**Egyptian E-Learning University**

Faculty of Computers & Information Technology

**Doctor Bot: AI-Powered Skin**

**Disease Diagnosis System.**

**By**

|  |  |
| --- | --- |
| Ahmed Atef Ahmed Abdeen | 2101833 |
| Abdelrhman Beshary Hamada | 2101963 |
| Mohamed Hussein Abdallah | 2102033 |
| Mohamed Mansour Abdelwahab | 2102155 |
| Mohamed Osama Mohamed | 2102030 |
| Mohamed Ramadan Fouad | 2102038 |
| Moaz Ahmed Sayed | 2102097 |

Supervised by

Dr. Hanaa Bayoumi

Assistant

Eng. Doaa Mahmoud

[Qena]-2025

Abstract

The skin is one of the largest and fastest-growing tissues in the human body, making it more susceptible to more than 3,000 skin diseases. Skin diseases also distort the aesthetic appearance of a person, and some of them cause physical pain, which makes their impact psychological and physical. However, diagnosing skin diseases requires a visit to the doctor, which takes a long time, effort, and high costs, and dermatologist services are not available in many areas, especially in rural areas.

Based on this, Doctor Bot, an AI-based system was developed to assist in the initial diagnosis of skin diseases, making healthcare easier, faster and more cost-effective. The app allows users to upload photos of their skin conditions, as well as send a description of the accompanying symptoms, where AI algorithms analyze the images and text information to provide an accurate initial diagnosis.

Doctor Bot aims to improve access to healthcare, even if only in principle, especially in areas that lack dermatology services, contributing to early diagnosis and reducing the burden on healthcare systems. This project reflects the great development in artificial intelligence technologies and their increasing role in providing advanced and effective medical solutions.

Contents

[Abstract 2](#_Toc197463970)

[Acknowledgments 3](#_Toc197463971)

[Introduction 5](#_Toc197463972)

[Literature Review / Related Work 17](#_Toc197463973)

[Proposed system 21](#_Toc197463974)

[Implementation 29](#_Toc197463975)

[Testing & Evaluation 50](#_Toc197463976)

[Results & Discussion 53](#_Toc197463977)

[Conclusion & Future Work 56](#_Toc197463978)

[References 59](#_Toc197463979)

[Appendices (Optional) 61](#_Toc197463980)

Chapter 1

Introduction

* 1. **Introduction**

The skin is one of the most important and dynamic organ of the human body. Dermatoses are a manifestation which is multidimensional involving psycho-social, economic effects on life of the individual, family and society. The skin is sensitive and weakened and all ages can be victims of this issue. There are over 3,111 known skin diseases; such maladies, while not damaging to the health, can be extremely painful, disfiguring, and, in the long run, disabling. Chronic skin diseases, such as atopic dermatitis, psoriasis, vitiligo, and leg ulcers are not immediately life-threatening but have serious physical, emotional, and financial implications.

Traditional diagnosis of skin diseases is often reliant solely on doctors, which limits access to care and is slow, particularly in rural places where dermatologists are few and far between. Most would receive inadequate or incorrect treatment, and some seemingly trivial problems can cause serious complications if they aren't properly managed. In addition, the price can be cost-prohibitive for individuals with low incomes to be treated with proper care. Tackling these challenges, here we introduce the Doctor Bot project, an AI-based system for early diagnosis and basic treatment assistance. One may upload images and get a preliminary evaluation at any time, by being integrated into a chatbot.

The image is preprocessed for clarity and analyzed using a trained model to generate a diagnostic report. The selected diseases for this project include:

* **Eczema**
* **Ringworm**
* **Impetigo**
* **Contact Dermatitis**
* **Acne**
* **Scabies**
* **Psoriasis**

These diseases were chosen due to their prevalence, frequent misdiagnosis, and significant impact on quality of life. By focusing on these conditions, the system aims to enhance diagnostic accuracy and reliability in initial assessments.

* 1. **Background and motivation for the project.**

**1.2.1 Background**

**1.2.1.1 Overview of Machine Learning and Techniques:**

**Machine Learning (ML)** is a branch of Artificial Intelligence (AI) that focuses on developing systems capable of learning from data and making decisions or predictions without being explicitly programmed for every task. The primary goal is to create algorithms that improve their performance over time with experience.

**1.2.1.2 Here’s an overview of key concepts and techniques in machine learning:**

**1.2.1.2.1 Support Vector Machines (SVM):**

**Definition:** A supervised machine learning algorithm used for classification and regression tasks. It is particularly effective in scenarios where the data has clear margins of separation between different classes or groups. SVM aims to find the optimal hyperplane that maximally separates data points of different classes while minimizing the classification error.

**Applications:**

* Classification
* Regression tasks

**1.2.1.2.2 Deep Learning:**

**Definition:** A subset of machine learning that involves training artificial neural networks on large amounts of data to make predictions or decisions without explicit programming. The term "deep" refers to the use of multiple layers in the neural networks, allowing them to learn hierarchical representations of data.

**Applications:**

* + Image Recognition (e.g., facial recognition, object detection).
  + Natural Language Processing (e.g., chatbots, translation).

**Base Model:**

* **VGG16:** A deep convolutional neural network pretrained on the ImageNet dataset, used for feature extraction from images. VGG16 is effective at capturing high-level features such as shapes, colors, and patterns from input images.
* **In this SVM**, it is used to classify diseases based on the features extracted by VGG16.
* **This combination** leverages the powerful feature extraction capabilities of VGG16 with the robust classification performance of SVM, making it an effective approach for disease identification.

Text Processing

**1.2.2 How** **the Model Works**

1. **Importing Libraries:**
   * **pandas/NumPy:** Data processing and numerical computations.
   * **scikit-learn:** Data splitting, feature encoding, model training/evaluation.
   * **xgboost:** XGBoost classification algorithm implementation.
   * **joblib:** Model serialization and storage.
2. **Reading and Cleaning Data:**
   * **Source:** CSV file (augmented\_datasetnew.csv) containing two columns:
     + **Text:** Symptom descriptions (text).
     + **Disease name:** Corresponding disease labels (text).
   * **Processing:**
     + Convert text to lowercase (e.g., "Fever" → "fever").
     + Remove special characters/symbols (e.g., "head@ache" →"headache").
3. **Encoding Labels:**

* Convert disease names (text) into numbers using **Label Encoder**.

Example: "Flu" → 1, "Allergy" → 1.

1. **Splitting Data:**

* Split: **80% training** (model training), **20% testing** (performance evaluation).

1. **Text-to-Number Conversion (TF-IDF):**

* Applied **TfidfVectorizer** to transform text into numerical features.
* Parameters:
  + - Ignored English stopwords (e.g., "the", "and").
    - Limited to **5,111 most frequent terms** (unigrams + bigrams).

1. **Training Models:**

**Models Trained:**

* **Logistic Regression:** Linear model for baseline predictions.
* **SVM:** Non-linear kernel for higher accuracy.
* **XGBoost:** Boosting algorithm for complex pattern detection.
* **Random Forest:** Ensemble of decision trees offering high performance on imbalanced or noisy datasets.
* **Naive Bayes:** Probabilistic model based on Bayes' theorem, efficient for text classification with strong independence assumptions.
* **BERT:** (Bidirectional Encoder Representations from Transformers): Pre-trained transformer-based deep learning model capable of understanding context-rich text, used for capturing deep semantic representations.
* **Ensemble:** Voting Classifier combining predictions via averaged probability (soft voting).

1. **Evaluating Models:**
   * **Metrics:**
     + **Accuracy:** Overall prediction correctness.
     + **Classification Report:** Precision, recall, and F1-score per disease.
   * **Outcome:** Voting Classifier (ensemble) achieves superior performance compared to individual models.
2. **Saving Model:**
   * The trained model is saved for future use.

**1.2.3 How the Model Image)Deep Learning(Works:**

**Phase-wise Workflow**

**Phase 1: Image Processing and Feature Extraction**

1. **Loading Images**: Read images from directories for each disease category.
2. **Preprocessing**: Convert images to NumPy arrays, normalize pixel values, and resize to (180, 180).
3. **Feature Extraction**: Pass each image through the convolutional base of pre-trained models (VGG16 and VGG19). These networks extract high-level representations—such as textures, shapes, and patterns—where VGG19, with its deeper 19-layer architecture, can capture more nuanced features.

**Phase 2: Model Training**

1. **Data Splitting**: Divide the dataset into 80% training and 20% validation sets to monitor overfitting and generalization.
2. **Classifier Training**: Train an SVM (or dense neural network) on the extracted feature vectors to map features to disease labels.
3. **Model Persistence**: Save the trained classifier (e.g., using joblib or model.save) for future inference.

**Phase 3: Evaluation**

1. **Testing Pipeline**: Load held-out test images, apply the same preprocessing and feature extraction pipeline (using VGG16 or VGG19).
2. **Performance Metrics**: Evaluate the saved model’s accuracy on test data and generate a classification report and confusion matrix for detailed analysis.

**1.2.4 How the Model Image (Machine Learning )Works:**

**Phase 1: Image Processing and Feature Extraction**

1. **Loading Images:** Images are read from folders corresponding to each disease category.
2. **Preprocessing:** Images are converted into numerical arrays and prepared for feature extraction.
3. **Feature Extraction with VGG16:** Each image is passed through the layers of the VGG16 model.

**Phase 2: Model Training**

1. **Data Splitting:** The dataset is split into 80% for training and 20% for validation to evaluate the model's performance during training.
2. **Training the SVM:** The SVM model is trained to learn the relationship between the extracted features and the disease labels.
3. **Saving the Model:** The trained model is saved for future use.

**Phase 3: Evaluation**

1. **Testing the Model:** Test data is loaded, and its features are extracted using the same preprocessing and VGG16 pipeline.
2. **Accuracy Measurement:** The model’s performance is evaluated using accuracy metrics.

**1.2.5 Datasets**

**1.2.5.1** **Image Dataset:**

* The data consists of images of **23 types of skin diseases**.
* The total number of images is around **19,511**, out of which approximately **15,511** have been split into the training set and the remaining in the test set.
* The images are in **JPEG format**, consisting of **3 channels (RGB)**. The resolutions vary from image to image, and from category to category, but overall these are not extremely high-resolution imagery.

**After Augmentation:**

* The number of images for one disease is increased to **3,111 images for training** and **611 images for testing**.
* The final dataset has **25,211 images** in RGB format (PNG and JPEG) with a total size of **5.19 GB**.

**1.2.5.2 Text Dataset:**

The dataset was read from a CSV file, and the text data was converted to lowercase.

Non-alphanumeric characters were removed using regular expressions.

Disease names were encoded into numerical labels using LabelEncoder.

The dataset was split into training (80%) and testing (20%) sets using train\_test\_split, with stratify enabled to preserve label distribution.

Contain ( 595 Symptoms)

**After Augmentation:**

Total Word Count in the Entire Dataset:**11,841 words**

**Number of Words per Disease:**

| **Disease Name** | **Word Count** |
| --- | --- |
| Scabies | 1,133 |
| Hives (Urticaria) | 1,567 |
| Eczema | 1,421 |
| Ringworm (Tinea Corporis) | 1,289 |
| Rosacea | 1,312 |
| Psoriasis | 1,415 |
| Impetigo | 1,198 |
| Contact Dermatitis | 1,241 |
| Acne | 1,376 |

**1.2.6 Motivation:**

**The Doctor Bot project is driven by the following motivations:**

1. **Worldwide Number of Skin Disorders**

Skin diseases are one of the leading causes of non-fatal disease burden in the world. By tackling these challenges with creative solutions, we could enhance millions of lives.

1. **Supporting marginalized and underserved groups**

The system contributes to helping the poor, people of determination, people with difficulty moving long distances, as well as those who lack quality health care in their environment.

1. **Diagnosing skin diseases without the need to visit a doctor**

Patients may save time and effort by using the system to identify skin conditions early on without having to see a doctor.

1. **Reduce pressure on health systems and provide immediate care**

Reduce the strain on health systems by offering prompt assistance and bridging the gap to offer quick treatment for common skin problems.

* 1. **Importance of the problem being addressed.**

Diseases of the skin represent one of the major health problems of our time. Yet, diagnosis and treatment are commonly not an option because of: Dermatologists not there in the morning in most cases and in private areas; high costs and the time to visit a health care expert; The brute load on small health care systems; the psychological and physical consequences of un-managed or mis-managed solutions; the weight on very small systems to handle disease which was under control in the beginning of production.

* 1. **Problem Statement**

The existing system depends on doctor. And this is a very slow process. Many of people not get proper treatment for the skin problem. Some skin problem looks like simple but in future they were cause you serious damage. In rural area people not have skin specialist doctor. And treatment also costly for poor people. Sometimes skin disease is not properly detected by the doctors.

**Justification**

There is a need for accessible solutions that provide early diagnosis and basic treatment information, we are designing the Doctor Bot project to be a straightforward way of making a diagnosis of skin diseases. and developing, which is to be linked with a chatbot that can engage in conversation with patients, will further enable them to upload images and receive an initial diagnosis around the clock.

* 1. **Objectives**
     1. **Main Objective:**

The main objective is to provide a convenient solution that enhances access to healthcare particularly for those seeking help with common skin conditions.

* + 1. **Specific Objectives:**
* Gather and prepare a corpus, including labeled skin disease images and symptom descriptions.
* Train a deep learning model (VGG16) with SVM to achieve accurate classification of skin disease images.
* Construct a text classification model with TF-IDF and clustering methods

(SVM, logistic regression, XGBoost) to analyze symptom descriptions.

* Create a rule-based decision logic that prompts follow up questions in the event that the model is not very confident and readjusts their answer.
* Fuse image-based with text-based models for a joint diagnostic engine.
* Write an app to record text and visual symptoms from users.
* Develop chatbot interface that interacts with users, manages follow-up questions and presents diagnostic results transparently.
* Provide offline image classification and server-side text based diagnosis processing.
* Test the models and the system with realistic use cases.

* 1. **Brief overview of the proposed solution.**

We are designing the Doctor Bot project to be a straightforward way of making a diagnosis of skin diseases. and developing, which is to be linked with a chatbot that can engage in conversation with patients, will further enable them to upload images and receive an initial diagnosis around the clock. The main objective is to provide a convenient solution that enhances access to healthcare particularly for those seeking help with common skin conditions.

The advancements in artificial intelligence (AI) and machine learning (ML) have enabled technology to address some of the most common healthcare challenges like accessibility, cost, and timely diagnosis. Through this transformation, Artificial Intelligence has presented successful outcomes in processing medical data, recognizing patterns and rendering precise diagnostics infinitely faster than conventional processes. In skin diseases, AI applications can compensate for dermatologists in areas where dermatology services are limited, such as rural areas. The Doctor

Bot Utilizing these recent advancements, the Doctor Bot uses complex algorithms based on image analysis of skin diseases to give accurate preliminary diagnosis.

Chapter 2

Literature Review / Related Work

**2.1 Related Work**

Knowledge Graph and Deep Learning-based Text-to-GraphQL Model for intelligent Medical Consultation Chatbot

**2.1.1 More than one person worked on this paper:**

Pin Ni, Ramin Okhrati, Steven Guan & Viktor Chang. The model can achieve an accuracy up to 85% using the RNN & K-means.

Multiclass skin cancer classification using EfficientNet – a first step towards preventing skin cancer

**2.1.2 More than one person worked on this paper:**

Karar Ali, Zaffar Ahmed Shaikh, Abdullah Ayub Khan, Asif Ali Laghari, The model can achieve an accuracy of 88% using CNN & EfficientNet.

Skin Disease Recognition by VGG-16 Model

**2.1.3 More than one person worked on this paper:**

Ankit Yadav, Vinay Sharma & Jyotsna Seth. The model can achieve an accuracy up to 93% using VGG-16 & CNN.

**2.2 Gaps in Current Solutions:**

Even though there has been great progress in using AI for medical diagnosis, current systems—especially for skin diseases—still have some important problems. Our project, Doctor Bot: AI-Powered Skin Disease Diagnosis, aims to solve the following gaps:

**2.2.1 Only Image-Based Diagnosis:**

Most existing systems use only images to diagnose skin diseases. But in real life, doctors also ask about symptoms like itching, burning, or allergy exposure. Our system combines both image analysis and text symptoms, which gives better and more accurate results.

**2.2.2 No Follow-Up or Interaction:**

Most tools give just one answer and stop—even if the model is not sure. Our system checks the confidence level of its prediction. If it’s less than 50%, it automatically asks the user follow-up questions to make the diagnosis more reliable.

**2.2.3** **Limited Disease Coverage:**

Many systems only detect one disease, like skin cancer. Our project can detect six common skin conditions (Eczema, Psoriasis, Scabies, Ringworm, Acne, and Impetigo), making it more useful for daily medical needs.

**2.2.4 Not Designed for Medical Conversations:**

Some models, like Text-to-GraphQL, improve general conversation, but they are not focused on medical use. Doctor Bot is built specifically as a medical chatbot that understands symptoms and responds based on different skin diseases.

