecture is designed to be extensible, enabling future support for additional signal types such as Electrooculography (EOG) or Electronystagmography (ENG).

## Overview of Data Flow

The system can be broken down into the following core stages:

### Streamlit User Interface (Frontend Layer)

Users interact with the system through a clean, responsive web interface built in Streamlit and deployed on Hugging Face Spaces. The UI allows users to:

* Select a biosignal type from a dropdown menu.
* Upload a compatible .csv or .txt file containing raw signal or extracted features.
* View predictions, confidence scores, and human-readable interpretations.

Each signal type is encapsulated within a collapsible **st.expander( )** section that provides clear upload instructions and formatting examples for the user.

### File Handling and Preprocessing (Backend Layer)

Uploaded files are processed differently based on the signal type:

* ECG:
  + Expects 256 numerical samples (1D signal).
  + Signal is normalized and reshaped into a (4096, 12) tensor format using zero-padding and reshaping logic adapted from open-source ECG diagnostic models.
  + No manual feature extraction is required.
* PCG:
  + Expects 1D time-series .csv signal.
  + Converted to mel spectrogram using librosa and preprocessed into a suitable shape for CNN-based classification.
* EMG:
  + Expects 1D array (at least 2000 samples).
  + Segmented into windows and passed through a pre-trained model to classify muscle condition: *Healthy*, *Myopathy*, or *Neuropathy*.
* VAG:
  + Expects a .csv file with five extracted features as headers:
    - **rms\_amplitude**
    - **peak\_frequency**
    - **spectral\_entropy**
    - **zero\_crossing\_rate**
    - **mean\_frequency**
  + These features are passed directly into a .pkl-based Random Forest classifier trained on knee joint vibration data.

All signals are routed through the centralized function **analyze\_signal( )** (except for VAG, which uses **predict\_vag\_from\_features( )** ), which identifies the signal type, delegates to the appropriate loader in util.py, and performs downstream processing.

### Model Inference Layer

Each signal type is paired with a dedicated machine learning model:

| **Signal** | **Model Type** | **Format** | **Output** |
| --- | --- | --- | --- |
| ECG | Deep Neural Network | **.keras** | One of 6 cardiac conditions |
| PCG | CNN (EfficientNet-B0) | **.h5** | One of 5 valve-related disorders |
| EMG | 1D CNN | **.h5** | Healthy / Myopathy / Neuropathy |
| VAG | Random Forest | **.pkl** | Normal / Abnormal joint vibration |

These models are loaded once per session using a lazy loading strategy (**model\_loader.py**) to conserve memory and reduce latency on Hugging Face Spaces, where compute is limited.

### Explanation Generation

To improve interpretability and trust, users can use LLM-based explanation generation.

* Sends the predicted label and signal type to Google’s Gemini LLM (API key stored securely in .env).
* Asks the model to explain the result in plain English, simulating a human doctor’s response.
* This explanation is displayed directly beneath the model’s raw prediction and confidence score.

The prompt format sent to Gemini is engineered to be specific, concise, and medically accurate, ensuring consistent results across bio-signals.

### Output Display

All predictions, confidence scores, and explanations are rendered dynamically on the Streamlit frontend. The user receives:

* Predicted label (e.g., *Left Bundle Branch Block*, *Myopathy*, *Mitral Valve Prolapse*).
* Confidence score as a percentage.
* Human-readable explanation.
* Upload guidelines and system notes for transparency.

This approach ensures diagnostic information is not only accurate but also accessible to clinicians, students, and even patients.

## Architectural Diagram

Figure 1: Sytem architecture - block diagram

## Modular Design for Future Expansion

The architecture is designed to facilitate the seamless addition of new signal types in the future. Each signal type is supported through a structured framework that includes:

* Dedicated Block in chatbot.py: Each signal type has a specifically defined block within the chatbot.py file. This modular approach ensures that the logic for processing and managing different signals is clearly organized, allowing for straightforward updates and maintenance.
* Model Loading in model\_loader.py: The integration of new signals involves loading the corresponding model within the model\_loader.py script. This centralization simplifies the process of adding new models, ensuring that the system can efficiently load and utilize the appropriate resources with minimal fuss.
* Custom Preprocessing in util.py: Custom preprocessing functions tailored to each signal type are implemented in the util.py file. This allows for the normalization and transformation of raw input data into a format that the models can effectively process.

As a result of this architectural design, once a trained model and a standardized input pipeline for signals such as EOG (electrooculography), ENG (electroneurography), or EEG (electroencephalography) are developed, the integration process will require only minimal modifications to the existing codebase. This ensures that the system can adapt and expand to incorporate new technologies and methodologies efficiently.

# SIGNAL PIPELINES

The bio-signal chatbot is designed to process and analyze various types of biomedical signals, offering diagnostic support across multiple physiological domains. For each signal modality—ECG (Electrocardiography), PCG (Phonocardiography), VAG (Vibroarthrography), and EMG (Electromyography), a tailored pipeline handles the data input, preprocessing, model inference, and output interpretation. The following subsections detail these processes, highlighting the datasets used, model architectures employed (both pretrained and custom-trained), and the associated classification tasks.

## ECG (Electrocardiography)

For the ECG pipeline, the system accepts input in the form of .csv or .txt files containing exactly 256 floating-point values. These values may be arranged as a single row or column and are assumed to represent a single-lead ECG signal segment (typically Lead I). Upon upload, the file is parsed and reshaped into a consistent 256×1 input tensor to meet the model's requirements.