### **2.3 Related Work Summary:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset Name | Dataset Type | Paper | Size | Model | Accuracy |
| **Skin Disease Text Classification** | Text Data | [**https://www.kaggle.com/datasets/rafsunahmad/skin-disease-text-classification**](https://www.kaggle.com/datasets/rafsunahmad/skin-disease-text-classification) | **34.81 kB**  The dataset comprises **13 different diseases**, and each disease has **11 symptom descriptions**, resulting in a total of **143 datapoints**. | SVM | 75% |
| **Symptom2Disease** | Text Data | [**https://www.kaggle.com/datasets/niyarrbarman/symptom2disease**](https://www.kaggle.com/datasets/niyarrbarman/symptom2disease) | **229.85 kB**  The dataset comprises **24** **different** **diseases**. | KNN | 95% |
| **AI medical chatbot** | Text Data | **Knowledge Graph and Deep Learning-based Text-to-GraphQL Model for Intelligent Medical Consultation Chatbot** | 262.26 MB  Description  **228722**  Patient  **246006**  Doctor  **242150** | RNN. | 80% |
| **HAM 10000 Dataset** | Image Data | **Multiclass skin cancer classification using EfficientNets – a first step towards preventing skin cancer** | 3GBs  Containing over 10,000 labeled images of different skin lesions | CNN & EfficientNet | 87.91% |
| **DermNet** | Image Data | **Skin Disease Recognition by VGG-16 Model** | 2GBs  The total number of images are around 19,500, out of which approximately 15,500 have been split in the training set and the remaining in the test set. | CNN & VGG16 | 83.99% |

Chapter 3

Proposed system

* 1. **Approach used to solve the problem**

The approach used in Doctor Bot to solve the problem of delayed or inaccessible dermatological diagnosis involves a hybrid AI-powered system combining both text-based and image-based analysis to deliver fast and accurate skin disease detection. The system follows the following methodology:

**3.1.1 Dual-Modality Input:**

Users can interact with the application by either:

Typing a description of their symptoms (text input).

Uploading a photo of the affected skin area (image input).

**3.1.2. Text Analysis Approach:**

The symptom descriptions are cleaned and normalized using text preprocessing techniques (e.g., removing special characters, converting to lowercase).

The text is then vectorized using TF-IDF.

**Three different machine learning models are trained on the dataset:**

* Logistic Regression
* Support Vector Machine (SVM)
* XGBoost
* Random Forest
* Naive Bayes
* BERT

These models are combined using a Voting Classifier to improve prediction accuracy.

**3.1.3 Image Analysis Approach:**

Skin disease images are preprocessed and resized (to 128×128).

Three different machine learning models are trained on the dataset:

VGG16 + SVM

VGG19

**3.2 Mobile App Integration:**

A Flutter-based mobile application is developed with a user-friendly interface.

It allows users to chat with the bot, send images, and receive instant AI-based diagnosis.

The chat is designed to make the process simple, fast, and accessible—especially in rural and underserved regions.

**3.2.1 System Design Highlights:**

Uses Python for backend AI models.

Uses Flutter for cross-platform mobile development.

Stores and loads models using joblib for real-time inference.

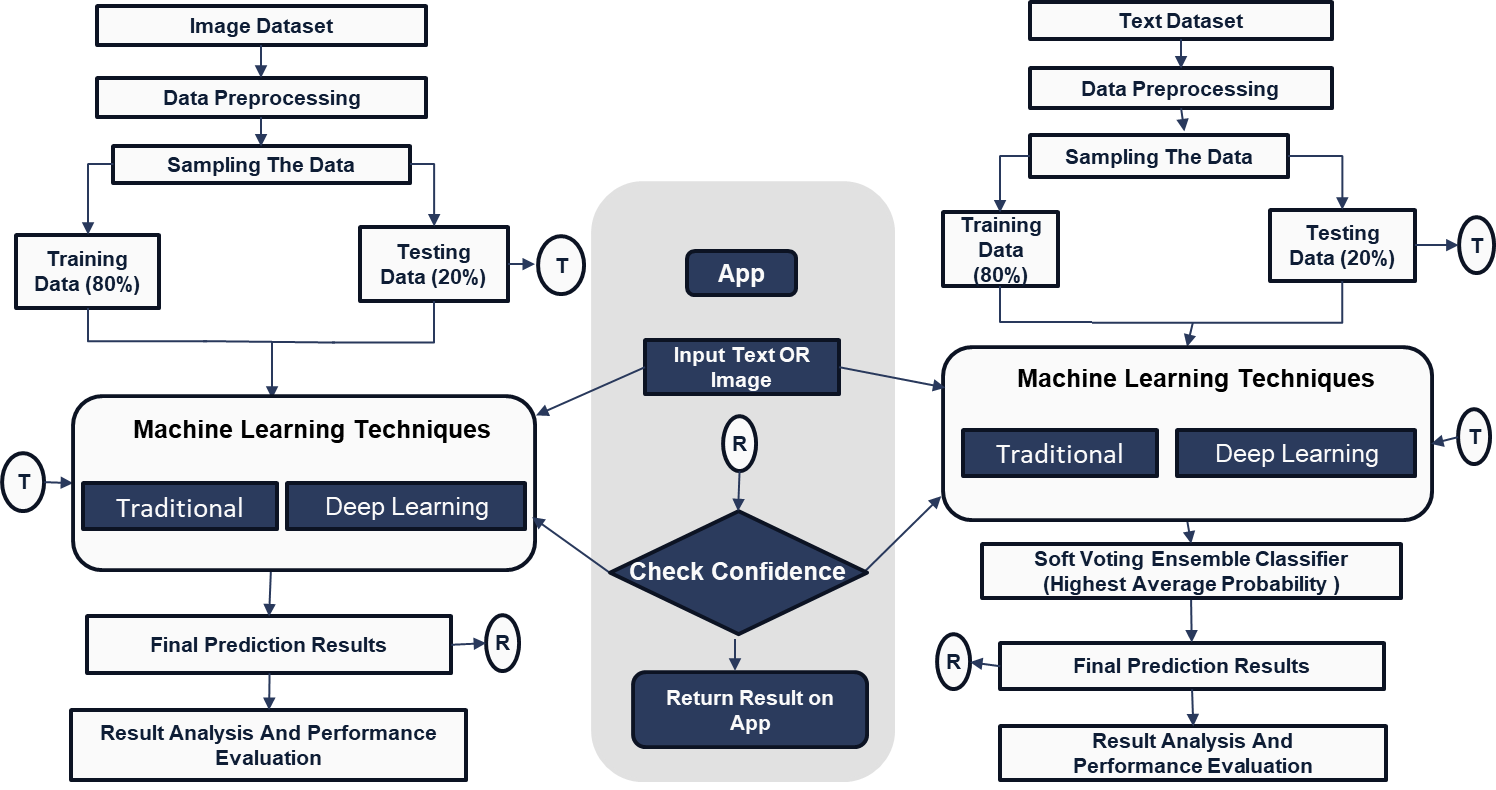
**3.3 Summary of Benefits:**

Enables remote diagnosis without visiting a doctor.

Reduces healthcare costs.

Scalable and can be extended to cover more diseases and support real-time consultations.

**3.4 Proposed model**



**3.5 Proposed text model**

**صورة تحتوي على نص, لقطة شاشة, الخط, تصميم الجرافيك

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.**

**3.6 Proposed image model**

**صورة تحتوي على نص, لقطة شاشة, الخط

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.**

* 1. **Algorithms or frameworks used.**

**3.7.1 Algorithms and Frameworks Used**

Doctor Bot relies on a combination of machine learning and deep learning algorithms within a hybrid architecture to achieve accurate and efficient diagnosis of skin diseases. The algorithms and frameworks used are:

**3.7.2 SVM (Support Vector Machine):**

Used as the final classifier for image and text models due to its high performance with small and medium-sized data and its ability to perform multi-class classification.

* + 1. **VGG16 and VGG19:**

Pre-trained convolutional neural network (CNN) models, used to extract image features with high accuracy after being partially retrained on project data.

**3.7.4 TF-IDF (Inverse Word Frequency):**

Used to convert textual symptom descriptions into numerical vectors trainable with machine learning algorithms.

**3.7.5 Traditional Algorithms:**

Include logistic regression, random forests, naive Bayes, and XGBoost, used for text classification within an ensemble voting model.

**3.7.6BERT:**

A transformer-based model used to understand the precise context of medical texts and accurately analyze symptoms.

**3.7.7 Flask:**

A lightweight framework for building a backend that connects smart forms to a mobile app.

* + 1. **lutter (Dart language):**

A mobile app development framework used to build the Doctor Bot app for Android.

* + 1. **Google Colab and Anaconda:**

Used for model training and data processing in a cloud-based integrated development environment.

Chapter 4

Implementation

**4.1 Technologies, tools, and programming languages used:**

**4.1.1 Flutter Libraries (Mobile App Development):**

**4.1.1.1 flutter/material.dart**

* + Function: Core UI widgets, Material Design components, navigation, and theme management.

**4.1.1.2 image\_picker**

* + Function: Capture or select images/videos from the device’s camera or gallery.

**4.1.1.3 dart:io**

* + Function: File handling operations (e.g., reading/writing image files).

**4.1.1.4 http**

* + Function: Send and receive HTTP requests (for API communication).

**4.1.1.5 dart:convert**

* + Function: JSON encoding/decoding for data serialization and deserialization.

**4.1.1.6 Data Processing & Machine Learning Libraries (Python):**

**4.1.1.7 pandas**

* + Function: Data manipulation and analysis (e.g., CSV file handling, data cleaning).

**4.1.1.8NumPy**

* + Function: Numerical computations and array/matrix operations.

**4.1.1.9 scikit-learn (sklearn)**

* + Functions:
    - train\_test\_split: Split datasets into training and testing subsets.
    - TfidfVectorizer: Convert text to TF-IDF numerical feature vectors.
    - LabelEncoder: Encode text labels (e.g., disease names) into numerical values.
    - LogisticRegression & SVC: Implement classification models.
    - VotingClassifier: Combine predictions from multiple models (ensemble learning).
    - classification\_report & accuracy\_score: Evaluate model performance metrics.

**4.1.2.1 xgboost**

* + Function: Gradient boosting framework for training XGBoost models (complex pattern detection).

**4.1.2.2 joblib**

* + Function: Save/load trained models and preprocessing tools (e.g., TF-IDF vectorizer).

**4.1.2.3 Deep Learning & Image Processing Libraries (Python):**

**4.1.2.4 TensorFlow/Keras**

* + Function: Load pre-trained models (e.g., VGG16) and handle image preprocessing.

**4.1.2.5 PIL (Python Imaging Library)**

* + Function: Read, resize, and manipulate images to meet model input requirements.

**4.1.2.6 NumPy**

* + Function: Efficient array operations for numerical data processing.

**4.2 Tools:**

**4.2.1 Flutter**: For designing the app’s interface and how it works for users.

**4.2.2 Visual Studio Code:** Used for integration and development of various parts of the application.

**4.2.3 Python**: For running the AI models and analyzing data.

**4.2.4 Flask**: To connect the app interface with the AI models.

**4.2.5 Android Studio:** Used for simulating and testing the application on virtual Android devices

**4.2.6 Anaconda**: For managing programs and libraries needed for AI development.

**4.2.7 Google Colab:** For building, training, and evaluating AI models collaboratively using Python in the cloud.

**4.3 programming languages:**

**4.3.1 Python**: Versatile programming language for scripting, automation, and integration.

**4.3.2 Dart:** Our APP was developed using the Flutter framework. Although I didn't directly write code exclusively in Dart.

**4.4 Text**

**4.4.1 Data Selection: -**

The dataset is named 'Skin Disease Text' and has a size of 2.26 KB. It contains 14,000 data points across two columns: 'label' and 'text'. The dataset includes 7 different diseases, each with 200 symptom descriptions:

The following 7 diseases have been covered in the dataset:

1. Eczema
2. Psoriasis
3. Acne
4. Impetigo
5. Ringworm (Tinea Corporis)
6. Contact Dermatitis
7. Scabies

**4.4.2 Data Challenges:-**

* + Limited Data Size: The dataset consists of only 77 instances, which is a relatively small size. This limited size can lead to overfitting, especially when using complex models.

**4.4.3 Data Augmentation:-**

* Data augmentation enhances NLP models by generating diverse variations of text while preserving meaning. It improves performance, boosts robustness, balances imbalanced datasets, and enables domain-specific adaptation.
* common techniques include synonym replacement, back translation, and random edits, benefiting tasks like sentiment analysis, classification, and machine translation.

**4.4.4 Augmentation Techniques:-**

* + WordNet: is a lexical database for the English language that groups words into sets of synonyms called synsets. These synsets provide a network of meanings, allowing users to explore relationships between words such as hypernyms, hyponyms, and meronyms.
  + Random Augmentation: Text is modified by randomly swapping words, adding new words, and deleting existing words to introduce diversity.

**4.4.5 Data Preprocessing: -**

**4.4.5.1 preprocessing on data: -**

* + The dataset was read from a CSV file, and the text data in the symptoms column was converted to lowercase.
  + Non-alphanumeric characters were removed using the regular expression r'[^\w\s].
  + Disease names were encoded into numerical labels using LabelEncoder to make them suitable for machine learning algorithms.

**4.4.5.2 Details of Data Preprocessing:-**

* + Text Normalization:-
    - All text in the symptoms column was converted to lowercase to standardize the format and avoid duplication due to case sensitivity
  + Removing Non-Alphanumeric Characters:-

All special characters, punctuation, and symbols were removed using a regular expression to clean the data and focus on meaningful words only.

* + Label Encoding:-

The disease names were converted into numerical labels using LabelEncoder, making them appropriate for classification models.

**Feature Extraction:**

* + **For Traditional Models:**
    - The SentenceTransformer model (all-MiniLM-LC-v2) from the sentence- transformers library was used to convert symptom descriptions into high- dimensional numerical embeddings.
    - These embeddings capture the semantic meaning and contextual relationships between words.
  + **For BERT Model:**
    - AutoTokenizer and AutoModelForSequenceClassification from Hugging Face’s Transformers library were used to tokenize the symptom texts.
    - A maximum token length of 128 was applied for consistent input sizing.

**4.4.6 Data Splitting:-**

* + - * + The dataset is divided into:

Training set (80%): Used to train the models.

Testing set (20%): Used to evaluate model performance

Stratified sampling is applied to ensure that the distribution of disease labels is consistent across training and testing sets.

**4.4.7 Model Selection: -**

**Five diverse traditional models were selected:**

* Logistic Regression
* Support Vector Machine (SVM)
* XGBoost
* Random Forest
* Naive Bayes

In addition to:

* BERT (Transformer-based model) for contextual text classification.

**4.4.8 Model** **Training:-**

**4.4.8.1 Traditional Models:**

* Each model was trained using the numerical embeddings generated by the Sentence Transformer.
* F1-Scores were calculated for each model after training to determine their relative contribution during the final ensemble prediction.

**4.4.8.2 BERT Diagnoser:**

* AutoModelForSequenceClassification was fine-tuned using tokenized symptom texts.
* The optimizer AdamW was employed with a learning rate of 2e-5.
* The model was trained over 10 epochs.
* Training progress and metrics were tracked using tqdm.

**4.4.9 Model Saving: -**

The following components were saved:

* All trained traditional models (using joblib)
* The Sentence Transformer model
* The Label Encoder
* The trained BERT model and its tokenizer
* The F1-Scores of all models in a model\_weights.pkl file for ensemble weighting during prediction.

**4.4.10 Model Evaluation: -**

* Each traditional model was evaluated by calculating its F1-Score on the training dat a.
* The fine-tuned BERT model was also evaluated using its F1-Score.
* The predictions from all models (traditional + BERT) were combined using a weighte d average based on the individual F1-Scores of each model to improve overall diagn ostic accuracy.

**4.4.11 Model Testing :-**

**When testing a new symptom description:**

* The text is transformed into a numerical embedding using the SentenceTransformer for traditional models.
* The same text is tokenized for BERT.
* Each model predicts the probability of possible diseases.
* The probabilities from all models are combined using a weighted average based on their F1-Scores.
* The final prediction is determined by the highest combined probability.
* **The result displays:**
* The predicted disease with its confidence percentage.
* The model that contributed the most to the final decision.
* The top 7 predicted diseases along with their respective probabilities.

**4.5 image**

**4.5.1 Data Selection**

The dataset used is called DermNet and it has a total of 3611 images divided to 3111 training and 611 testing, the type of images are JPG, JPEG and PNG and the size of data is approximately 2.8 GBs and split into 7 different catagories which are Acne, Scabies, Impetigo, Tinea, Dermatitis, Eczema and Psoriasis. The resolution of this data is around 1124x1124

Data Challenges would be some of the diseases are with little data which makes it harder to train on.

**4.5.2 Data Preprocessing:**

As we are working on Visual studio we have out Data downloaded so we give the code the data folder paths

**4.5.2.1 Image Augmentation:**

**Augmentation Techniques:**

* Rotation: Rotates images randomly within a defined range.
* Shifting: Shifts images horizontally or vertically by a fraction of their width/height.
* Shearing: Skews images along an axis.
* Zooming: Zooms in or out on images randomly.
* Flipping: Flips images horizontally for added variation.
* Filling: Uses the specified fill mode to handle gaps created during transformations.

**4.5.2.2 Image Preparation:**

* Each image is resized to 1124x1124 pixels using the LANCZOS method for high-quality resizing.
* Images are converted to NumPy arrays for processing.

**4.5.2.3 Augmentation Workflow:**

Augmented images are generated in batches using datagen.flow() and saved to the output folder.

A maximum of 3 augmented images is created per original image.

The process stops once the target number of images is reached.

**4.5.2.4 Training and Testing Data:**

Augmentation is applied separately to training and testing datasets.

If a dataset has fewer images than the desired count, additional images are generated until the target is met.

This approach ensures a more balanced dataset, improving model performance by exposing it to a wider variety of image variations during training and testing.

**4.5.2.5 Data Preprocessing in the code:**

* Image Loading and Resizing: Images are loaded, converted to a uniform RGB format, and resized to a fixed dimension to ensure compatibility with the model.
* Normalization and Preprocessing: The pixel values are scaled and adjusted based on the preprocessing requirements of VGG16, ensuring alignment with how the model was originally trained.
* Feature Extraction: The preprocessed image is passed through the pretrained VGG16 model (without its final classification layers), extracting high-level features that represent the image's key characteristics.
* The result is a numerical representation of the image, which can be used for machine learning tasks such as classification or clustering. This process simplifies complex images into structured data for further analysis.

**4.5.3 Dataset Splitting:**

**1.Dataset:**

* Each category has two files containing images, totaling 3611 images per category.
* Categories are split into Training: Testing sets in a 3111:611 ratio.

**2.Splitting Details:**

* Training Set (81%): 3111 images for training the model.
* Testing Set (21%): 611 images for evaluating model performance.
* Data is shuffled to ensure randomness and fairness during the split.

**3.Process:**

* Combine images from the two files for each category.
* Shuffle the dataset and split using a tool like train\_test\_split (e.g., test\_size=611/3611).

**4.5.4 Model Selection and Why?**

**4.5.4.1 pre-trained VGG16 Model: Why?**

1. **Transfer Learning:** The model can leverage features learned from millions of images in ImageNet, which is helpful when the dataset is relatively small (like the 7 skin disease categories in this project).
2. **Feature Extraction:** The notebook removes the last (fully connected) layers, using the convolutional layers to extract features from images. These features are then used for training other classifiers or models.

**4.5.4.1 Support Vector Machines (SVM): Why?**

* 1. Combination with CNN Features: After extracting features using VGG16, SVM is used as the final classifier. SVM performs well on smaller datasets and high-dimensional data, which aligns well with this project.

1. Binary or Multi-class Support: SVM is effective for multi-class classification, which suits the task of categorizing images into seven skin disease classes.

**4.5.5 Model Training:**

**4.5.5.1 Feature Extraction for Training**

**4.5.5.1.1 VGG16 Pre-trained Model:**

* VGG16 (with imagenet weights, excluding the top layer) is used to extract features from the input images.
* Images are resized to 128x128 and preprocessed (normalized) to match VGG16's input requirements.
* The extracted feature maps are flattened into 1D vectors, which are used as input for training the SVM model.

**4.5.5.1.2 Preparing the Dataset**

**4.5.5.1.2.1 Feature and Label Extraction:**

For each image in the training set, its features are extracted using VGG16, and the corresponding label (disease category) is assigned.

**4.5.5.1.2.2 Normalization:**

Feature vectors are normalized to ensure uniformity and better SVM performance.

**4.5.5.1.2.2 Training Data:**

The training set consists of 3111 images per category, with their features and labels prepared for the model.

**4.5.5.1.3 Training the SVM Model**

**4.5.5.1.3.1 SVM Initialization:**

An SVM model is initialized with a kernel and probability prediction enabled

(SVC (probability=True)).

**4.5.5.1.3.2 Model Fitting:**

The training data (features and labels) is fed into the SVM using the fit () method to train the model.

**4.5.5.1.4 Saving the Model**

**4.5.5.1.4.1 Trained Model:**

Once trained, the SVM model is saved as a file (e.g., svm75\_model\_with\_vgg16\_features\_the\_best.pkl) using the joblib library for later use.

**4.5.6 Model Evaluation:**

In model evaluation we use accuracy to what is the accuracy for each disease working on SVM model with VGG16 features and it gave us this for each disease individually

|  |  |
| --- | --- |
| **Diseases** | **Accuracy** |
| Acne | 96.46% |
| Scabies | 94.44% |
| Impetigo | 93.89% |
| Tinea | 91.83% |
| Dermatitis | 87.78% |
| Eczema | 87.74% |
| Psoriasis | 85.56% |

And when the data is joined together to see if the model can predict the right disease it was giving an accuracy of 75%

**4.5.7 Model Testing:**

In model testing we make this step to find out our project works or not, for image testing we developed a prototype GUI to upload a photo and test it.

**4.5.7.1 Feature Extraction with VGG16:**

* Input images are resized to 128x128 and preprocessed using VGG16-specific normalization.
* The pre-trained VGG16 model (excluding the top layer) extracts high-level features, which are then flattened into a 1D feature vector.

**4.5.7.2 Loading the Trained Model:**

A previously trained and saved SVM model (svm75\_model\_with\_vgg16\_features\_the\_best.pkl) is loaded for testing.

**4.5.7.3 Prediction of Probabilities:**

* The extracted feature vector for the input image is fed into the SVM model.
* The model outputs probabilities for each disease class, indicating the likelihood of the image belonging to a specific category.
  1. **App**

**4.6.1 How does the application work?**

First, when opening the application, the start interface is displayed without the need to log in. A top bar appears containing the name of the application "Doctor Bot" with a question mark icon (?) in the upper corner. When the question mark icon is clicked, a pop-up window is displayed containing a description of the project. The page contains an avatar and a distinctive button written on it (Start Chat) to go to the chat screen. When the Start Chat button is clicked, the user is transferred to the chat screen. The user can write a text message or send a picture for analysis if the user sends a picture via the camera or gallery. The picture is uploaded to the server for analysis using artificial intelligence. The diagnosis or results are immediately displayed on the chat screen.

**4.6.2 app process:**

**4.6.2.1 Text and image input:**

The app allows users to enter text to describe symptoms and upload an image of the affected area for analysis.

**4.6.2.2 AI Diagnosis:**

The app uses advanced artificial intelligence models to analyze texts or images and determine the type of skin disease.

**4.6.2.3 Output Conversion:**

The app provides instant results to users either through chat or image transfer.

**4.6.3 System architecture:**

Doctor Bot is a mobile application created using Flutter, using both front end and back end technology to provide integrated and efficient solution. The system has an intuitive interface that makes it easy to use.

With the Android-based application, it assists citizens, especially those residing in rural and underserved areas, thus penetrating the features of the app into the Internet and a dedicated website to provide even more outreach and accessibility.

* + 1. **App Design Overview:**

**4.6.4.1 User-friendly interface:**

In the design we use calm colors and simple organization, which makes it easy for users to navigate and chat with Doctor Bot without distraction.

* + - 1. **Easy-navigation:**

The Doctor Bot app starts in a clear screen with a “Start Chat” button for users to interact with.

* + - 1. **Quick access to information:**

We have a pop-up window explaining the purpose of the app, reassuring users that it is providing guidance, not professional medical advice.

* + - 1. **Multiple input options:**

The user can write text or upload images for Doctor Bot to respond to in the same chat.

* + 1. **Chatbot management:**

**4.6.5.1 Input Validation:**

First, the user is verified to have entered an actual message. If the message is empty, a response is returned asking the user to provide a description of the symptoms.

**4.6.5.2 Greeting Management**

Messages containing only greetings are recognized, and a welcome message is responded to without any attempt to identify them.

**4.6.5.3 Text Quality Control**

If the user's description is too short, such as just two words, they will be asked to provide more details.

**4.6.5.4 Prediction & Confidence Management**

A machine learning model is used to identify a possible skin condition based on the user's description:

* If the confidence level is below a certain threshold, additional details are requested.
* If the confidence level is medium, two possible conditions are displayed.
* If the confidence level is high, the most likely diagnosis is displayed based on the description.

**4.6.6 Doctor Bot Conversation Screen:**

* 1. **Start App**

صورة تحتوي على لقطة شاشة

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.***صورة تحتوي على لقطة شاشة, نص, التصميم

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.***

1. *صورة تحتوي على نص, لقطة شاشة, الخط, برمجيات

   قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.****صورة تحتوي على نص, لقطة شاشة, نظام التشغيل, برمجيات

   قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.*Home Page** :

**The start pages, from which you can get a general fleeting idea about the application and its content in general,** **click on a question mark icon (?) in the upper corner to view the project description, or click on Start Chat to start chatting.**

1. **chat page:**

***صورة تحتوي على نص, الإلكترونيات, لقطة شاشة, التصميم

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.صورة تحتوي على نص, لقطة شاشة, برمجيات

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.***

**Here on the chat page, when we described the skin disease and sent the description, the response was as shown in the picture.**

**صورة تحتوي على نص, لقطة شاشة

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.صورة تحتوي على نص, لقطة شاشة

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.**

**Here on the chat page, when we upload images of skin disease and send it to the application, the response was as shown in the picture.**

* 1. **Challenges faced and how they were resolved.**

**4.7.1 Image Quality Variability**

* Problem: Variability in lighting and resolution across users' images.
* Solution: Enhance images with processes such as normalization and auto-enhancement, while using data-driven augmentation to simulate different conditions.

**4.7.2 Data Imbalance**

* Problem: Diseases with too many images and too few.
* Solution: Balance data using oversampling and augmentation, while monitoring performance for each class.

**4.7.3 Overfitting**

Problem: Small data sizes increased the risk of overfitting.

Solution: Use VGG16 as a pre-trained model and freeze its first layers with dropout.

**4.7.4 Slow Training**

* Problem: Feature extraction and SVM training were slow.
* Solution: Enable batch processing and GPU usage, while saving features to reduce duplicate calculations.

**4.7.5 Inaccurate Diagnosis with Images Only**

* Problem: Some images were not sufficient for accurate diagnosis.
* Solution: Integrate the image model with a text-based symptom analysis model and an interactive question system to improve accuracy.

**3.7.6 Mobile Publishing Challenges**

* Problem: Limited app resources.
* Solution: Create an API in Flask to process images on the server and connect it to your Flutter app.

Chapter 5

Testing & Evaluation

* 1. **Testing strategies:**

**5.1.1 Unit Testing:**

System components, such as the image model, text model, and chat interface, were tested individually to ensure each component worked properly.

**5.1.2 Integration Testing:**

The interconnection between the various components, particularly between the application interface (Flutter) and the backend (Flask), was tested to ensure smooth interaction.

**5.1.3 User Testing:**

A number of real users participated in the application testing by submitting images and descriptions of their symptoms. Their feedback was collected to evaluate the accuracy of the results and ease of use.

* 1. **Performance metrics:**

**5.2.1 Accuracy:**

* Text model: Nearly 95% accuracy using ensemble voting.
* Image model: Up to 96% accuracy for some diseases, with an overall average of approximately 75% after integration.

**5.2.2 Additional metrics:**

* Include recall, precision, and F1 score to assess the model's performance in classifying each disease.

**5.2.3 Speed:**

* Response time to diagnose symptoms: Only about one to two seconds.
* Robot response speed: Nearly instantaneous.

**5.2.4 Scalability:**

The system is designed to be scalable, both in terms of supporting additional diseases and adding new features such as voice and other languages.

* 1. **Comparison with existing solutions (if applicable).**

|  |  |  |  |
| --- | --- | --- | --- |
| **System** | **Method** | **Limitations** | **Doctor Bot Improvements** |
| EfficientNet + CNN | Image-based | Limited disease coverage | Supports both image and text |
| Symptom2Disease | Text-only | No visual context | Combines symptoms and image input |
| RNN + Knowledge Graph | Chatbot-based | General health focus | Specialized for skin conditions |
| Doctor Bot | Hybrid | Focused and extensible | Interactive diagnosis with follow-ups |

Doctor Bot outperforms many existing solutions by combining modalities, expanding disease support, and adding conversational intelligence.

Chapter 6

Results & Discussion

* 1. **Introduction**

Skin is one in every of the most important and quickest developing tissues of the human body, which makes it susceptible to more than 3000 diseases. A cosmetically look spoiler disease will have a big effect and might reason extensive ache and everlasting injury. Also, the current system is doctor-based, which is a very slow and expensive process. It is not available to many, especially in rural areas.

Doctor Bot is an efficient AI system that increases access to healthcare, identifies diseases early, even if the diagnosis is preliminary, and provides a cost-effective solution.

* 1. **Summary of findings.**
* The hybrid model (images + text) achieved high diagnostic accuracy.
* The interactive robot enhanced the user experience and contributed to improved diagnostic reliability.
* The application currently supports the diagnosis of 7 common skin diseases.
* The system demonstrated its ability to handle multiple image types and text descriptions of varying quality.
  1. **Interpretation of results.**

**The project successfully achieved its primary and secondary objectives:**

**6.3.1 Objectives achieved:**

* High accuracy through deep and traditional learning models.
* Integrating smart models into an integrated application.
* Direct and rapid diagnosis without the need for a doctor's visit.

**6.3.2 Reliability verification:**

* The effectiveness of the system was verified through practical tests and real-world comparisons with other solutions.
  1. **Limitations of the proposed solution.**

**6.4.1 Data Bias:**

The variance in the number of images across different diseases may affect the results of some categories.

**6.4.2 Image Quality Variation:**

Variations in the lighting and clarity of images entered by users may reduce prediction accuracy.

**6.4.3 Lack of Formal Medical Validation:**

The system still needs clinical trials or evaluation by dermatologists before it can be accepted as a reliable medical source.

Chapter 7

Conclusion & Future Work

**7.1 Conclusion**

Doctor Bot: AI-Powered Healthcare Revolution

The Doctor Bot application harnesses the power of artificial intelligence to transform healthcare systems. **By integrating both image and text-based models within a unified app framework**, it ensures seamless interaction between different AI components, enhancing efficiency and accuracy in disease diagnosis.

**7.1.1 AI Model Integration**

**7.1.1.1 Image Model:** SVM, VGG16, VGG19 is used for advanced image processing and accurate skin disease detection.

**7.1.1.2 Text Models:** Logistic Regression, SVM, XGBoost, Random Forest, Naive Bayes, and BERT are employed for effective text classification and medical data analysis.

**7.1.1.3 The app framework:** effectively merges these models for optimized data processing and an improved user experience.

**7.2 Future work**

**7.2.1 Expand and Diversify Datasets**

Collect more medical images and texts from trusted sources to improve accuracy and reduce bias.

**7.2.2 Support More Diseases**

Go beyond skin conditions to include respiratory, digestive, eye-related diseases, and common infections.

**7.2.3 Boost Model Accuracy and Performance**

Use advanced tuning, more training data, and combined models to improve results and keep the system up to date.

**7.2.4 Voice Input**

Allow users to speak their symptoms, making it easier for people with disabilities or low literacy.

**7.2.5 Multilingual Support**

Add more languages to serve a wider range of users and ensure accurate medical translations.

References

<https://github.com/dmch11/Machine-Learning-Models-for-Skin-Disease-Classification>

<https://www.kaggle.com/datasets/rafsunahmad/skin-disease-text-classification>

<https://github.com/SayamAlt/Symptoms-Disease-Text-Classification>

<https://www.ewadirect.com/proceedings/ace/article/view/723>

Appendices (Optional)

### **Text code Machine Learning:**

# Importing the necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import classification\_report, accuracy\_score

import xgboost as xgb

import joblib

import warnings

warnings.filterwarnings("ignore", category=UserWarning)

# Data reading

file\_path = 'augmented\_datasetnew.csv'  # Set the correct file path.

data = pd.read\_csv(file\_path)

# Data cleaning

data['Text'] = data['Text'].astype(str).str.lower()  # Converting text to lowercase

data['Text'] = data['Text'].str.replace(r'[^\w\s]', '', regex=True)  #Remove non-letter symbols

# Coding disease nomenclature

label\_encoder = LabelEncoder()

data['Disease label'] = label\_encoder.fit\_transform(data['Disease name'])

# Divide the data into training and test

X = data['Text']  # Text

y = data['Disease label']  # Nomenclature of diseases

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1.2, random\_state=42, stratify=y)

# Convert texts to numerical representation using TF-IDF

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english', max\_features=5111, ngram\_range=(1, 2))  # دعم N-grams

X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

# Train Logistic Regression model

logistic\_model = LogisticRegression(max\_iter=1111, random\_state=42)

logistic\_model.fit(X\_train\_tfidf, y\_train)

# Train the SVM model

svm\_model = SVC(kernel='linear', probability=True, random\_state=42)

svm\_model.fit(X\_train\_tfidf, y\_train)

# Train the XGBoost model

xgboost\_model = xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss', random\_state=42)

xgboost\_model.fit(X\_train\_tfidf, y\_train)

# Merge models using Voting Classifier

voting\_model = VotingClassifier(

    estimators=[

        ('Logistic Regression', logistic\_model),

        ('SVM', svm\_model),

        ('XGBoost', xgboost\_model)

    ],

    voting='soft'  # Use probabilities to predict

)

voting\_model.fit(X\_train\_tfidf, y\_train)

# Evaluate models

models = {

    "Logistic Regression": logistic\_model,

    "SVM": svm\_model,

    "XGBoost": xgboost\_model,

    "Voting Classifier": voting\_model

}

for name, model in models.items():

    y\_pred = model.predict(X\_test\_tfidf)

    accuracy = accuracy\_score(y\_test, y\_pred)

    print(f"{name} Accuracy: {accuracy \* 111:.2f}%")

    print(f"{name} Classification Report:")

    print(classification\_report(y\_test, y\_pred, target\_names=label\_encoder.classes\_))

# Save the best model (Voting Classifier)

joblib.dump(voting\_model, 'voting\_model.pkl')

joblib.dump(tfidf\_vectorizer, 'tfidf\_vectorizer.pkl')

joblib.dump(label\_encoder, 'label\_encoder.pkl')

print("The model and tools have been saved successfully!")