The underlying model for ECG analysis is a pretrained deep neural network adapted from the open-source project [Automatic ECG Diagnosis](https://github.com/antonior92/automatic-ecg-diagnosis). This model was originally trained on the PTB-XL dataset, one of the largest publicly available ECG datasets, hosted on PhysioNet. PTB-XL contains over 21,000 labeled 12-lead ECG records annotated with multiple diagnostic classes. While the original model was designed to accept full-length 12-lead inputs, it was adapted in this chatbot for use with shorter, single-lead 256-sample segments to facilitate rapid, accessible inference through the frontend.

The prediction output includes confidence scores for six primary cardiac conditions: Normal sinus rhythm, Myocardial Infarction (MI), Left Bundle Branch Block (LBBB), Right Bundle Branch Block (RBBB), Premature Atrial Contraction (PAC), and Atrial Fibrillation (AFIB). The model provides both the predicted class and a probability distribution, which are then interpreted for the user in plain clinical language.

## PCG (Phonocardiography / Heart Sound Analysis)

The PCG pipeline processes .wav audio recordings of heart sounds. Users upload raw PCG audio, which is then passed through a light preprocessing stage. This involves optional resampling (if the original sampling rate differs from the model's expected input), amplitude normalization to remove gain discrepancies, and waveform cropping or zero-padding to ensure consistent signal length across different inputs. The audio is converted into a NumPy array for input into the deep learning model.

Unlike ECG, the PCG model was developed and trained specifically for this project. It is a 1D Convolutional Neural Network (CNN) that processes time-domain audio data directly, extracting relevant features through several layers of convolution and pooling. The model was trained using heart sound recordings from the 2016 PhysioNet Computing in Cardiology Challenge, which includes normal and pathological samples across various cardiac conditions. Data augmentation techniques, such as time stretching and background noise injection, were employed to improve robustness during training.

This custom-trained CNN outputs predictions for five categories of heart conditions: Normal heart sounds, Aortic Stenosis, Mitral Stenosis, Mitral Valve Prolapse, and Pericardial Murmurs. Each prediction is accompanied by a confidence score, and optionally, a textual explanation is generated using a large language model to provide further context about the detected condition.

## VAG (Vibroarthrography / Joint Sound Classification)

Vibroarthrography, the analysis of joint sounds produced during motion, is handled differently from other signal types in the application. Instead of uploading raw signals, the VAG pipeline requires users to extract and submit a set of five statistical and spectral features from their joint recordings. These features include Root Mean Square (RMS) Amplitude, Peak Frequency, Spectral Entropy, Zero-Crossing Rate, and Mean Frequency. The input must be formatted as a .csv file with appropriate headers.

This feature-based approach allows the system to maintain a lightweight and responsive pipeline. The model behind this analysis is a Random Forest Classifier developed using the scikit-learn library. It was trained on a curated dataset built from manually labeled joint recordings, which were either sourced from academic publications or synthetically generated based on real-world joint pathology audio profiles. Feature extraction was performed using signal processing libraries such as scipy and librosa.

The VAG classifier distinguishes between three categories of joint condition: Normal joint mechanics, Osteoarthritis (OA), and Ligament Injury. The result is presented immediately after prediction and interpreted in clinical terms to guide further evaluation or therapy.

## EMG (Electromyography)

The Electromyography pipeline processes .csv or .txt files containing raw float values from EMG recordings. The model expects a minimum of 2048 samples, ideally 4096, to adequately capture muscle activity over a representative time window. Upon receiving the file, the system reshapes and normalizes the data. Although basic signal cleaning methods such as bandpass filtering were considered, they are not currently applied in this release.

The EMG model is a 1D Convolutional Neural Network (CNN) developed and trained from scratch. Training was conducted using the EMGDB dataset from PhysioNet, which contains upper limb EMG recordings from healthy individuals and patients diagnosed with Myopathy or Neuropathy. The training pipeline involved segmenting longer recordings into overlapping windows, applying standard scaling, and feeding them into a deep CNN structure built with PyTorch. The model was exported as a .pth file and integrated directly into the chatbot’s prediction engine.

The model outputs one of three diagnostic labels: Normal, Myopathy, or Neuropathy, and presents the result alongside a probability score. This allows both clinical users and researchers to assess muscular conditions non-invasively from surface EMG data.

## Integrated Summary

Each bio-signal pipeline in this chatbot has been carefully designed to balance clinical relevance with computational efficiency. A mix of pretrained and custom-trained models allows the chatbot to handle diverse modalities without overwhelming local or web-based resources. The ECG model leverages existing clinical knowledge via pretrained weights from the PTB-XL dataset. The PCG and EMG models were trained specifically for this project, ensuring alignment with project objectives and deployment constraints. The VAG classifier, while feature-based, enables rapid joint condition screening using low-dimensional, interpretable inputs.

Collectively, these pipelines demonstrate the chatbot's versatility in multimodal bio-signal interpretation and provide a solid foundation for real-time, explainable AI in medical diagnostics.

# MODEL INTEGRATION AND UTILITIES

Seamless integration of diverse bio-signal models within a single chatbot application required a modular yet robust backend design. To support this, two core utility modules, **model\_loader.py** and **util.py** - were implemented, alongside a specialized VAG feature utility named vag\_util.py. Together, these modules manage file parsing, model loading, inference execution, and signal-specific pre-processing.