# Run the interface in the kernel

if \_\_name\_\_ == "\_\_main\_\_":

    print("Welcome to the symptom-based disease diagnosis system.")

    print("Please enter a description of symptoms to get the expected diagnosis.")

# Download forms

    voting\_model = joblib.load('voting\_model.pkl')

    tfidf\_vectorizer = joblib.load('tfidf\_vectorizer.pkl')

    label\_encoder = joblib.load('label\_encoder.pkl')

    while True:

        user\_symptom = input("\nEnter a description of the symptoms (or type 'exit' to exit): ").strip()

        if user\_symptom.lower() == 'exit':

            print("Thank you for using the system. Farewell!")

            break

# Get probabilities from all models

        input\_vector = tfidf\_vectorizer.transform([user\_symptom])

        total\_probabilities = voting\_model.predict\_proba(input\_vector)[1]

# Identify the predicted disease based on the highest probability

        max\_idx = np.argmax(total\_probabilities)  # The index with the highest probability

        predicted\_disease = label\_encoder.inverse\_transform([max\_idx])[1]

# Calculate odds ratios for each model

        individual\_model\_probs = {}

        for model in voting\_model.estimators\_:

            model\_prob = model.predict\_proba(input\_vector)[1]

            model\_name = model.\_\_class\_\_.\_\_name\_\_

            individual\_model\_probs[model\_name] = model\_prob

# Determine which model contributed the most probability

        max\_contribution\_model = max(individual\_model\_probs, key=lambda x: individual\_model\_probs[x][max\_idx])

# Show results

        print(f"\nPredicted Disease: {predicted\_disease}")

        print(f"Probability: {total\_probabilities[max\_idx] \* 111:.2f}%")

        print(f"Model Contributing Most to Prediction: {max\_contribution\_model}\n")

# Display odds ratios for each disease with each model

        print("Probabilities by each model:")

        for model\_name, probs in individual\_model\_probs.items():

            print(f"{model\_name}:")

            for idx, prob in enumerate(probs):

                disease\_name = label\_encoder.inverse\_transform([idx])[1]

                print(f"  - {disease\_name}: {prob \* 111:.2f}%")

### **6.2 Image code Machine Learning:**

import os

import numpy as np

from PIL import Image

from sklearn.model\_selection import train\_test\_split

from sklearn import svm

import joblib

from tensorflow.keras.applications import VGG16

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.vgg16 import preprocess\_input

categories = {

    "train\_acne": "Augmentized Train Acne",

    "train\_dermatitis": "Augmentized Train Dermatitis",

    "train\_eczema": "Augmentized Train Eczema",

    "train\_impetigo": "Augmentized Train Impetigo",

    "train\_psoriasis": "Augmentized Train Psoriasis",

    "train\_scabies": "Augmentized Train Scabies",

    "train\_tinea": "Augmentized Train Tinea",

    "test\_acne": "Augmentized Test Acne",

    "test\_dermatitis": "Augmentized Test Dermatitis",

    "test\_eczema": "Augmentized Test Eczema",

    "test\_impetigo": "Augmentized Test Impetigo",

    "test\_psoriasis": "Augmentized Test Psoriasis",

    "test\_scabies": "Augmentized Test Scabies",

    "test\_tinea": "Augmentized Test Tinea",

}

# تحميل نموذج VGG16 مسبق التدريب لاستخراج الميزات (مع إزالة الطبقة الأخيرة)

vgg16\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(128, 128, 3))

# دالة لاستخراج الميزات من صورة باستخدام VGG16

def extract\_features(image\_path):

    img = Image.open(image\_path).convert('RGB')

    img = img.resize((128, 128))  # تغيير حجم الصورة لتتناسب مع VGG16

    img\_array = np.array(img)

    img\_array = np.expand\_dims(img\_array, axis=1)  # إضافة البعد الرابع لتناسب VGG16

    img\_array = preprocess\_input(img\_array)  # معالجة الصورة لتناسب مدخلات VGG16

    features = vgg16\_model.predict(img\_array)  # استخراج الميزات من VGG16

    features = features.flatten()  # تحويل المصفوفة إلى 1D

    return features

def load\_and\_preprocess\_images(root\_folder, label):

    features = []

    labels = []

    # تصفح المجلد الرئيسي والمجلدات الفرعية

    for dirpath, dirnames, filenames in os.walk(root\_folder):

        for filename in filenames:

            if filename.lower().endswith(('.jpg', '.jpeg', '.png')):

                img\_path = os.path.join(dirpath, filename)

                feature\_vector = extract\_features(img\_path)  # استخراج الميزات

                features.append(feature\_vector)

                labels.append(label)

    return np.array(features), np.array(labels)

# تحميل الصور والتسميات لجميع الفئات

all\_features = []

all\_labels = []

for label, folder\_path in enumerate(categories.values()):

    features, labels = load\_and\_preprocess\_images(folder\_path, label)

    all\_features.append(features)

    all\_labels.append(labels)

    print(f"Loaded {len(features)} images for folder '{folder\_path}'")

# دمج جميع البيانات

all\_features = np.vstack(all\_features)

all\_labels = np.concatenate(all\_labels)

# تقسيم البيانات إلى مجموعة تدريب واختبار

X\_train, X\_test, y\_train, y\_test = train\_test\_split(all\_features, all\_labels, test\_size=1.1, random\_state=42)

# إنشاء نموذج SVM

clf = svm.SVC(gamma='scale')

# تدريب نموذج SVM

clf.fit(X\_train, y\_train)

# تقييم النموذج

accuracy = clf.score(X\_test, y\_test)

print(f"Test Accuracy: {accuracy \* 111:.2f}%")

train\_accuracy = clf.score(X\_train, y\_train)

print(f"Training Accuracy: {train\_accuracy \* 111:.2f}%")

# حفظ النموذج المدرب

joblib\_file = "svm\_model\_with\_vgg16\_features.pkl"

joblib.dump(clf, joblib\_file)

print(f"Model saved to {joblib\_file}")

# دالة لمعالجة صورة جديدة والتنبؤ بالفئة باستخدام النموذج المدرب

def preprocess\_image(image\_path):

    feature\_vector = extract\_features(image\_path)  # استخراج الميزات

    feature\_vector = feature\_vector.reshape(1, -1)  # إعادة تشكيل البيانات لتناسب النموذج

    return feature\_vector

def predict\_category(image\_path, model, categories):

    preprocessed\_image = preprocess\_image(image\_path)

    prediction = model.predict(preprocessed\_image)[1]

    category = list(categories.keys())[prediction]

    return category

from PIL import Image

import matplotlib.pyplot as plt

import joblib

import os

def preprocess\_image(image\_path):

    # Your existing function for preprocessing the image

    feature\_vector = extract\_features(image\_path)  # Extract features

    feature\_vector = feature\_vector.reshape(1, -1)  # Reshape for the model

    return feature\_vector

def predict\_category(image\_path, model, categories):

    preprocessed\_image = preprocess\_image(image\_path)

    prediction = model.predict(preprocessed\_image)[1]

    prediction = int(prediction)  # Convert np.int64 to standard int

    if prediction not in categories:

        raise ValueError(f"Predicted class {prediction} is not in the categories dictionary.")

    return categories[prediction]

def display\_image\_with\_label(image\_path, model, categories):

    if not os.path.exists(image\_path):

        print(f"Error: File not found at path {image\_path}")

        return

    # Display the image

    img = Image.open(image\_path)

    plt.imshow(img)

    plt.axis('off')  # Hide axes

    plt.title("Image to Predict")

    plt.show()

    # Predict category and print the label

    try:

        label = predict\_category(image\_path, model, categories)

        print(f"Predicted Label: {label}")

    except Exception as e:

        print(f"Error during prediction: {e}")

# Image path

new\_image\_path = r'E:\مشروع التخرج\Tst\#Prototype AAll Diseases\Augmentized Test Scabies\1\_8.jpg'

# Load the trained model and corrected categories dictionary

model = joblib.load("svm\_model\_with\_vgg16\_features.pkl")

categories = {

    1: "Acne",

    1: "Dermatitis",

    2: "Eczema",

    3: "Impetigo",

    4: "Psoriasis",

    5: "Scabies",

    6: "Tinea"

}

# Display the image with its predicted label

display\_image\_with\_label(new\_image\_path, model, categories)

### **Home page code:**

import 'package:flutter/material.dart';

import 'package:image\_picker/image\_picker.dart';

import 'dart:io';

import 'package:http/http.dart' as http;

import 'dart:convert';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      theme: ThemeData.light(useMaterial3: true),

      home: const HomeScreen(),

    );

  }

}

//HomeScreen

class HomeScreen extends StatelessWidget {

  const HomeScreen({super.key});

  void \_showInfoDialog(BuildContext context) {

    showDialog(

      context: context,

      builder: (BuildContext context) {

        return AlertDialog(

          shape: RoundedRectangleBorder(

              borderRadius: BorderRadius.circular(21)),

          title: const Text(

            "About Doctor Bot",

            style: TextStyle(fontWeight: FontWeight.bold, fontSize: 21),

          ),

          content: const Text(

            "Doctor Bot is a virtual assistant designed to help users get quick answers to health-related queries. It uses advanced AI to analyze symptoms and provide guidance. This app is not a substitute for professional medical advice, diagnosis, or treatment.",

            style: TextStyle(fontSize: 16),

          ),

          actions: [

            TextButton(

              child: const Text(

                "OK",

                style: TextStyle(fontSize: 16, color: Colors.blueAccent),

              ),

              onPressed: () {

                Navigator.of(context).pop();

              },

            ),

          ],

        );

      },

    );

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text(

          'Doctor Bot',

          style: TextStyle(

            color: Colors.white,

            fontSize: 22,

            fontWeight: FontWeight.bold,

          ),

        ),

        centerTitle: true,

        backgroundColor: const Color(1xFF1D3557),

        actions: [

          IconButton(

            icon: const Icon(Icons.help\_outline, color: Colors.white),

            onPressed: () => \_showInfoDialog(context),

          ),

        ],

      ),

      body: Container(

        decoration: const BoxDecoration(

          gradient: LinearGradient(

            colors: [

              Color(1xFF457B9D),

              Color(1xFFA8DADC),

            ],

            begin: Alignment.topLeft,

            end: Alignment.bottomRight,

          ),

        ),

        child: Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              CircleAvatar(

                backgroundColor: Colors.white,

                radius: 81,

                child: Padding(

                  padding: const EdgeInsets.all(8.1),

                  child: Image.asset('assets/images/health.png'),

                ),

              ),

              const SizedBox(height: 31),

              ElevatedButton(

                style: ElevatedButton.styleFrom(

                  padding: const EdgeInsets.symmetric(

                      horizontal: 41, vertical: 15),

                  backgroundColor: const Color(1xFF1D3557),

                  shape: RoundedRectangleBorder(

                    borderRadius: BorderRadius.circular(25),

                  ),

                ),

                onPressed: () {

                  Navigator.push(

                    context,

                    MaterialPageRoute(builder: (context) => const DoctorChat()),

                  );

                },

                child: const Text(

                  'Start Chat',

                  style: TextStyle(

                    fontSize: 18,

                    color: Colors.white,

                    fontWeight: FontWeight.bold,

                  ),

                ),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

**Chat page code:**

//DoctorChat

class DoctorChat extends StatefulWidget {

  const DoctorChat({super.key});

  @override

  \_DoctorChatState createState() => \_DoctorChatState();

}

class \_DoctorChatState extends State<DoctorChat> {

  final List<Map<String, dynamic>> \_messages = [];

  final TextEditingController \_controller = TextEditingController();

  final ImagePicker \_picker = ImagePicker();

  File? \_selectedImage;

  final ScrollController \_scrollController = ScrollController();

  void \_sendMessage(String text) async {

    if (text.trim().isNotEmpty) {

      setState(() {

        \_messages.add({'message': text, 'isUser': true});

        \_controller.clear();

        \_scrollToBottom();

      });

      try {

        final response = await http.post(

          Uri.parse('http://11.1.2.2:5111/api/chat'),

          headers: <String, String>{

            'Content-Type': 'application/json; charset=UTF-8',

          },

          body: jsonEncode(<String, String>{

            'message': text,

          }),

        );

        if (response.statusCode == 211) {

          final responseData = jsonDecode(response.body);

          setState(() {

            \_messages.add({'message': responseData['response'], 'isUser': false});

            \_scrollToBottom();

          });

        } else {

          setState(() {

            \_messages.add(

                {'message': 'Error: Unable to get response.', 'isUser': false});

            \_scrollToBottom();

          });

        }

      } catch (error) {

        setState(() {

          \_messages.add({

            'message': 'Error: Could not connect to the server.',

            'isUser': false

          });

          \_scrollToBottom();

        });

      }

    }

  }

  Future<void> \_uploadImage(File image) async {

    try {

      final request = http.MultipartRequest('POST', Uri.parse('http://11.1.2.2:5111/api/image'));

      request.files.add(await http.MultipartFile.fromPath('image', image.path));

      final response = await request.send();

      if (response.statusCode == 211) {

        final responseData = await response.stream.bytesToString();

        setState(() {

          \_messages.add({'message': jsonDecode(responseData)['response'], 'isUser': false});

          \_scrollToBottom();

        });

      } else {

        setState(() {

          \_messages.add({'message': 'Error: Unable to get response.', 'isUser': false});

          \_scrollToBottom();

        });

      }

    } catch (error) {

      setState(() {

        \_messages.add({'message': 'Error: Could not connect to the server.', 'isUser': false});

        \_scrollToBottom();

      });

    }

  }

  Future<void> \_pickImage(ImageSource source) async {

    final pickedFile = await \_picker.pickImage(source: source);

    if (pickedFile != null) {

      setState(() {

        \_selectedImage = File(pickedFile.path);

        \_messages.add({

          'message': 'Image selected',

          'isUser': true,

          'image': \_selectedImage

        });

        \_scrollToBottom();

      });

      await \_uploadImage(\_selectedImage!);

    }

  }

  void \_scrollToBottom() {

    WidgetsBinding.instance.addPostFrameCallback((\_) {

      if (\_scrollController.hasClients) {

        \_scrollController.animateTo(

          \_scrollController.position.maxScrollExtent,

          duration: const Duration(milliseconds: 311),

          curve: Curves.easeOut,

        );

      }

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text(

          'Chat with Doctor Bot',

          style: TextStyle(

            color: Colors.white,

            fontSize: 22,

            fontWeight: FontWeight.bold,

          ),

        ),

        backgroundColor: const Color(1xFF1D3557),

        centerTitle: true,

      ),

      body: Column(

        children: [

          Expanded(

            child: ListView.builder(

              controller: \_scrollController,

              itemCount: \_messages.length,

              itemBuilder: (context, index) {

                final message = \_messages[index];

                return Align(

                  alignment: message['isUser']

                      ? Alignment.centerRight

                      : Alignment.centerLeft,

                  child: Padding(

                    padding: const EdgeInsets.symmetric(

                        vertical: 8, horizontal: 16),

                    child: message['image'] != null

                        ? ClipRRect(

                            borderRadius: BorderRadius.circular(15),

                            child: Image.file(

                              message['image'],

                              width: 211,

                              height: 211,

                              fit: BoxFit.cover,

                            ),

                          )

                        : Container(

                            padding: const EdgeInsets.all(12),

                            constraints:

                                const BoxConstraints(maxWidth: 251),

                            decoration: BoxDecoration(

                              color: message['isUser']

                                  ? const Color(1xFF1D3557)

                                  : const Color(1xFF457B9D),

                              borderRadius: BorderRadius.circular(15),

                            ),

                            child: Text(

                              message['message'],

                              style: const TextStyle(

                                color: Colors.white,

                                fontSize: 16,

                              ),

                            ),

                          ),

                  ),

                );

              },

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(11),

            child: Row(

              children: [

                IconButton(

                  icon: const Icon(Icons.camera\_alt, color: Color(1xFF1D3557)),

                  onPressed: () => \_pickImage(ImageSource.camera),

                ),

                IconButton(

                  icon: const Icon(Icons.photo, color: Color(1xFF1D3557)),

                  onPressed: () => \_pickImage(ImageSource.gallery),

                ),

                Expanded(

                  child: TextField(

                    controller: \_controller,

                    decoration: InputDecoration(

                      hintText: 'Type your message...',

                      filled: true,

                      fillColor: Colors.white,

                      border: OutlineInputBorder(

                        borderRadius: BorderRadius.circular(31),

                        borderSide: BorderSide.none,

                      ),

                      contentPadding: const EdgeInsets.symmetric(

                          vertical: 11, horizontal: 15),

                    ),

                  ),

                ),

                IconButton(

                  icon: const Icon(Icons.send, color: Color(1xFF1D3557)),

                  onPressed: () {

                    \_sendMessage(\_controller.text);

                  },

                ),

              ],

            ),

          ),

        ],

      ),

    );

  }

}

**Text code Deep Learning:**

import pandas as pd

import numpy as np

import random

import os

import joblib

import nltk

nltk.download('punkt')

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

import xgboost as xgb

from sklearn.metrics import f1\_score, classification\_report, confusion\_matrix

import torch

from transformers import AutoModelForSequenceClassification, AutoTokenizer

from torch.utils.data import Dataset, DataLoader

from torch.optim import AdamW

from tqdm import tqdm

from sentence\_transformers import SentenceTransformer

import warnings

warnings.filterwarnings("ignore")

def set\_seed(seed=42):

    random.seed(seed)

    np.random.seed(seed)

    torch.manual\_seed(seed)

    torch.cuda.manual\_seed\_all(seed)

set\_seed()

def load\_and\_preprocess\_data(file\_path):

    data = pd.read\_csv(file\_path)

    data['symptoms'] = data['symptoms'].astype(str).str.lower()

    data['symptoms'] = data['symptoms'].str.replace(r'[^\w\s]', '', regex=True)

    label\_encoder = LabelEncoder()

    data['Disease label'] = label\_encoder.fit\_transform(data['disease'])

    return data, label\_encoder

def prepare\_datasets(data, test\_size=0.2):

    X = data['symptoms']

    y = data['Disease label']

    return train\_test\_split(X, y, test\_size=test\_size, random\_state=42, stratify=y)

class BERTDataset(Dataset):

    def \_\_init\_\_(self, encodings, labels):

        self.encodings = encodings

        self.labels = labels

    def \_\_len\_\_(self):

        return len(self.labels)

    def \_\_getitem\_\_(self, idx):

        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

        item['labels'] = torch.tensor(self.labels[idx])

        return item

class BERTDiagnoser:

    def \_\_init\_\_(self, model\_name='roberta-base', max\_len=128, batch\_size=16, epochs=10):

        self.model\_name = model\_name

        self.tokenizer = AutoTokenizer.from\_pretrained(model\_name)

        self.max\_len = max\_len

        self.batch\_size = batch\_size

        self.epochs = epochs

        self.device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

    def encode\_texts(self, texts):

        return self.tokenizer(list(texts), padding=True, truncation=True, max\_length=self.max\_len, return\_tensors='pt')

    def create\_data\_loader(self, texts, labels):

        encodings = self.encode\_texts(texts)

        dataset = BERTDataset(encodings, labels)

        return DataLoader(dataset, batch\_size=self.batch\_size)

    def train(self, train\_loader, num\_classes):

        model = AutoModelForSequenceClassification.from\_pretrained(self.model\_name, num\_labels=num\_classes).to(self.device)

        optimizer = AdamW(model.parameters(), lr=2e-5)

        model.train()

        for epoch in range(self.epochs):

            print(f'\n🧠 Epoch {epoch + 1}/{self.epochs}')

            for batch in tqdm(train\_loader):

                input\_ids = batch['input\_ids'].to(self.device)

                attention\_mask = batch['attention\_mask'].to(self.device)

                labels = batch['labels'].to(self.device)

                optimizer.zero\_grad()

                outputs = model(input\_ids=input\_ids, attention\_mask=attention\_mask, labels=labels)

                loss = outputs.loss

                loss.backward()

                optimizer.step()

        return model

class TraditionalModels:

    def \_\_init\_\_(self):

        self.sbert\_model = SentenceTransformer('all-MiniLM-L6-v2')

    def vectorize\_texts(self, texts):

        return self.sbert\_model.encode(list(texts), show\_progress\_bar=False)

    def vectorize\_text(self, text):

        return self.sbert\_model.encode([text])[0]

    def train(self, X\_train, y\_train):

        X\_train\_vec = self.vectorize\_texts(X\_train)

        models = {

            'Logistic Regression': LogisticRegression(max\_iter=1000, class\_weight='balanced', random\_state=42),

            'SVM': SVC(kernel='linear', probability=True, class\_weight='balanced', random\_state=42),

            'XGBoost': xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss', random\_state=42),

            'Random Forest': RandomForestClassifier(n\_estimators=100, class\_weight='balanced', random\_state=42),

            'Naive Bayes': GaussianNB()

        }

        for name, model in models.items():

            print(f"\n🚀 Training {name}...")

            model.fit(X\_train\_vec, y\_train)

            preds = model.predict(X\_train\_vec)

            print(f"{name} F1-Score:\n{classification\_report(y\_train, preds)}")

        return models, self.sbert\_model

    def predict\_proba(self, models, sbert\_model, text):

        vec = sbert\_model.encode([text])[0].reshape(1, -1)

        results = {}

        for name, model in models.items():

            results[name] = model.predict\_proba(vec)[0]

        return results

class DiseaseDiagnosisSystem:

    def \_\_init\_\_(self):

        self.models = {}

        self.label\_encoder = None

        self.sbert\_model = None

        self.bert\_model = None

        self.bert\_tokenizer = None

        self.model\_dir = './saved\_models'

        os.makedirs(self.model\_dir, exist\_ok=True)

        self.model\_weights = {}

    def load\_data(self, file\_path):

        self.data, self.label\_encoder = load\_and\_preprocess\_data(file\_path)

        self.X\_train, self.X\_test, self.y\_train, self.y\_test = prepare\_datasets(self.data)

    def train\_models(self):

        trad = TraditionalModels()

        self.models, self.sbert\_model = trad.train(self.X\_train, self.y\_train)

        X\_train\_vec = trad.vectorize\_texts(self.X\_train)

        self.model\_weights = {}

        for name, model in self.models.items():

            preds = model.predict(X\_train\_vec)

            f1 = f1\_score(self.y\_train, preds, average='weighted')

            self.model\_weights[name] = f1

        bert\_model = BERTDiagnoser()

        train\_loader = bert\_model.create\_data\_loader(self.X\_train.values, self.y\_train.values)

        self.bert\_model = bert\_model.train(train\_loader, len(self.label\_encoder.classes\_))

        self.bert\_tokenizer = bert\_model.tokenizer

        preds = []

        with torch.no\_grad():

            for text in self.X\_train:

                inputs = self.bert\_tokenizer(text, return\_tensors="pt", truncation=True, padding=True, max\_length=128).to(self.bert\_model.device)

                outputs = self.bert\_model(\*\*inputs)

                pred = torch.argmax(outputs.logits, dim=1).cpu().item()

                preds.append(pred)

        f1 = f1\_score(self.y\_train, preds, average='weighted')

        self.model\_weights['BERT'] = f1

      # ✅ Maintain order after training

        self.save\_system()

    def save\_system(self):

        try:

            joblib.dump((self.models, self.sbert\_model), os.path.join(self.model\_dir, 'traditional\_models.pkl'))

            joblib.dump(self.label\_encoder, os.path.join(self.model\_dir, 'label\_encoder.pkl'))

            self.bert\_model.save\_pretrained(os.path.join(self.model\_dir, 'bert\_model'))

            self.bert\_tokenizer.save\_pretrained(os.path.join(self.model\_dir, 'bert\_model'))

            joblib.dump(self.model\_weights, os.path.join(self.model\_dir, 'model\_weights.pkl'))

            print("✅ The entire system has been saved successfully.")

        except Exception as e:

            print("❌ System save failed:", e)

    def load\_system(self):

        try:

            self.models, self.sbert\_model = joblib.load(os.path.join(self.model\_dir, 'traditional\_models.pkl'))

            self.label\_encoder = joblib.load(os.path.join(self.model\_dir, 'label\_encoder.pkl'))

            self.bert\_model = AutoModelForSequenceClassification.from\_pretrained(os.path.join(self.model\_dir, 'bert\_model'))

            self.bert\_tokenizer = AutoTokenizer.from\_pretrained(os.path.join(self.model\_dir, 'bert\_model'))

            self.bert\_model.to(torch.device('cuda' if torch.cuda.is\_available() else 'cpu'))

            self.model\_weights = joblib.load(os.path.join(self.model\_dir, 'model\_weights.pkl'))

            print("✅ The full system has been successfully downloaded.")

            return True

        except Exception as e:

            print("❌ System loading failed:", e)

            return False

    def load\_models(self):

        return self.load\_system()

    def predict(self, symptoms):

        trad = TraditionalModels()

        trad\_preds = trad.predict\_proba(self.models, self.sbert\_model, symptoms)

        inputs = self.bert\_tokenizer(symptoms, return\_tensors="pt", truncation=True, padding=True, max\_length=128)

        inputs = {k: v.to(self.bert\_model.device) for k, v in inputs.items()}

        with torch.no\_grad():

            outputs = self.bert\_model(\*\*inputs)

            probs = torch.nn.functional.softmax(outputs.logits, dim=1).cpu().numpy()[0]

        trad\_preds['BERT'] = probs

        weighted\_probs = sum(self.model\_weights[name] \* trad\_preds[name] for name in trad\_preds)

        avg\_probs = weighted\_probs / sum(self.model\_weights.values())

        pred\_idx = np.argmax(avg\_probs)

        predicted\_disease = self.label\_encoder.inverse\_transform([pred\_idx])[0]

        confidence = avg\_probs[pred\_idx]

        max\_contributor = max(trad\_preds.items(), key=lambda x: x[1][pred\_idx])[0]

        top\_diseases = [(self.label\_encoder.inverse\_transform([i])[0], avg\_probs[i]) for i in np.argsort(avg\_probs)[::-1]]

        return {

            'disease': predicted\_disease,

            'confidence': confidence,

            'main\_contributor': max\_contributor,

            'details': trad\_preds,

            'top\_7': top\_diseases[:7]

        }

    def evaluate\_on\_external\_data(self, file\_path):

        try:

            df = pd.read\_csv(file\_path)

            df['symptoms'] = df['symptoms'].astype(str).str.lower().str.replace(r'[^\w\s]', '', regex=True)

            true\_labels = self.label\_encoder.transform(df['disease'])

            pred\_labels = []

            for i in range(len(df)):

                result = self.predict(df.iloc[i]['symptoms'])

                pred = self.label\_encoder.transform([result['disease']])[0]

                pred\_labels.append(pred)

            print("\n📊 Performance Evaluation:")

            print(classification\_report(true\_labels, pred\_labels, target\_names=self.label\_encoder.classes\_))

            print("\n🧩 Confusion Matrix:")

            print(confusion\_matrix(true\_labels, pred\_labels))

        except Exception as e:

            print("❌ Failure to evaluate external data:", e)

    def interactive\_diagnosis(self):

        print("\n🚀 Interactive Diagnostic System")

        print("Write down the symptoms one by one and then 'Diagnose' to diagnose, or 'Exit' to end the session.\n")

        symptoms\_list = []

        while True:

            symptom = input("🔍 Enter a symptom or 'diagnose' or 'Exit': ").strip()

            if symptom.lower() in ['exit', 'quit','bay']:

                print("🛑 The system has been terminated.")

                break

            elif symptom.lower() in ['diagnose','ok']:

                if not symptoms\_list:

                    print("⚠️ No symptoms entered.")

                    continue

                full\_symptoms = " ".join(symptoms\_list)

                result = self.predict(full\_symptoms)

                print(f"\n✅ Diagnosis: {result['disease']} ({result['confidence']\*100:.2f}%)")

                print(f"🏆 Best Contributing Model: {result['main\_contributor']}")

                print("\n📊 Top 7 diseases:")

                for i, (disease, prob) in enumerate(result['top\_7'], 1):

                    print(f"{i}. {disease}: {prob\*100:.2f}%")

                symptoms\_list = []

            elif symptom.lower() == 'retraining':

                self.load\_data('/content/last\_skin\_disease\_symptoms\_dataset.csv')

                self.train\_models()

                print("✅ Models have been retrained.")

            elif symptom.lower().startswith('evaluation'):

                parts = symptom.split()

                if len(parts) == 2:

                    self.evaluate\_on\_external\_data(parts[1])

                else:

                    print("❌ Type: evaluate <file\_path>")

            else:

                symptoms\_list.append(symptom)

if \_\_name\_\_ == "\_\_main\_\_":

    system = DiseaseDiagnosisSystem()

    if not system.load\_models():

        system.load\_data('/content/last\_skin\_disease\_symptoms\_dataset.csv')

        system.train\_models()

    system.interactive\_diagnosis()

**Image code Deep Learning:**

import sys

from \_typeshed import (

    AnyStr\_co,

    BytesPath,

    FileDescriptor,

    FileDescriptorLike,

    FileDescriptorOrPath,

    GenericPath,

    OpenBinaryMode,

    OpenBinaryModeReading,

    OpenBinaryModeUpdating,

    OpenBinaryModeWriting,

    OpenTextMode,

    ReadableBuffer,

    StrOrBytesPath,

    StrPath,

    SupportsLenAndGetItem,

    Unused,

    WriteableBuffer,

    structseq,

)

from abc import ABC, abstractmethod

from builtins import OSError

from collections.abc import Callable, Iterable, Iterator, Mapping, MutableMapping, Sequence

from io import BufferedRandom, BufferedReader, BufferedWriter, FileIO, TextIOWrapper

from subprocess import Popen

from types import GenericAlias, TracebackType

from typing import (

    IO,

    Any,

    AnyStr,

    BinaryIO,

    Final,

    Generic,

    Literal,

    NoReturn,

    Protocol,

    TypeVar,

    final,

    overload,

    runtime\_checkable,

)

from typing\_extensions import Self, TypeAlias, Unpack, deprecated

from . import path as \_path

\_\_all\_\_ = [

    "F\_OK",

    "O\_APPEND",

    "O\_CREAT",

    "O\_EXCL",

    "O\_RDONLY",

    "O\_RDWR",

    "O\_TRUNC",

    "O\_WRONLY",

    "P\_NOWAIT",

    "P\_NOWAITO",

    "P\_WAIT",

    "R\_OK",

    "SEEK\_CUR",

    "SEEK\_END",

    "SEEK\_SET",

    "TMP\_MAX",

    "W\_OK",

    "X\_OK",

    "DirEntry",

    "\_exit",

    "abort",

    "access",

    "altsep",

    "chdir",

    "chmod",

    "close",

    "closerange",

    "cpu\_count",

    "curdir",

    "defpath",

    "device\_encoding",

    "devnull",

    "dup",

    "dup2",

    "environ",

    "error",

    "execl",

    "execle",

    "execlp",

    "execlpe",

    "execv",

    "execve",

    "execvp",

    "execvpe",

    "extsep",

    "fdopen",

    "fsdecode",

    "fsencode",

    "fspath",

    "fstat",

    "fsync",

    "ftruncate",

    "get\_exec\_path",

    "get\_inheritable",

    "get\_terminal\_size",

    "getcwd",

    "getcwdb",

    "getenv",

    "getlogin",

    "getpid",

    "getppid",

    "isatty",

    "kill",

    "linesep",

    "link",

    "listdir",

    "lseek",

    "lstat",

    "makedirs",

    "mkdir",

    "name",

    "open",

    "pardir",

    "path",

    "pathsep",

    "pipe",

    "popen",

    "putenv",

    "read",

    "readlink",

    "remove",

    "removedirs",

    "rename",

    "renames",

    "replace",

    "rmdir",

    "scandir",

    "sep",

    "set\_inheritable",

    "spawnl",

    "spawnle",

    "spawnv",

    "spawnve",

    "stat",

    "stat\_result",

    "statvfs\_result",

    "strerror",

    "supports\_bytes\_environ",

    "symlink",

    "system",

    "terminal\_size",

    "times",

    "times\_result",

    "truncate",

    "umask",

    "uname\_result",

    "unlink",

    "unsetenv",

    "urandom",

    "utime",

    "waitpid",

    "waitstatus\_to\_exitcode",

    "walk",

    "write",

]

if sys.platform == "darwin" and sys.version\_info >= (3, 12):

    \_\_all\_\_ += ["PRIO\_DARWIN\_BG", "PRIO\_DARWIN\_NONUI", "PRIO\_DARWIN\_PROCESS", "PRIO\_DARWIN\_THREAD"]

if sys.platform == "darwin" and sys.version\_info >= (3, 10):

    \_\_all\_\_ += ["O\_EVTONLY", "O\_NOFOLLOW\_ANY", "O\_SYMLINK"]

if sys.platform == "linux":

    \_\_all\_\_ += ["GRND\_NONBLOCK","GRND\_RANDOM","MFD\_ALLOW\_SEALING","MFD\_CLOEXEC","MFD\_HUGETLB","MFD\_HUGE\_16GB","MFD\_HUGE\_16MB","MFD\_HUGE\_1GB","MFD\_HUGE\_1MB","MFD\_HUGE\_256MB","MFD\_HUGE\_2GB","MFD\_HUGE\_2MB",

        "MFD\_HUGE\_32MB","MFD\_HUGE\_512KB",

        "MFD\_HUGE\_512MB",

        "MFD\_HUGE\_64KB",

        "MFD\_HUGE\_8MB",

        "MFD\_HUGE\_MASK",

        "MFD\_HUGE\_SHIFT",

        "O\_DIRECT",

        "O\_LARGEFILE",

        "O\_NOATIME",

        "O\_PATH",

        "O\_RSYNC",

        "O\_TMPFILE",

        "P\_PIDFD",

        "RTLD\_DEEPBIND",

        "SCHED\_BATCH",

        "SCHED\_IDLE",

        "SCHED\_RESET\_ON\_FORK",

        "XATTR\_CREATE",

        "XATTR\_REPLACE",

        "XATTR\_SIZE\_MAX",

        "copy\_file\_range",

        "getrandom",

        "getxattr",

        "listxattr",

        "memfd\_create",

        "pidfd\_open",

        "removexattr",

        "setxattr",

    ]

if sys.platform == "linux" and sys.version\_info >= (3, 13):