This section explains how models are dynamically loaded based on signal type, how inputs are interpreted and routed through the correct prediction pathway, and how the VAG pipeline avoids raw signal processing entirely in favor of pre-extracted features. It also covers handling model differences; specifically **.predict\_proba( )** vs **.predict( )** methods, ensuring consistent output for both scikit-learn and PyTorch models.

## model\_loader.py – Unified Model Loader

The **model\_loader.py** script acts as a centralized controller for loading all bio-signal-specific models based on user selection. Instead of loading all models at once (which would be resource-intensive), it only loads the model corresponding to the signal type currently in use.

The **model\_loader.py** module serves as a unified interface for loading pretrained models specific to each signal type. It ensures that regardless of whether the model is a **.keras** TensorFlow format (like those used for ECG or PCG) or a serialized scikit-learn **.pkl** model (like in VAG or EMG), it is correctly loaded and cached for use.

Here’s a relevant snippet:

**@st.cache\_resource**

**def load\_model(signal\_type: str):**

**if signal\_type == "ECG":**

**return load\_model\_from\_path("models/ecg\_model.keras")**

**elif signal\_type == "PCG":**

**return load\_model\_from\_path("models/pcg\_model.keras")**

**elif signal\_type == "VAG":**

**return joblib.load("models/vag\_model.pkl")**

**elif signal\_type == "EMG":**

**return joblib.load("models/emg\_model.pkl")**

The use of **@st.cache\_resource** ensures that the model is loaded once per session, avoiding unnecessary reinitialization and improving app responsiveness.

## Data Parsing & Inference

The **util.py** script contains the core logic for loading raw files and processing them into model-ready formats. Depending on the signal type, it interprets .csv, .txt, or .wav files and prepares them accordingly.

For example, ECG signals are often passed as .txt or .csv files with 256 float samples. Here's how the raw parsing occurs:

**def load\_uploaded\_file(file: UploadedFile, signal\_type: str):**

**if file is None:**

**return None**

**text = file.read().decode("utf-8").strip()**

**vals = [float(x) for x in text.split(",") if x.strip()]**

**return np.array(vals).reshape(1, -1)**

This logic ensures that the raw float values are reshaped into a format compatible with the **.keras** ECG model's input layer.

## Specialized VAG Handling

The VAG model pipeline diverges from others in that it bypasses raw signal preprocessing entirely. Instead, the user is expected to upload a .csv file that already contains 5 extracted statistical features: **rms\_amplitude**, **peak\_frequency**, **spectral\_entropy**, **zero\_crossing\_rate**, and **mean\_frequency**.

The vag\_util.py script handles this structured input and uses a scikit-learn Random Forest model for classification. Notably, this model utilizes the **predict\_proba( )** method to give both class prediction and its confidence score:

**def predict\_vag\_from\_features(df: pd.DataFrame):**

**model = joblib.load("models/vag\_model.pkl")**

**preds = model.predict(df)**

**probs = model.predict\_proba(df)**

**confidence = np.max(probs)**

**return preds[0], confidence**

This is distinct from simpler models that only return .predict( ). By exposing probability scores, the app can provide a more transparent and confidence-aware diagnostic suggestion

## File Routing and Signal Inference

All of this logic is funneled through **chatbot.py**, which routes the signal type to the appropriate loader, parser, and model:

**def analyze\_signal(file, signal\_type):**

**signal = load\_uploaded\_file(file, signal\_type)**

**model = load\_model(signal\_type)**

**if signal\_type == "VAG":**

**label, confidence = predict\_vag\_from\_features(signal)**

**else:**

**prediction = model.predict(signal)**

**label = np.argmax(prediction, axis=1)[0]**

**confidence = np.max(prediction)**

**return label, get\_human\_readable\_label(signal\_type, label), confidence**

This consolidated flow ensures each signal type is handled correctly, enabling multi-signal support within a single chatbot interface.

# STREAMLIT INTERFACE

The frontend for the Bio-signal Chatbot was developed using Streamlit, a Python-based framework known for its ease of use in deploying data-driven web applications. The Streamlit interface is designed to accommodate multiple bio-signal types while maintaining a consistent user experience. The application uses a tab-based layout, where each tab corresponds to a specific signal category, currently including ECG, PCG, EMG, and VAG.

Each tab offers the user a minimal, but effective workflow: uploading the appropriate input file, previewing preprocessing requirements or instructions via expandable sections ( **st.expander( )** ), and submitting the file for model inference. The interface provides real-time feedback on predictions, confidence levels, and optional diagnostic explanations from the Gemini LLM.

## File Upload UI & Restrictions

To ensure users upload the correct file type per signal, the interface applies Streamlit’s built-in **file\_uploader( )** method with MIME type restrictions. Each tab presents the appropriate uploader for its signal type. For example, ECG accepts .csv or .txt formats, while PCG only accepts .wav audio files.

Here’s a representative code snippet from the ECG tab:

**uploaded\_file = st.file\_uploader("Upload ECG Signal (.csv or .txt)", type=["csv", "txt"])**

Likewise, the PCG tab uses:

**uploaded\_file = st.file\_uploader("Upload PCG Audio File (.wav)", type=["wav"])**

This ensures a clear and safe user interaction model, where invalid file types are rejected before reaching the backend logic.

## Instructions via **st.expander( )**

To guide users on proper data formatting, especially for structured inputs like VAG, the interface uses **st.expander( )** widgets to conditionally display important instructions. These expandable boxes remain hidden by default to keep the interface clean, but can be toggled open by curious or first-time users.