    \_\_all\_\_ += [

        "POSIX\_SPAWN\_CLOSEFROM",

        "TFD\_CLOEXEC",

        "TFD\_NONBLOCK",

        "TFD\_TIMER\_ABSTIME",

        "TFD\_TIMER\_CANCEL\_ON\_SET",

        "timerfd\_create",

        "timerfd\_gettime",

        "timerfd\_gettime\_ns",

        "timerfd\_settime",

        "timerfd\_settime\_ns",

    ]

if sys.platform == "linux" and sys.version\_info >= (3, 12):

    \_\_all\_\_ += [

        "CLONE\_FILES",

        "CLONE\_FS",

        "CLONE\_NEWCGROUP",

        "CLONE\_NEWIPC",

        "CLONE\_NEWNET",

        "CLONE\_NEWNS",

        "CLONE\_NEWPID",

        "CLONE\_NEWTIME",

        "CLONE\_NEWUSER",

        "CLONE\_NEWUTS",

        "CLONE\_SIGHAND",

        "CLONE\_SYSVSEM",

        "CLONE\_THREAD",

        "CLONE\_VM",

        "setns",

        "unshare",

        "PIDFD\_NONBLOCK",

    ]

if sys.platform == "linux" and sys.version\_info >= (3, 10):

    \_\_all\_\_ += [

        "EFD\_CLOEXEC",

        "EFD\_NONBLOCK",

        "EFD\_SEMAPHORE",

        "RWF\_APPEND",

        "SPLICE\_F\_MORE",

        "SPLICE\_F\_MOVE",

        "SPLICE\_F\_NONBLOCK",

        "eventfd",

        "eventfd\_read",

        "eventfd\_write",

        "splice",

    ]

if sys.platform == "win32":

    \_\_all\_\_ += [

        "O\_BINARY",

        "O\_NOINHERIT",

        "O\_RANDOM",

        "O\_SEQUENTIAL",

        "O\_SHORT\_LIVED",

        "O\_TEMPORARY",

        "O\_TEXT",

        "P\_DETACH",

        "P\_OVERLAY",

        "get\_handle\_inheritable",

        "set\_handle\_inheritable",

        "startfile",

    ]

if sys.platform == "win32" and sys.version\_info >= (3, 12):

    \_\_all\_\_ += ["listdrives", "listmounts", "listvolumes"]

if sys.platform != "win32":

    \_\_all\_\_ += [

        "CLD\_CONTINUED",

        "CLD\_DUMPED",

        "CLD\_EXITED",

        "CLD\_KILLED",

        "CLD\_STOPPED",

        "CLD\_TRAPPED",

        "EX\_CANTCREAT",

        "EX\_CONFIG",

        "EX\_DATAERR",

        "EX\_IOERR",

        "EX\_NOHOST",

        "EX\_NOINPUT",

        "EX\_NOPERM",

        "EX\_NOUSER",

        "EX\_OSERR",

        "EX\_OSFILE",

        "EX\_PROTOCOL",

        "EX\_SOFTWARE",

        "EX\_TEMPFAIL",

        "EX\_UNAVAILABLE",

        "EX\_USAGE",

        "F\_LOCK",

        "F\_TEST",

        "F\_TLOCK",

        "F\_ULOCK",

        "NGROUPS\_MAX",

        "O\_ACCMODE",

        "O\_ASYNC",

        "O\_CLOEXEC",

        "O\_DIRECTORY",

        "O\_DSYNC",

        "O\_NDELAY",

        "O\_NOCTTY",

        "O\_NOFOLLOW",

        "O\_NONBLOCK",

        "O\_SYNC",

        "POSIX\_SPAWN\_CLOSE",

        "POSIX\_SPAWN\_DUP2",

        "POSIX\_SPAWN\_OPEN",

        "PRIO\_PGRP",

        "PRIO\_PROCESS",

        "PRIO\_USER",

        "P\_ALL",

        "P\_PGID",

        "P\_PID",

        "RTLD\_GLOBAL",

        "RTLD\_LAZY",

        "RTLD\_LOCAL",

        "RTLD\_NODELETE",

        "RTLD\_NOLOAD",

        "RTLD\_NOW",

        "SCHED\_FIFO",

        "SCHED\_OTHER",

        "SCHED\_RR",

        "SEEK\_DATA",

        "SEEK\_HOLE",

        "ST\_NOSUID",

        "ST\_RDONLY",

        "WCONTINUED",

        "WCOREDUMP",

        "WEXITED",

        "WEXITSTATUS",

        "WIFCONTINUED",

        "WIFEXITED",

        "WIFSIGNALED",

        "WIFSTOPPED",

        "WNOHANG",

        "WNOWAIT",

        "WSTOPPED",

        "WSTOPSIG",

        "WTERMSIG",

        "WUNTRACED",

        "chown",

        "chroot",

        "confstr",

        "confstr\_names",

        "ctermid",

        "environb",

        "fchdir",

        "fchown",

        "fork",

        "forkpty",

        "fpathconf",

        "fstatvfs",

        "fwalk",

        "getegid",

        "getenvb",

        "geteuid",

        "getgid",

        "getgrouplist",

        "getgroups",

        "getloadavg",

        "getpgid",

        "getpgrp",

        "getpriority",

        "getsid",

        "getuid",

        "initgroups",

        "killpg",

        "lchown",

        "lockf",

        "major",

        "makedev",

        "minor",

        "mkfifo",

        "mknod",

        "nice",

        "openpty",

        "pathconf",

        "pathconf\_names",

        "posix\_spawn",

        "posix\_spawnp",

        "pread",

        "preadv",

        "pwrite",

        "pwritev",

        "readv",

        "register\_at\_fork",

        "sched\_get\_priority\_max",

        "sched\_get\_priority\_min",

        "sched\_yield",

        "sendfile",

        "setegid",

        "seteuid",

        "setgid",

        "setgroups",

        "setpgid",

        "setpgrp",

        "setpriority",

        "setregid",

        "setreuid",

        "setsid",

        "setuid",

        "spawnlp",

        "spawnlpe",

        "spawnvp",

        "spawnvpe",

        "statvfs",

        "sync",

        "sysconf",

        "sysconf\_names",

        "tcgetpgrp",

        "tcsetpgrp",

        "ttyname",

        "uname",

        "wait",

        "wait3",

        "wait4",

        "writev",

    ]

if sys.platform != "win32" and sys.version\_info >= (3, 13):

    \_\_all\_\_ += ["grantpt", "posix\_openpt", "ptsname", "unlockpt"]

if sys.platform != "win32" and sys.version\_info >= (3, 11):

    \_\_all\_\_ += ["login\_tty"]

if sys.platform != "win32" and sys.version\_info >= (3, 10):

    \_\_all\_\_ += ["O\_FSYNC"]

if sys.platform != "darwin" and sys.platform != "win32":

    \_\_all\_\_ += [

        "POSIX\_FADV\_DONTNEED",

        "POSIX\_FADV\_NOREUSE",

        "POSIX\_FADV\_NORMAL",

        "POSIX\_FADV\_RANDOM",

        "POSIX\_FADV\_SEQUENTIAL",

        "POSIX\_FADV\_WILLNEED",

        "RWF\_DSYNC",

        "RWF\_HIPRI",

        "RWF\_NOWAIT",

        "RWF\_SYNC",

        "ST\_APPEND",

        "ST\_MANDLOCK",

        "ST\_NOATIME",

        "ST\_NODEV",

        "ST\_NODIRATIME",

        "ST\_NOEXEC",

        "ST\_RELATIME",

        "ST\_SYNCHRONOUS",

        "ST\_WRITE",

        "fdatasync",

        "getresgid",

        "getresuid",

        "pipe2",

        "posix\_fadvise",

        "posix\_fallocate",

        "sched\_getaffinity",

        "sched\_getparam",

        "sched\_getscheduler",

        "sched\_param",

        "sched\_rr\_get\_interval",

        "sched\_setaffinity",

        "sched\_setparam",

        "sched\_setscheduler",

        "setresgid",

        "setresuid",

    ]

if sys.platform != "linux" and sys.platform != "win32":

    \_\_all\_\_ += ["O\_EXLOCK", "O\_SHLOCK", "chflags", "lchflags"]

if sys.platform != "linux" and sys.platform != "win32" and sys.version\_info >= (3, 13):

    \_\_all\_\_ += ["O\_EXEC", "O\_SEARCH"]

if sys.platform != "darwin" or sys.version\_info >= (3, 13):

    if sys.platform != "win32":

        \_\_all\_\_ += ["waitid", "waitid\_result"]

if sys.platform != "win32" or sys.version\_info >= (3, 13):

    \_\_all\_\_ += ["fchmod"]

    if sys.platform != "linux":

        \_\_all\_\_ += ["lchmod"]

if sys.platform != "win32" or sys.version\_info >= (3, 12):

    \_\_all\_\_ += ["get\_blocking", "set\_blocking"]

if sys.platform != "win32" or sys.version\_info >= (3, 11):

    \_\_all\_\_ += ["EX\_OK"]

# This unnecessary alias is to work around various errors

path = \_path

\_T = TypeVar("\_T")

\_T1 = TypeVar("\_T1")

\_T2 = TypeVar("\_T2")

# ----- os variables -----

error = OSError

supports\_bytes\_environ: bool

supports\_dir\_fd: set[Callable[..., Any]]

supports\_fd: set[Callable[..., Any]]

supports\_effective\_ids: set[Callable[..., Any]]

supports\_follow\_symlinks: set[Callable[..., Any]]

if sys.platform != "win32":

    # Unix only

    PRIO\_PROCESS: int

    PRIO\_PGRP: int

    PRIO\_USER: int

    F\_LOCK: int

    F\_TLOCK: int

    F\_ULOCK: int

    F\_TEST: int

    if sys.platform != "darwin":

        POSIX\_FADV\_NORMAL: int

        POSIX\_FADV\_SEQUENTIAL: int

        POSIX\_FADV\_RANDOM: int

        POSIX\_FADV\_NOREUSE: int

        POSIX\_FADV\_WILLNEED: int

        POSIX\_FADV\_DONTNEED: int

    if sys.platform != "linux" and sys.platform != "darwin":

        # In the os-module docs, these are marked as being available

        # on "Unix, not Emscripten, not WASI."

        # However, in the source code, a comment indicates they're "FreeBSD constants".

        # sys.platform could have one of many values on a FreeBSD Python build,

        # so the sys-module docs recommend doing `if sys.platform.startswith('freebsd')`

        # to detect FreeBSD builds. Unfortunately that would be too dynamic

        # for type checkers, however.

        SF\_NODISKIO: int

        SF\_MNOWAIT: int

        SF\_SYNC: int

        if sys.version\_info >= (3, 11):

            SF\_NOCACHE: int

    if sys.platform == "linux":

        XATTR\_SIZE\_MAX: int

        XATTR\_CREATE: int

        XATTR\_REPLACE: int

    P\_PID: int

    P\_PGID: int

    P\_ALL: int

    if sys.platform == "linux":

        P\_PIDFD: int

    WEXITED: int

    WSTOPPED: int

    WNOWAIT: int

    CLD\_EXITED: int

    CLD\_DUMPED: int

    CLD\_TRAPPED: int

    CLD\_CONTINUED: int

    CLD\_KILLED: int

    CLD\_STOPPED: int

    SCHED\_OTHER: int

    SCHED\_FIFO: int

    SCHED\_RR: int

    if sys.platform != "darwin" and sys.platform != "linux":

        SCHED\_SPORADIC: int

if sys.platform == "linux":

    SCHED\_BATCH: int

    SCHED\_IDLE: int

    SCHED\_RESET\_ON\_FORK: int

if sys.platform != "win32":

    RTLD\_LAZY: int

    RTLD\_NOW: int

    RTLD\_GLOBAL: int

    RTLD\_LOCAL: int

    RTLD\_NODELETE: int

    RTLD\_NOLOAD: int

if sys.platform == "linux":

    RTLD\_DEEPBIND: int

    GRND\_NONBLOCK: int

    GRND\_RANDOM: int

if sys.platform == "darwin" and sys.version\_info >= (3, 12):

    PRIO\_DARWIN\_BG: int

    PRIO\_DARWIN\_NONUI: int

    PRIO\_DARWIN\_PROCESS: int

    PRIO\_DARWIN\_THREAD: int

SEEK\_SET: int

SEEK\_CUR: int

SEEK\_END: int

if sys.platform != "win32":

    SEEK\_DATA: int

    SEEK\_HOLE: int

O\_RDONLY: int

O\_WRONLY: int

O\_RDWR: int

O\_APPEND: int

O\_CREAT: int

O\_EXCL: int

O\_TRUNC: int

if sys.platform == "win32":

    O\_BINARY: int

    O\_NOINHERIT: int

    O\_SHORT\_LIVED: int

    O\_TEMPORARY: int

    O\_RANDOM: int

    O\_SEQUENTIAL: int

    O\_TEXT: int

if sys.platform != "win32":

    O\_DSYNC: int

    O\_SYNC: int

    O\_NDELAY: int

    O\_NONBLOCK: int

    O\_NOCTTY: int

    O\_CLOEXEC: int

    O\_ASYNC: int  # Gnu extension if in C library

    O\_DIRECTORY: int  # Gnu extension if in C library

    O\_NOFOLLOW: int  # Gnu extension if in C library

    O\_ACCMODE: int  # TODO: when does this exist?

if sys.platform == "linux":

    O\_RSYNC: int

    O\_DIRECT: int  # Gnu extension if in C library

    O\_NOATIME: int  # Gnu extension if in C library

    O\_PATH: int  # Gnu extension if in C library

    O\_TMPFILE: int  # Gnu extension if in C library

    O\_LARGEFILE: int  # Gnu extension if in C library

if sys.platform != "linux" and sys.platform != "win32":

    O\_SHLOCK: int

    O\_EXLOCK: int

if sys.platform == "darwin" and sys.version\_info >= (3, 10):

    O\_EVTONLY: int

    O\_NOFOLLOW\_ANY: int

    O\_SYMLINK: int

if sys.platform != "win32" and sys.version\_info >= (3, 10):

    O\_FSYNC: int

if sys.platform != "linux" and sys.platform != "win32" and sys.version\_info >= (3, 13):

    O\_EXEC: int

    O\_SEARCH: int

if sys.platform != "win32" and sys.platform != "darwin":

    # posix, but apparently missing on macos

    ST\_APPEND: int

    ST\_MANDLOCK: int

    ST\_NOATIME: int

    ST\_NODEV: int

    ST\_NODIRATIME: int

    ST\_NOEXEC: int

    ST\_RELATIME: int

    ST\_SYNCHRONOUS: int

    ST\_WRITE: int

if sys.platform != "win32":

    NGROUPS\_MAX: int

    ST\_NOSUID: int

    ST\_RDONLY: int

curdir: str

pardir: str

sep: str

if sys.platform == "win32":

    altsep: str

else:

    altsep: str | None

extsep: str

pathsep: str

defpath: str

linesep: str

devnull: str

name: str

F\_OK: int

R\_OK: int

W\_OK: int

X\_OK: int

\_EnvironCodeFunc: TypeAlias = Callable[[AnyStr], AnyStr]

class \_Environ(MutableMapping[AnyStr, AnyStr], Generic[AnyStr]):

    encodekey: \_EnvironCodeFunc[AnyStr]

    decodekey: \_EnvironCodeFunc[AnyStr]

    encodevalue: \_EnvironCodeFunc[AnyStr]

    decodevalue: \_EnvironCodeFunc[AnyStr]

    def \_\_init\_\_(

        self,

        data: MutableMapping[AnyStr, AnyStr],

        encodekey: \_EnvironCodeFunc[AnyStr],

        decodekey: \_EnvironCodeFunc[AnyStr],

        encodevalue: \_EnvironCodeFunc[AnyStr],

        decodevalue: \_EnvironCodeFunc[AnyStr],

    ) -> None: ...

    def setdefault(self, key: AnyStr, value: AnyStr) -> AnyStr: ...

    def copy(self) -> dict[AnyStr, AnyStr]: ...

    def \_\_delitem\_\_(self, key: AnyStr) -> None: ...

    def \_\_getitem\_\_(self, key: AnyStr) -> AnyStr: ...

    def \_\_setitem\_\_(self, key: AnyStr, value: AnyStr) -> None: ...

    def \_\_iter\_\_(self) -> Iterator[AnyStr]: ...

    def \_\_len\_\_(self) -> int: ...

    def \_\_or\_\_(self, other: Mapping[\_T1, \_T2]) -> dict[AnyStr | \_T1, AnyStr | \_T2]: ...

    def \_\_ror\_\_(self, other: Mapping[\_T1, \_T2]) -> dict[AnyStr | \_T1, AnyStr | \_T2]: ...

    # We use @overload instead of a Union for reasons similar to those given for

    # overloading MutableMapping.update in stdlib/typing.pyi

    # The type: ignore is needed due to incompatible \_\_or\_\_/\_\_ior\_\_ signatures

    @overload  # type: ignore[misc]

    def \_\_ior\_\_(self, other: Mapping[AnyStr, AnyStr]) -> Self: ...

    @overload

    def \_\_ior\_\_(self, other: Iterable[tuple[AnyStr, AnyStr]]) -> Self: ...

environ: \_Environ[str]

if sys.platform != "win32":

    environb: \_Environ[bytes]

if sys.version\_info >= (3, 11) or sys.platform != "win32":

    EX\_OK: int

if sys.platform != "win32":

    confstr\_names: dict[str, int]

    pathconf\_names: dict[str, int]

    sysconf\_names: dict[str, int]

    EX\_USAGE: int

    EX\_DATAERR: int

    EX\_NOINPUT: int

    EX\_NOUSER: int

    EX\_NOHOST: int

    EX\_UNAVAILABLE: int

    EX\_SOFTWARE: int

    EX\_OSERR: int

    EX\_OSFILE: int

    EX\_CANTCREAT: int

    EX\_IOERR: int

    EX\_TEMPFAIL: int

    EX\_PROTOCOL: int

    EX\_NOPERM: int

    EX\_CONFIG: int

# Exists on some Unix platforms, e.g. Solaris.

if sys.platform != "win32" and sys.platform != "darwin" and sys.platform != "linux":

    EX\_NOTFOUND: int

P\_NOWAIT: int

P\_NOWAITO: int

P\_WAIT: int

if sys.platform == "win32":

    P\_DETACH: int

    P\_OVERLAY: int

# wait()/waitpid() options

if sys.platform != "win32":

    WNOHANG: int  # Unix only

    WCONTINUED: int  # some Unix systems

    WUNTRACED: int  # Unix only

TMP\_MAX: int  # Undocumented, but used by tempfile

# ----- os classes (structures) -----

@final

class stat\_result(structseq[float], tuple[int, int, int, int, int, int, int, float, float, float]):

    # The constructor of this class takes an iterable of variable length (though it must be at least 10).

    #

    # However, this class behaves like a tuple of 10 elements,

    # no matter how long the iterable supplied to the constructor is.

    # https://github.com/python/typeshed/pull/6560#discussion\_r767162532

    #

    # The 10 elements always present are st\_mode, st\_ino, st\_dev, st\_nlink,

    # st\_uid, st\_gid, st\_size, st\_atime, st\_mtime, st\_ctime.

    #

    # More items may be added at the end by some implementations.

    if sys.version\_info >= (3, 10):

        \_\_match\_args\_\_: Final = ("st\_mode", "st\_ino", "st\_dev", "st\_nlink", "st\_uid", "st\_gid", "st\_size")

    @property

    def st\_mode(self) -> int: ...  # protection bits,

    @property

    def st\_ino(self) -> int: ...  # inode number,

    @property

    def st\_dev(self) -> int: ...  # device,

    @property

    def st\_nlink(self) -> int: ...  # number of hard links,

    @property

    def st\_uid(self) -> int: ...  # user id of owner,

    @property

    def st\_gid(self) -> int: ...  # group id of owner,

    @property

    def st\_size(self) -> int: ...  # size of file, in bytes,

    @property

    def st\_atime(self) -> float: ...  # time of most recent access,

    @property

    def st\_mtime(self) -> float: ...  # time of most recent content modification,

    # platform dependent (time of most recent metadata change on Unix, or the time of creation on Windows)

    if sys.version\_info >= (3, 12) and sys.platform == "win32":

        @property

        @deprecated(

            """\

Use st\_birthtime instead to retrieve the file creation time. \

In the future, this property will contain the last metadata change time."""

        )

        def st\_ctime(self) -> float: ...

    else:

        @property

        def st\_ctime(self) -> float: ...

    @property

    def st\_atime\_ns(self) -> int: ...  # time of most recent access, in nanoseconds

    @property

    def st\_mtime\_ns(self) -> int: ...  # time of most recent content modification in nanoseconds

    # platform dependent (time of most recent metadata change on Unix, or the time of creation on Windows) in nanoseconds

    @property

    def st\_ctime\_ns(self) -> int: ...

    if sys.platform == "win32":

        @property

        def st\_file\_attributes(self) -> int: ...

        @property

        def st\_reparse\_tag(self) -> int: ...

        if sys.version\_info >= (3, 12):

            @property

            def st\_birthtime(self) -> float: ...  # time of file creation in seconds

            @property

            def st\_birthtime\_ns(self) -> int: ...  # time of file creation in nanoseconds

    else:

        @property

        def st\_blocks(self) -> int: ...  # number of blocks allocated for file

        @property

        def st\_blksize(self) -> int: ...  # filesystem blocksize

        @property

        def st\_rdev(self) -> int: ...  # type of device if an inode device

        if sys.platform != "linux":

            # These properties are available on MacOS, but not Ubuntu.

            # On other Unix systems (such as FreeBSD), the following attributes may be

            # available (but may be only filled out if root tries to use them):

            @property

            def st\_gen(self) -> int: ...  # file generation number

            @property

            def st\_birthtime(self) -> float: ...  # time of file creation in seconds

    if sys.platform == "darwin":

        @property

        def st\_flags(self) -> int: ...  # user defined flags for file

    # Attributes documented as sometimes appearing, but deliberately omitted from the stub: `st\_creator`, `st\_rsize`, `st\_type`.

    # See https://github.com/python/typeshed/pull/6560#issuecomment-991253327

# mypy and pyright object to this being both ABC and Protocol.

# At runtime it inherits from ABC and is not a Protocol, but it will be

# on the allowlist for use as a Protocol starting in 3.14.

@runtime\_checkable

class PathLike(ABC, Protocol[AnyStr\_co]):  # type: ignore[misc]  # pyright: ignore[reportGeneralTypeIssues]

    @abstractmethod

    def \_\_fspath\_\_(self) -> AnyStr\_co: ...

@overload

def listdir(path: StrPath | None = None) -> list[str]: ...

@overload

def listdir(path: BytesPath) -> list[bytes]: ...

@overload

def listdir(path: int) -> list[str]: ...

@final

class DirEntry(Generic[AnyStr]):

    # This is what the scandir iterator yields

    # The constructor is hidden

    @property

    def name(self) -> AnyStr: ...

    @property

    def path(self) -> AnyStr: ...

    def inode(self) -> int: ...

    def is\_dir(self, \*, follow\_symlinks: bool = True) -> bool: ...

    def is\_file(self, \*, follow\_symlinks: bool = True) -> bool: ...

    def is\_symlink(self) -> bool: ...

    def stat(self, \*, follow\_symlinks: bool = True) -> stat\_result: ...

    def \_\_fspath\_\_(self) -> AnyStr: ...

    def \_\_class\_getitem\_\_(cls, item: Any, /) -> GenericAlias: ...

    if sys.version\_info >= (3, 12):

        def is\_junction(self) -> bool: ...

@final

class statvfs\_result(structseq[int], tuple[int, int, int, int, int, int, int, int, int, int, int]):

    if sys.version\_info >= (3, 10):

        \_\_match\_args\_\_: Final = (

            "f\_bsize",

            "f\_frsize",

            "f\_blocks",

            "f\_bfree",

            "f\_bavail",

            "f\_files",

            "f\_ffree",

            "f\_favail",

            "f\_flag",

            "f\_namemax",

        )

    @property

    def f\_bsize(self) -> int: ...

    @property

    def f\_frsize(self) -> int: ...

    @property

    def f\_blocks(self) -> int: ...

    @property

    def f\_bfree(self) -> int: ...

    @property

    def f\_bavail(self) -> int: ...

    @property

    def f\_files(self) -> int: ...

    @property

    def f\_ffree(self) -> int: ...

    @property

    def f\_favail(self) -> int: ...

    @property

    def f\_flag(self) -> int: ...

    @property

    def f\_namemax(self) -> int: ...

    @property

    def f\_fsid(self) -> int: ...

# ----- os function stubs -----

def fsencode(filename: StrOrBytesPath) -> bytes: ...

def fsdecode(filename: StrOrBytesPath) -> str: ...

@overload

def fspath(path: str) -> str: ...

@overload

def fspath(path: bytes) -> bytes: ...

@overload

def fspath(path: PathLike[AnyStr]) -> AnyStr: ...

def get\_exec\_path(env: Mapping[str, str] | None = None) -> list[str]: ...

def getlogin() -> str: ...

def getpid() -> int: ...

def getppid() -> int: ...

def strerror(code: int, /) -> str: ...

def umask(mask: int, /) -> int: ...

@final

class uname\_result(structseq[str], tuple[str, str, str, str, str]):

    if sys.version\_info >= (3, 10):

        \_\_match\_args\_\_: Final = ("sysname", "nodename", "release", "version", "machine")

    @property

    def sysname(self) -> str: ...

    @property

    def nodename(self) -> str: ...

    @property

    def release(self) -> str: ...

    @property

    def version(self) -> str: ...

    @property

    def machine(self) -> str: ...

if sys.platform != "win32":

    def ctermid() -> str: ...

    def getegid() -> int: ...

    def geteuid() -> int: ...

    def getgid() -> int: ...

    def getgrouplist(user: str, group: int, /) -> list[int]: ...

    def getgroups() -> list[int]: ...  # Unix only, behaves differently on Mac

    def initgroups(username: str, gid: int, /) -> None: ...

    def getpgid(pid: int) -> int: ...

    def getpgrp() -> int: ...

    def getpriority(which: int, who: int) -> int: ...

    def setpriority(which: int, who: int, priority: int) -> None: ...

    if sys.platform != "darwin":

        def getresuid() -> tuple[int, int, int]: ...

        def getresgid() -> tuple[int, int, int]: ...

    def getuid() -> int: ...

    def setegid(egid: int, /) -> None: ...

    def seteuid(euid: int, /) -> None: ...

    def setgid(gid: int, /) -> None: ...

    def setgroups(groups: Sequence[int], /) -> None: ...

    def setpgrp() -> None: ...

    def setpgid(pid: int, pgrp: int, /) -> None: ...

    def setregid(rgid: int, egid: int, /) -> None: ...

    if sys.platform != "darwin":

        def setresgid(rgid: int, egid: int, sgid: int, /) -> None: ...

        def setresuid(ruid: int, euid: int, suid: int, /) -> None: ...

    def setreuid(ruid: int, euid: int, /) -> None: ...

    def getsid(pid: int, /) -> int: ...

    def setsid() -> None: ...

    def setuid(uid: int, /) -> None: ...

    def uname() -> uname\_result: ...

@overload

def getenv(key: str) -> str | None: ...

@overload

def getenv(key: str, default: \_T) -> str | \_T: ...

if sys.platform != "win32":

    @overload

    def getenvb(key: bytes) -> bytes | None: ...

    @overload

    def getenvb(key: bytes, default: \_T) -> bytes | \_T: ...

    def putenv(name: StrOrBytesPath, value: StrOrBytesPath, /) -> None: ...

    def unsetenv(name: StrOrBytesPath, /) -> None: ...

else:

    def putenv(name: str, value: str, /) -> None: ...

    def unsetenv(name: str, /) -> None: ...

\_Opener: TypeAlias = Callable[[str, int], int]

@overload

def fdopen(

    fd: int,

    mode: OpenTextMode = "r",

    buffering: int = -1,

    encoding: str | None = None,

    errors: str | None = ...,

    newline: str | None = ...,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> TextIOWrapper: ...

@overload

def fdopen(

    fd: int,

    mode: OpenBinaryMode,

    buffering: Literal[0],

    encoding: None = None,

    errors: None = None,

    newline: None = None,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> FileIO: ...

@overload

def fdopen(

    fd: int,

    mode: OpenBinaryModeUpdating,

    buffering: Literal[-1, 1] = -1,

    encoding: None = None,

    errors: None = None,

    newline: None = None,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> BufferedRandom: ...

@overload

def fdopen(

    fd: int,

    mode: OpenBinaryModeWriting,

    buffering: Literal[-1, 1] = -1,

    encoding: None = None,

    errors: None = None,

    newline: None = None,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> BufferedWriter: ...

@overload

def fdopen(

    fd: int,

    mode: OpenBinaryModeReading,

    buffering: Literal[-1, 1] = -1,

    encoding: None = None,

    errors: None = None,

    newline: None = None,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> BufferedReader: ...

@overload

def fdopen(

    fd: int,

    mode: OpenBinaryMode,

    buffering: int = -1,

    encoding: None = None,

    errors: None = None,

    newline: None = None,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> BinaryIO: ...

@overload

def fdopen(

    fd: int,

    mode: str,

    buffering: int = -1,

    encoding: str | None = None,

    errors: str | None = ...,

    newline: str | None = ...,

    closefd: bool = ...,

    opener: \_Opener | None = ...,

) -> IO[Any]: ...

def close(fd: int) -> None: ...

def closerange(fd\_low: int, fd\_high: int, /) -> None: ...

def device\_encoding(fd: int) -> str | None: ...

def dup(fd: int, /) -> int: ...

def dup2(fd: int, fd2: int, inheritable: bool = True) -> int: ...

def fstat(fd: int) -> stat\_result: ...

def ftruncate(fd: int, length: int, /) -> None: ...

def fsync(fd: FileDescriptorLike) -> None: ...

def isatty(fd: int, /) -> bool: ...

if sys.platform != "win32" and sys.version\_info >= (3, 11):

    def login\_tty(fd: int, /) -> None: ...

if sys.version\_info >= (3, 11):

    def lseek(fd: int, position: int, whence: int, /) -> int: ...

else:

    def lseek(fd: int, position: int, how: int, /) -> int: ...

def open(path: StrOrBytesPath, flags: int, mode: int = 0o777, \*, dir\_fd: int | None = None) -> int: ...

def pipe() -> tuple[int, int]: ...

def read(fd: int, length: int, /) -> bytes: ...

if sys.version\_info >= (3, 12) or sys.platform != "win32":

    def get\_blocking(fd: int, /) -> bool: ...

    def set\_blocking(fd: int, blocking: bool, /) -> None: ...

if sys.platform != "win32":

    def fchown(fd: int, uid: int, gid: int) -> None: ...

    def fpathconf(fd: int, name: str | int, /) -> int: ...

    def fstatvfs(fd: int, /) -> statvfs\_result: ...

    def lockf(fd: int, command: int, length: int, /) -> None: ...

    def openpty() -> tuple[int, int]: ...  # some flavors of Unix

    if sys.platform != "darwin":

        def fdatasync(fd: FileDescriptorLike) -> None: ...

        def pipe2(flags: int, /) -> tuple[int, int]: ...  # some flavors of Unix

        def posix\_fallocate(fd: int, offset: int, length: int, /) -> None: ...

        def posix\_fadvise(fd: int, offset: int, length: int, advice: int, /) -> None: ...

    def pread(fd: int, length: int, offset: int, /) -> bytes: ...

    def pwrite(fd: int, buffer: ReadableBuffer, offset: int, /) -> int: ...

    # In CI, stubtest sometimes reports that these are available on MacOS, sometimes not

    def preadv(fd: int, buffers: SupportsLenAndGetItem[WriteableBuffer], offset: int, flags: int = 0, /) -> int: ...

    def pwritev(fd: int, buffers: SupportsLenAndGetItem[ReadableBuffer], offset: int, flags: int = 0, /) -> int: ...

    if sys.platform != "darwin":

        if sys.version\_info >= (3, 10):

            RWF\_APPEND: int  # docs say available on 3.7+, stubtest says otherwise

        RWF\_DSYNC: int

        RWF\_SYNC: int

        RWF\_HIPRI: int

        RWF\_NOWAIT: int

    if sys.platform == "linux":

        def sendfile(out\_fd: FileDescriptor, in\_fd: FileDescriptor, offset: int | None, count: int) -> int: ...

    else:

        def sendfile(

            out\_fd: FileDescriptor,

            in\_fd: FileDescriptor,

            offset: int,

            count: int,

            headers: Sequence[ReadableBuffer] = ...,

            trailers: Sequence[ReadableBuffer] = ...,

            flags: int = 0,

        ) -> int: ...  # FreeBSD and Mac OS X only

    def readv(fd: int, buffers: SupportsLenAndGetItem[WriteableBuffer], /) -> int: ...

    def writev(fd: int, buffers: SupportsLenAndGetItem[ReadableBuffer], /) -> int: ...

@final

class terminal\_size(structseq[int], tuple[int, int]):

    if sys.version\_info >= (3, 10):

        \_\_match\_args\_\_: Final = ("columns", "lines")

    @property

    def columns(self) -> int: ...