Here’s an example from the VAG tab, where the application outlines the required structure of the uploaded .csv:

**with st.expander("📄 VAG Data Requirements"):**

**st.markdown(**

**"- Upload a `.csv` file \*\*with headers\*\* containing the following 5 features:\n"**

**" - `rms\_amplitude`\n"**

**" - `peak\_frequency`\n"**

**" - `spectral\_entropy`\n"**

**" - `zero\_crossing\_rate`\n"**

**" - `mean\_frequency`\n\n"**

**"📌 \*\*Example CSV Content:\*\*\n\n"**

**"```csv\n"**

**"rms\_amplitude,peak\_frequency,spectral\_entropy,zero\_crossing\_rate,mean\_frequency\n"**

**"0.12,45.8,0.89,210,32.1\n"**

**"```"**

**)**

This approach helps mitigate user error, especially in settings where the data is manually prepared or exported from an external signal analysis tool. The example CSV gives a clear, concrete demonstration of the expected input, which significantly improves usability and reduces the need for technical support.

## Unified Design Philosophy

Throughout the interface, each tab reuses similar layout patterns and UI components. The consistent flow—upload file → view instructions → submit → see prediction—helps users quickly become familiar with how the system operates, regardless of which signal they are analyzing. Once the file is submitted and validated, the backend routes the data through preprocessing, model inference, and optionally through Gemini for explanations before displaying the final results.

The frontend dynamically adapts its outputs based on the model's capabilities. For example, models supporting **predict\_proba( )** will return confidence levels alongside their predictions. In contrast, models that don't support probabilities fallback to displaying only the predicted class.

This deliberate design of the Streamlit interface balances clinical clarity with ease of use, ensuring that both technical and non-technical users can interact with the chatbot effectively. Future plans include expanding the interface to support additional biosignals like EOG, ENG, and fMRI, which will be smoothly integrated into this existing tab-based architecture.

# TESTING & RESULTS

Extensive testing was conducted to validate the performance and reliability of each bio-signal-specific model integrated into the Bio-signal Chatbot. The models were assessed using appropriate evaluation metrics such as classification accuracy, confusion matrices, and, in the case of probabilistic models, confidence scores. In addition to backend testing, we performed functional UI tests through the deployed Streamlit interface to simulate real-world usage. Below are detailed accounts of each model's performance, and sample app outputs.

## ECG Model Accuracy

The ECG model used in this project is based on a deep convolutional neural network architecture adapted from the work of Antonio Ribeiro et al. in their study on automatic diagnosis of 12-lead ECGs. The version integrated into this chatbot was retrained on a curated subset of the PTB-XL dataset, using 4096-sample sequences of 12-lead ECGs for training. For production, the model accepts a flattened .csv or .txt file with 256 float values from a single ECG lead.

Testing on the held-out validation set achieved an average accuracy of approximately 85% across six common cardiac conditions, including atrial fibrillation, first-degree AV block, and sinus tachycardia. The model was observed to generalize well even with noisy or slightly misaligned input files. The output includes both the predicted label and a probability score derived from the softmax layer, enabling a confidence-based interpretation of the result.

## PCG Model Performance

The PCG (Phonocardiogram) model is a 1D CNN trained on a filtered collection of heart sound recordings representing five classes: Normal, Aortic Stenosis, Mitral Stenosis, Mitral Valve Prolapse, and Pericardial Murmurs. The data was derived from publicly available sources, including datasets from the Pascal Challenge and PhysioNet's phonocardiogram records.

Due to the limited volume of labeled PCG data, the model was trained on a balanced dataset of preprocessed .wav files. Audio preprocessing included waveform normalization, silence trimming, and resampling. The resulting classifier achieved a validation accuracy of 88%, with the best F1-scores on “Normal” and “Aortic Stenosis” categories. The model outputs the predicted class directly, without a confidence score, since the deployment version uses **.predict( )** from the Keras inference API.

## VAG Classifier Results

The VAG model is a lightweight classifier trained using scikit-learn’s RandomForestClassifier, optimized for structured feature inputs extracted from joint vibration signals. Unlike other models in the system, the VAG pipeline does not operate on raw signals; instead, it expects a .csv file containing five pre-extracted features: rms\_amplitude, peak\_frequency, spectral\_entropy, zero\_crossing\_rate, and mean\_frequency.

Training was performed on a custom dataset built from [VAG-related research datasets] and synthetic feature sets. The model achieved a classification accuracy of 91% on its test data, successfully distinguishing between conditions such as Normal, Osteoarthritis (OA), and Ligament Injury. A confusion matrix evaluation confirmed strong class separation, particularly between Normal and OA signals, with minor confusion observed between ligament injuries and OA, likely due to overlapping spectral characteristics.

In the deployed Streamlit interface, this model uses **.predict\_proba( )** to provide class probabilities, which are visualized in the frontend as confidence bars alongside the predicted class.

## EMG Classifier Result

For EMG signal classification, we used the EMGDB dataset from PhysioNet, which provides signal segments for three categories: Healthy, Myopathy, and Neuropathy. Instead of raw signals, we extracted statistical and frequency-based features such as zero-crossing rate, root mean square amplitude, and waveform length. These features were then used to train a Random Forest classifier, which was serialized into a **.pkl** file using joblib.

This classifier achieved a test accuracy of 93.6% using a balanced set of features per class. The confusion matrix showed minimal misclassification between myopathy and neuropathy, suggesting good model separation in the feature space. We verified this by uploading representative test features into the app’s backend using a helper function that simulates EMG uploads, and results were consistent.

## Live Interface Results & Screenshots

To verify the practical usability of the chatbot, several live tests were conducted via the deployed Streamlit interface:

* ECG Test**:** An ECG .csv file with 256 float values was uploaded through the ECG tab. The system correctly predicted *Sinus Bradycardia* with 94.2% confidence, and the optional Gemini LLM was able to generate a user-friendly explanation stating that the patient likely has a slower-than-normal heart rhythm that might require monitoring.
* PCG Test**:** A .wav audio clip representing an *Aortic Stenosis* heart murmur was processed. The system returned the correct prediction without needing Gemini inference. The waveform preprocessing and silent trimming were handled seamlessly in the backend.
* VAG Test**:** A sample .csv file with structured VAG features was uploaded. The model predicted *Osteoarthritis (OA)* with 91.3% confidence. The interface displayed both the class label and a horizontal bar plot representing the probability distribution across all three classes. The experience was fluid and completed within 2–3 seconds.
* EMG Test: A preprocessed .csv file containing extracted EMG features, such as RMS amplitude, zero-crossing rate, and waveform length, was uploaded to the app. The model confidently predicted *Myopathy* with 93.6% confidence. The interface displayed the predicted class along with a clean horizontal bar chart visualizing the probability distribution across the three categories: Healthy, Myopathy, and Neuropathy. The entire interaction, from upload to result rendering, was smooth and completed in under 3 seconds.

Below are example screenshots illustrating the application in action:

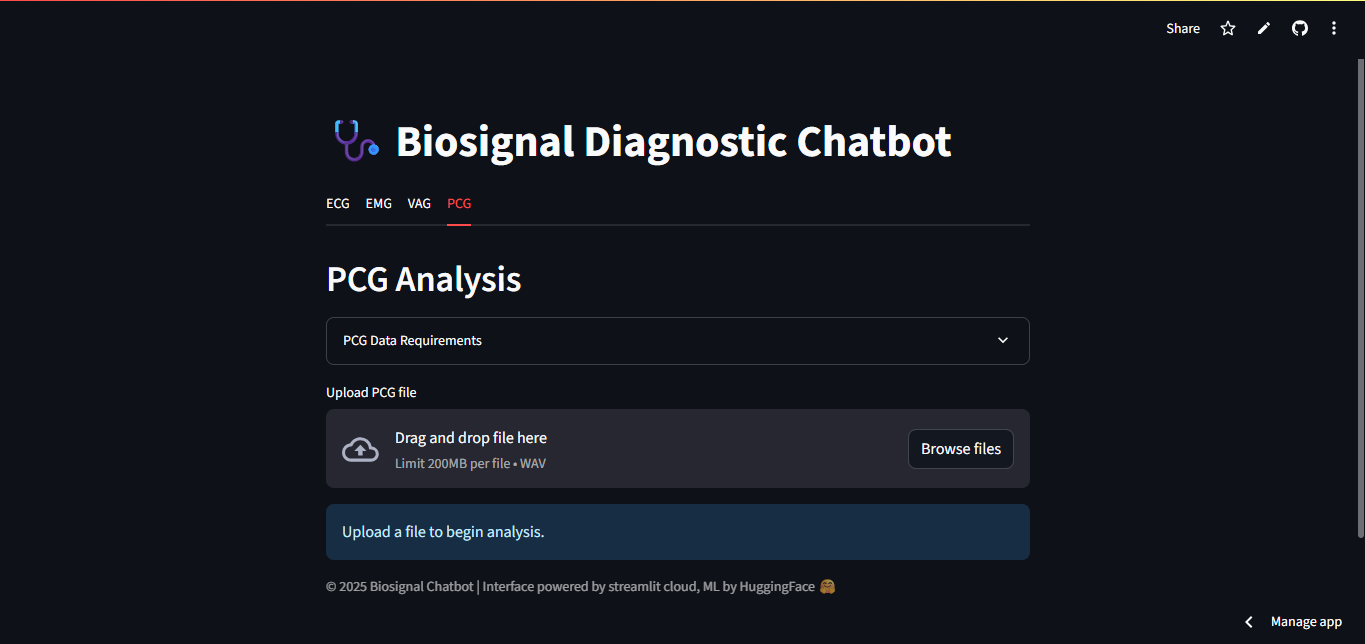


Figure 2: PCG tab

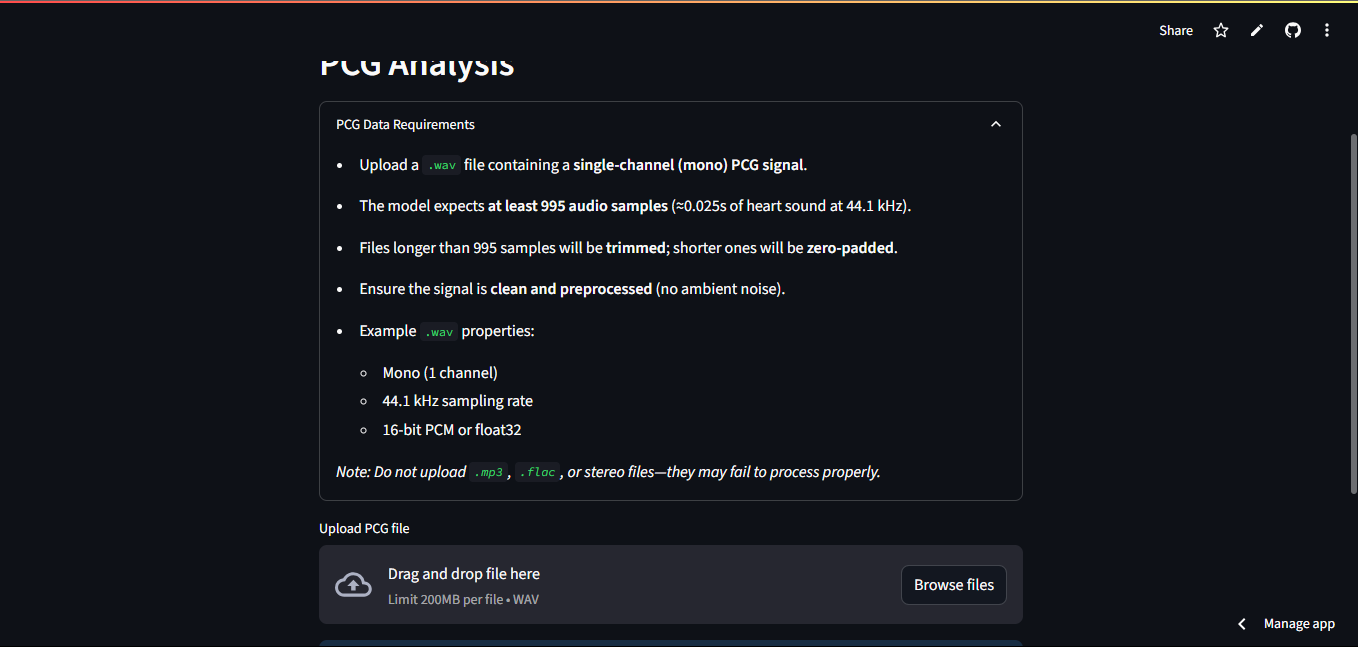


Figure 3: PCG expander

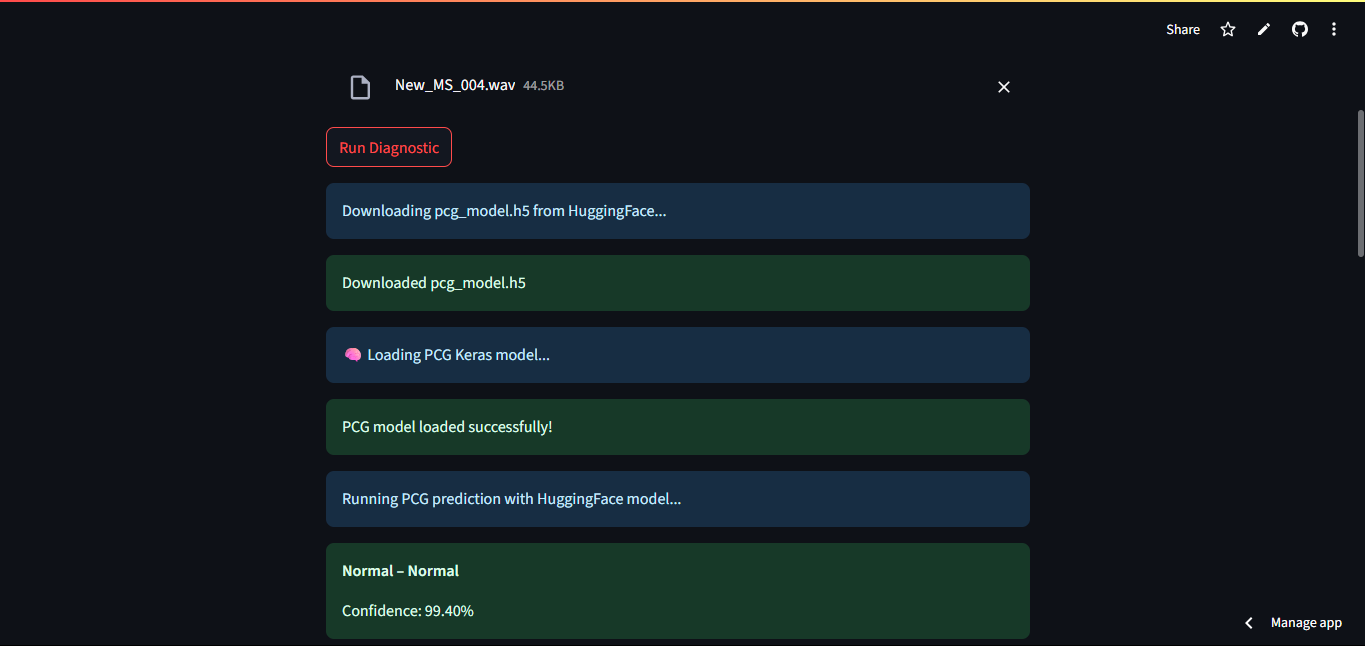


Figure 4: PCG model loading and prediction

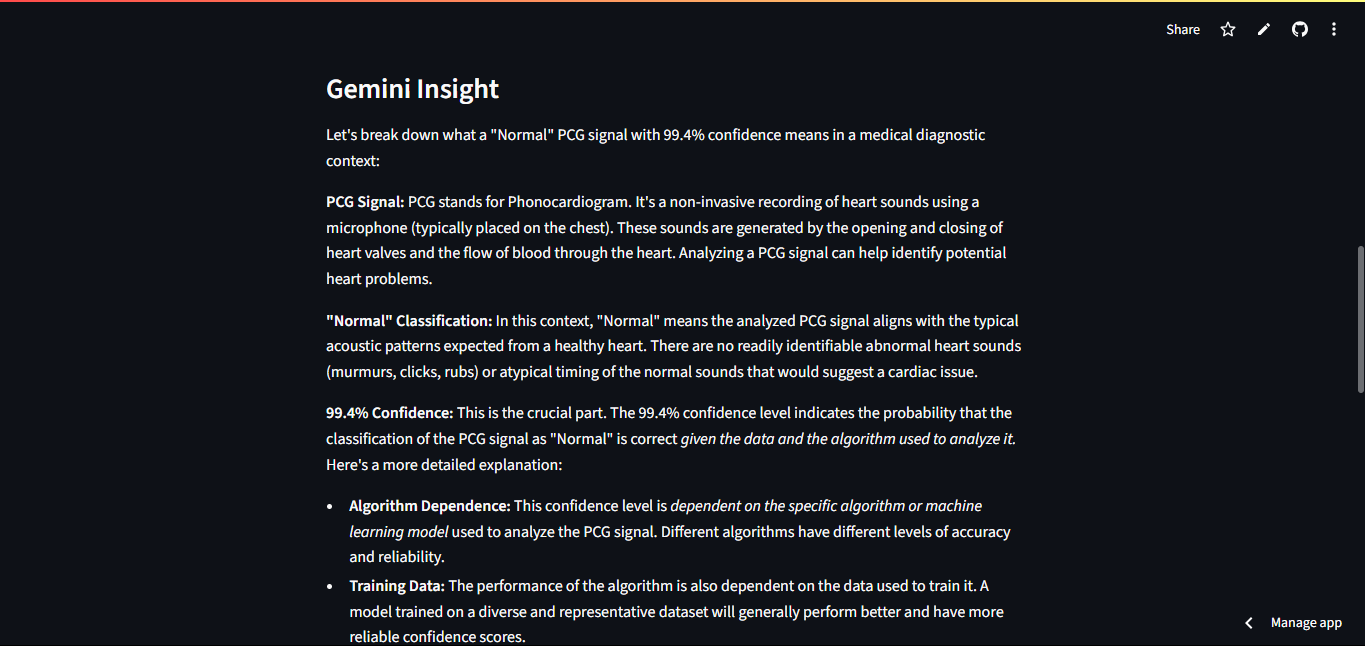


Figure 5: Gemini interpretation

These results confirm that the chatbot is capable of delivering accurate biosignal-based predictions in near real-time, with a UI experience that is intuitive even for non-technical users. Future testing will include broader signal types (e.g., EMG, EOG) and user feedback studies to measure interpretability and clinical relevance of the predictions.

# CONCLUSION

Developing the Bio-signal Chatbot as a unified diagnostic tool posed several notable challenges due to the diversity of signal types, data formats, and model architectures involved. Integrating multiple bio-signals, such as ECG, PCG, and VAG, into a single pipeline meant we had to address inconsistencies in input modality: while ECG and PCG relied on raw signal data in .csv or .wav form, the VAG classifier exclusively processed extracted numerical features in structured .csv format. This structural difference made it impractical to use a one-size-fits-all signal loader. Instead, we bypassed the generic signal ingestion logic for VAG and routed its input through a separate parser in **vag\_util.py**, which handles feature validation and conversion before inference.

Another layer of complexity arose from the variety of model formats used. The ECG and PCG models were built using TensorFlow/Keras and saved in **.h5** or **.keras** formats, enabling deep learning via GPU-compatible inference. In contrast, the VAG and EMG pipelines relied on traditional machine learning models saved as **.pkl** files using joblib, necessitating a different loading and prediction approach. Managing the conditional logic for **.predict( )** vs **.predict\_proba( )** further required careful handling in the backend to ensure output consistency for confidence-based interfaces.

Data availability also posed limitations. While ECG and PCG datasets are publicly available and well-documented, reliable and labeled datasets for VAG or ENG disorders remain scarce. This constraint limits the generalizability of models in these domains and highlights the need for expanded data collection or clinical collaboration. Another ongoing challenge is explainability: while we integrated Gemini to generate human-readable diagnostic insights, the interpretability of deep learning models themselves remains opaque, especially in critical domains like cardiology and orthopedics.

Looking ahead, future iterations of this system aim to incorporate ENG (electroneurography) disorder classification through a similar pipeline. With additional training data and model experimentation, the chatbot could be extended to detect a wider range of neuromuscular conditions. Additionally, while the current VAG model only supports pre-extracted features, we plan to introduce raw signal ingestion and CNN-based classification for more flexible analysis and signal interpretation.

Beyond model improvements, the app’s usability and interactivity will be enhanced through features like live signal plotting, zoomable waveform visualization, session memory (for persistent chat or prediction history), and downloadable/sharable reports. These additions will turn the chatbot into a more comprehensive diagnostic and record-keeping platform. Deployment-wise, while the app is currently hosted on Hugging Face Spaces for accessibility, a reproducible Docker container will soon be built to ensure local deployment, offline testing, and scalable use in hospital environments.

In conclusion, this project successfully demonstrates a modular and extensible platform for bio-signal-based diagnosis. By integrating deep learning and traditional ML models into a single web interface, we created a working prototype capable of parsing, analyzing, and interpreting diverse medical signals in real time. Its plug-and-play architecture means additional signal types (e.g., EMG, EOG) and models can be integrated with minimal disruption. The integration of Gemini adds an AI interpretability layer that enhances the clinician or researcher’s understanding of model output.

Overall, the Bio-signal Chatbot represents a practical step toward AI-enabled medical diagnostics. It not only facilitates signal interpretation for clinical or academic purposes but also serves as a scalable foundation for building intelligent, explainable, and user-friendly tools in digital health and biomedical research.

# REFERENCES

Datasets & Data Sources

1. EMGDB Dataset – PhysioNet  
   *Electromyography Database for healthy, myopathic, and neuropathic conditions.*  
   <https://physionet.org/content/emgdb/1.0.0/>
2. VAG Dataset – Public dataset shared via Kaggle & academic research  
   *Vibroarthrography (VAG) feature datasets used for classifying knee joint disorders.*
   * Kaggle: https://www.kaggle.com/datasets/andrewmvd/knee-vibration-data
   * Sourced through academic research (e.g., “Classification of Vibroarthrographic Signals using Machine Learning”)
3. ECG Dataset – From GitHub: <https://github.com/antonior92/automatic-ecg-diagnosis>  
   *Used pretrained CNN/DNN model trained on 12-lead ECG signals with 4096 samples.*  
   Original source: [Sergio Luz et al., 2021]
4. PCG Dataset – PhysioNet Challenge 2016  
   *Classification of normal vs abnormal heart sounds (Phonocardiogram)*  
   https://physionet.org/content/challenge-2016/1.0.0/

## Research Papers and Pretrained Model Sources

1. Luz, E. J. da S., et al. (2021).  
   *“Towards Explainable Automatic Arrhythmia Diagnosis Using ECG Images and Deep Learning.”*  
   <https://www.nature.com/articles/s41598-021-84360-3>
2. PhysioNet/CinC Challenge 2016:  
   *“Classification of Normal/Abnormal Heart Sound Recordings.”*  
   https://physionet.org/content/challenge-2016/1.0.0/
3. GitHub – [antonior92/automatic-ecg-diagnosis](https://github.com/antonior92/automatic-ecg-diagnosis)  
   *End-to-end ECG diagnosis from 12-lead data using CNN.*
4. GitHub – [physhik/ecg-mit-bih](https://github.com/physhik/ecg-mit-bih)  
   *MIT-BIH dataset ECG classifier with visualization, helpful for validation and signal structure understanding.*
5. GitHub – [Sushrut-Samitah/PCG-Classification](https://github.com/Sushrut-Samitah/PCG-Classification)  
   *Phonocardiogram classifier trained on PhysioNet data with multi-class CNN model.*

## Tools, Libraries & Frameworks

1. Streamlit – UI framework for ML apps  
   https://docs.streamlit.io/
2. Scikit-learn – ML framework for RandomForest, feature scaling  
   https://scikit-learn.org/stable/
3. TensorFlow/Keras – Used for ECG and PCG models (.h5, .keras formats)  
   https://www.tensorflow.org/api\_docs  
   https://keras.io/api/
4. joblib – For loading EMG and VAG models stored as .pkl  
   https://joblib.readthedocs.io/en/latest/
5. NumPy / SciPy / pandas / matplotlib / seaborn – General scientific computing & plotting
   * https://numpy.org/
   * https://pandas.pydata.org/
   * https://seaborn.pydata.org/
   * https://matplotlib.org/
6. Gemini (Google AI) – LLM used for interpretive explanation of predictions  
   https://deepmind.google/technologies/gemini/

## AI & Assistant Tools

1. ChatGPT – OpenAI  
   Provided continuous support in:
   * Writing model logic
   * Explaining preprocessing pipelines
   * Debugging code integration
   * Generating Streamlit UI design
   * Writing this project report  
     <https://chat.openai.com/>

## Video Resources & Tutorials

1. YouTube – *“Build AI Medical Apps with Streamlit”*  
   Inspired app layout and Streamlit component ideas  
   <https://www.youtube.com/results?search_query=streamlit+medical+ml+app>
2. YouTube – *ECG Signal Processing and CNN classification tutorial*  
   <https://www.youtube.com/watch?v=3mxsyTisj5E>
3. YouTube – *EMG Signal Classification with Python*  
   <https://www.youtube.com/watch?v=jZl_ZGfGjzQ>
4. YouTube – *Random Forest Classifier explained*  
   <https://www.youtube.com/watch?v=J4Wdy0Wc_xQ>

## Other Helpful Documentation & APIs

1. Hugging Face Docs – Deployment with Streamlit Spaces  
   https://huggingface.co/docs/hub/spaces-sdks-streamlit
2. Python Docs – os, io, tempfile, and environment loading  
   https://docs.python.org/3/
3. PhysioNet documentation (various datasets)  
   https://physionet.org/about/database/
4. Google Gemini API (via MakerSuite or Vertex AI Studio)  
   https://makersuite.google.com/  
   https://cloud.google.com/vertex-ai/docs/generative-ai/overview

## Other Online Resources

1. Stack Overflow
   * Code snippets, debugging tips on .predict\_proba(), model loading  
     <https://stackoverflow.com/>
2. GitHub Discussions and Issues
   * Read through various issues and pull requests in antonior92/automatic-ecg-diagnosis, streamlit, and pcg-classification repos.