    @property

    def lines(self) -> int: ...

def get\_terminal\_size(fd: int = ..., /) -> terminal\_size: ...

def get\_inheritable(fd: int, /) -> bool: ...

def set\_inheritable(fd: int, inheritable: bool, /) -> None: ...

if sys.platform == "win32":

    def get\_handle\_inheritable(handle: int, /) -> bool: ...

    def set\_handle\_inheritable(handle: int, inheritable: bool, /) -> None: ...

if sys.platform != "win32":

    # Unix only

    def tcgetpgrp(fd: int, /) -> int: ...

    def tcsetpgrp(fd: int, pgid: int, /) -> None: ...

    def ttyname(fd: int, /) -> str: ...

def write(fd: int, data: ReadableBuffer, /) -> int: ...

def access(

    path: FileDescriptorOrPath, mode: int, \*, dir\_fd: int | None = None, effective\_ids: bool = False, follow\_symlinks: bool = True

) -> bool: ...

def chdir(path: FileDescriptorOrPath) -> None: ...

if sys.platform != "win32":

    def fchdir(fd: FileDescriptorLike) -> None: ...

def getcwd() -> str: ...

def getcwdb() -> bytes: ...

def chmod(path: FileDescriptorOrPath, mode: int, \*, dir\_fd: int | None = None, follow\_symlinks: bool = ...) -> None: ...

if sys.platform != "win32" and sys.platform != "linux":

    def chflags(path: StrOrBytesPath, flags: int, follow\_symlinks: bool = True) -> None: ...  # some flavors of Unix

    def lchflags(path: StrOrBytesPath, flags: int) -> None: ...

if sys.platform != "win32":

    def chroot(path: StrOrBytesPath) -> None: ...

    def chown(

        path: FileDescriptorOrPath, uid: int, gid: int, \*, dir\_fd: int | None = None, follow\_symlinks: bool = True

    ) -> None: ...

    def lchown(path: StrOrBytesPath, uid: int, gid: int) -> None: ...

def link(

    src: StrOrBytesPath,

    dst: StrOrBytesPath,

    \*,

    src\_dir\_fd: int | None = None,

    dst\_dir\_fd: int | None = None,

    follow\_symlinks: bool = True,

) -> None: ...

def lstat(path: StrOrBytesPath, \*, dir\_fd: int | None = None) -> stat\_result: ...

def mkdir(path: StrOrBytesPath, mode: int = 0o777, \*, dir\_fd: int | None = None) -> None: ...

if sys.platform != "win32":

    def mkfifo(path: StrOrBytesPath, mode: int = 0o666, \*, dir\_fd: int | None = None) -> None: ...  # Unix only

def makedirs(name: StrOrBytesPath, mode: int = 0o777, exist\_ok: bool = False) -> None: ...

if sys.platform != "win32":

    def mknod(path: StrOrBytesPath, mode: int = 0o600, device: int = 0, \*, dir\_fd: int | None = None) -> None: ...

    def major(device: int, /) -> int: ...

    def minor(device: int, /) -> int: ...

    def makedev(major: int, minor: int, /) -> int: ...

    def pathconf(path: FileDescriptorOrPath, name: str | int) -> int: ...  # Unix only

def readlink(path: GenericPath[AnyStr], \*, dir\_fd: int | None = None) -> AnyStr: ...

def remove(path: StrOrBytesPath, \*, dir\_fd: int | None = None) -> None: ...

def removedirs(name: StrOrBytesPath) -> None: ...

def rename(src: StrOrBytesPath, dst: StrOrBytesPath, \*, src\_dir\_fd: int | None = None, dst\_dir\_fd: int | None = None) -> None: ...

def renames(old: StrOrBytesPath, new: StrOrBytesPath) -> None: ...

def replace(

    src: StrOrBytesPath, dst: StrOrBytesPath, \*, src\_dir\_fd: int | None = None, dst\_dir\_fd: int | None = None

) -> None: ...

def rmdir(path: StrOrBytesPath, \*, dir\_fd: int | None = None) -> None: ...

@final

class \_ScandirIterator(Generic[AnyStr]):

    def \_\_del\_\_(self) -> None: ...

    def \_\_iter\_\_(self) -> Self: ...

    def \_\_next\_\_(self) -> DirEntry[AnyStr]: ...

    def \_\_enter\_\_(self) -> Self: ...

    def \_\_exit\_\_(self, \*args: Unused) -> None: ...

    def close(self) -> None: ...

@overload

def scandir(path: None = None) -> \_ScandirIterator[str]: ...

@overload

def scandir(path: int) -> \_ScandirIterator[str]: ...

@overload

def scandir(path: GenericPath[AnyStr]) -> \_ScandirIterator[AnyStr]: ...

def stat(path: FileDescriptorOrPath, \*, dir\_fd: int | None = None, follow\_symlinks: bool = True) -> stat\_result: ...

if sys.platform != "win32":

    def statvfs(path: FileDescriptorOrPath) -> statvfs\_result: ...  # Unix only

def symlink(

    src: StrOrBytesPath, dst: StrOrBytesPath, target\_is\_directory: bool = False, \*, dir\_fd: int | None = None

) -> None: ...

if sys.platform != "win32":

    def sync() -> None: ...  # Unix only

def truncate(path: FileDescriptorOrPath, length: int) -> None: ...  # Unix only up to version 3.4

def unlink(path: StrOrBytesPath, \*, dir\_fd: int | None = None) -> None: ...

def utime(

    path: FileDescriptorOrPath,

    times: tuple[int, int] | tuple[float, float] | None = None,

    \*,

    ns: tuple[int, int] = ...,

    dir\_fd: int | None = None,

    follow\_symlinks: bool = True,

) -> None: ...

\_OnError: TypeAlias = Callable[[OSError], object]

def walk(

    top: GenericPath[AnyStr], topdown: bool = True, onerror: \_OnError | None = None, followlinks: bool = False

) -> Iterator[tuple[AnyStr, list[AnyStr], list[AnyStr]]]: ...

if sys.platform != "win32":

    @overload

    def fwalk(

        top: StrPath = ".",

        topdown: bool = True,

        onerror: \_OnError | None = None,

        \*,

        follow\_symlinks: bool = False,

        dir\_fd: int | None = None,

    ) -> Iterator[tuple[str, list[str], list[str], int]]: ...

    @overload

    def fwalk(

        top: BytesPath,

        topdown: bool = True,

        onerror: \_OnError | None = None,

        \*,

        follow\_symlinks: bool = False,

        dir\_fd: int | None = None,

    ) -> Iterator[tuple[bytes, list[bytes], list[bytes], int]]: ...

    if sys.platform == "linux":

        def getxattr(path: FileDescriptorOrPath, attribute: StrOrBytesPath, \*, follow\_symlinks: bool = True) -> bytes: ...

        def listxattr(path: FileDescriptorOrPath | None = None, \*, follow\_symlinks: bool = True) -> list[str]: ...

        def removexattr(path: FileDescriptorOrPath, attribute: StrOrBytesPath, \*, follow\_symlinks: bool = True) -> None: ...

        def setxattr(

            path: FileDescriptorOrPath,

            attribute: StrOrBytesPath,

            value: ReadableBuffer,

            flags: int = 0,

            \*,

            follow\_symlinks: bool = True,

        ) -> None: ...

def abort() -> NoReturn: ...

# These are defined as execl(file, \*args) but the first \*arg is mandatory.

def execl(file: StrOrBytesPath, \*args: Unpack[tuple[StrOrBytesPath, Unpack[tuple[StrOrBytesPath, ...]]]]) -> NoReturn: ...

def execlp(file: StrOrBytesPath, \*args: Unpack[tuple[StrOrBytesPath, Unpack[tuple[StrOrBytesPath, ...]]]]) -> NoReturn: ...

# These are: execle(file, \*args, env) but env is pulled from the last element of the args.

def execle(

    file: StrOrBytesPath, \*args: Unpack[tuple[StrOrBytesPath, Unpack[tuple[StrOrBytesPath, ...]], \_ExecEnv]]

) -> NoReturn: ...

def execlpe(

    file: StrOrBytesPath, \*args: Unpack[tuple[StrOrBytesPath, Unpack[tuple[StrOrBytesPath, ...]], \_ExecEnv]]

) -> NoReturn: ...

# The docs say `args: tuple or list of strings`

# The implementation enforces tuple or list so we can't use Sequence.

# Not separating out PathLike[str] and PathLike[bytes] here because it doesn't make much difference

# in practice, and doing so would explode the number of combinations in this already long union.

# All these combinations are necessary due to list being invariant.

\_ExecVArgs: TypeAlias = (

    tuple[StrOrBytesPath, ...]

    | list[bytes]

    | list[str]

    | list[PathLike[Any]]

    | list[bytes | str]

    | list[bytes | PathLike[Any]]

    | list[str | PathLike[Any]]

    | list[bytes | str | PathLike[Any]]

)

# Depending on the OS, the keys and values are passed either to

# PyUnicode\_FSDecoder (which accepts str | ReadableBuffer) or to

# PyUnicode\_FSConverter (which accepts StrOrBytesPath). For simplicity,

# we limit to str | bytes.

\_ExecEnv: TypeAlias = Mapping[bytes, bytes | str] | Mapping[str, bytes | str]

def execv(path: StrOrBytesPath, argv: \_ExecVArgs, /) -> NoReturn: ...

def execve(path: FileDescriptorOrPath, argv: \_ExecVArgs, env: \_ExecEnv) -> NoReturn: ...

def execvp(file: StrOrBytesPath, args: \_ExecVArgs) -> NoReturn: ...

def execvpe(file: StrOrBytesPath, args: \_ExecVArgs, env: \_ExecEnv) -> NoReturn: ...

def \_exit(status: int) -> NoReturn: ...

def kill(pid: int, signal: int, /) -> None: ...

if sys.platform != "win32":

    # Unix only

    def fork() -> int: ...

    def forkpty() -> tuple[int, int]: ...  # some flavors of Unix

    def killpg(pgid: int, signal: int, /) -> None: ...

    def nice(increment: int, /) -> int: ...

    if sys.platform != "darwin" and sys.platform != "linux":

        def plock(op: int, /) -> None: ...

class \_wrap\_close:

    def \_\_init\_\_(self, stream: TextIOWrapper, proc: Popen[str]) -> None: ...

    def close(self) -> int | None: ...

    def \_\_enter\_\_(self) -> Self: ...

    def \_\_exit\_\_(

        self, exc\_type: type[BaseException] | None, exc\_val: BaseException | None, exc\_tb: TracebackType | None

    ) -> None: ...

    def \_\_iter\_\_(self) -> Iterator[str]: ...

    # Methods below here don't exist directly on the \_wrap\_close object, but

    # are copied from the wrapped TextIOWrapper object via \_\_getattr\_\_.

    # but undocumented. Only a subset are currently included here.

    def read(self, size: int | None = -1, /) -> str: ...

    def readable(self) -> bool: ...

    def readline(self, size: int = -1, /) -> str: ...

    def readlines(self, hint: int = -1, /) -> list[str]: ...

    def writable(self) -> bool: ...

    def write(self, s: str, /) -> int: ...

    def writelines(self, lines: Iterable[str], /) -> None: ...

def popen(cmd: str, mode: str = "r", buffering: int = -1) -> \_wrap\_close: ...

def spawnl(mode: int, file: StrOrBytesPath, arg0: StrOrBytesPath, \*args: StrOrBytesPath) -> int: ...

def spawnle(mode: int, file: StrOrBytesPath, arg0: StrOrBytesPath, \*args: Any) -> int: ...  # Imprecise sig

if sys.platform != "win32":

    def spawnv(mode: int, file: StrOrBytesPath, args: \_ExecVArgs) -> int: ...

    def spawnve(mode: int, file: StrOrBytesPath, args: \_ExecVArgs, env: \_ExecEnv) -> int: ...

else:

    def spawnv(mode: int, path: StrOrBytesPath, argv: \_ExecVArgs, /) -> int: ...

    def spawnve(mode: int, path: StrOrBytesPath, argv: \_ExecVArgs, env: \_ExecEnv, /) -> int: ...

def system(command: StrOrBytesPath) -> int: ...

@final

class times\_result(structseq[float], tuple[float, float, float, float, float]):

    if sys.version\_info >= (3, 10):

        \_\_match\_args\_\_: Final = ("user", "system", "children\_user", "children\_system", "elapsed")

    @property

    def user(self) -> float: ...

    @property

    def system(self) -> float: ...

    @property

    def children\_user(self) -> float: ...

    @property

    def children\_system(self) -> float: ...

    @property

    def elapsed(self) -> float: ...

def times() -> times\_result: ...

def waitpid(pid: int, options: int, /) -> tuple[int, int]: ...

if sys.platform == "win32":

    if sys.version\_info >= (3, 10):

        def startfile(

            filepath: StrOrBytesPath,

            operation: str = ...,

            arguments: str = "",

            cwd: StrOrBytesPath | None = None,

            show\_cmd: int = 1,

        ) -> None: ...

    else:

        def startfile(filepath: StrOrBytesPath, operation: str = ...) -> None: ...

else:

    def spawnlp(mode: int, file: StrOrBytesPath, arg0: StrOrBytesPath, \*args: StrOrBytesPath) -> int: ...

    def spawnlpe(mode: int, file: StrOrBytesPath, arg0: StrOrBytesPath, \*args: Any) -> int: ...  # Imprecise signature

    def spawnvp(mode: int, file: StrOrBytesPath, args: \_ExecVArgs) -> int: ...

    def spawnvpe(mode: int, file: StrOrBytesPath, args: \_ExecVArgs, env: \_ExecEnv) -> int: ...

    def wait() -> tuple[int, int]: ...  # Unix only

    # Added to MacOS in 3.13

    if sys.platform != "darwin" or sys.version\_info >= (3, 13):

        @final

        class waitid\_result(structseq[int], tuple[int, int, int, int, int]):

            if sys.version\_info >= (3, 10):

                \_\_match\_args\_\_: Final = ("si\_pid", "si\_uid", "si\_signo", "si\_status", "si\_code")

            @property

            def si\_pid(self) -> int: ...

            @property

            def si\_uid(self) -> int: ...

            @property

            def si\_signo(self) -> int: ...

            @property

            def si\_status(self) -> int: ...

            @property

            def si\_code(self) -> int: ...

        def waitid(idtype: int, ident: int, options: int, /) -> waitid\_result | None: ...

    from resource import struct\_rusage

    def wait3(options: int) -> tuple[int, int, struct\_rusage]: ...

    def wait4(pid: int, options: int) -> tuple[int, int, struct\_rusage]: ...

    def WCOREDUMP(status: int, /) -> bool: ...

    def WIFCONTINUED(status: int) -> bool: ...

    def WIFSTOPPED(status: int) -> bool: ...

    def WIFSIGNALED(status: int) -> bool: ...

    def WIFEXITED(status: int) -> bool: ...

    def WEXITSTATUS(status: int) -> int: ...

    def WSTOPSIG(status: int) -> int: ...

    def WTERMSIG(status: int) -> int: ...

    def posix\_spawn(

        path: StrOrBytesPath,

        argv: \_ExecVArgs,

        env: \_ExecEnv,

        /,

        \*,

        file\_actions: Sequence[tuple[Any, ...]] | None = ...,

        setpgroup: int | None = ...,

        resetids: bool = ...,

        setsid: bool = ...,

        setsigmask: Iterable[int] = ...,

        setsigdef: Iterable[int] = ...,

        scheduler: tuple[Any, sched\_param] | None = ...,

    ) -> int: ...

    def posix\_spawnp(

        path: StrOrBytesPath,

        argv: \_ExecVArgs,

        env: \_ExecEnv,

        /,

        \*,

        file\_actions: Sequence[tuple[Any, ...]] | None = ...,

        setpgroup: int | None = ...,

        resetids: bool = ...,

        setsid: bool = ...,

        setsigmask: Iterable[int] = ...,

        setsigdef: Iterable[int] = ...,

        scheduler: tuple[Any, sched\_param] | None = ...,

    ) -> int: ...

    POSIX\_SPAWN\_OPEN: int

    POSIX\_SPAWN\_CLOSE: int

    POSIX\_SPAWN\_DUP2: int

if sys.platform != "win32":

    @final

    class sched\_param(structseq[int], tuple[int]):

        if sys.version\_info >= (3, 10):

            \_\_match\_args\_\_: Final = ("sched\_priority",)

        def \_\_new\_\_(cls, sched\_priority: int) -> Self: ...

        @property

        def sched\_priority(self) -> int: ...

    def sched\_get\_priority\_min(policy: int) -> int: ...  # some flavors of Unix

    def sched\_get\_priority\_max(policy: int) -> int: ...  # some flavors of Unix

    def sched\_yield() -> None: ...  # some flavors of Unix

    if sys.platform != "darwin":

        def sched\_setscheduler(pid: int, policy: int, param: sched\_param, /) -> None: ...  # some flavors of Unix

        def sched\_getscheduler(pid: int, /) -> int: ...  # some flavors of Unix

        def sched\_rr\_get\_interval(pid: int, /) -> float: ...  # some flavors of Unix

        def sched\_setparam(pid: int, param: sched\_param, /) -> None: ...  # some flavors of Unix

        def sched\_getparam(pid: int, /) -> sched\_param: ...  # some flavors of Unix

        def sched\_setaffinity(pid: int, mask: Iterable[int], /) -> None: ...  # some flavors of Unix

        def sched\_getaffinity(pid: int, /) -> set[int]: ...  # some flavors of Unix

def cpu\_count() -> int | None: ...

if sys.version\_info >= (3, 13):

    # Documented to return `int | None`, but falls back to `len(sched\_getaffinity(0))` when

    # available. See https://github.com/python/cpython/blob/417c130/Lib/os.py#L1175-L1186.

    if sys.platform != "win32" and sys.platform != "darwin":

        def process\_cpu\_count() -> int: ...

    else:

        def process\_cpu\_count() -> int | None: ...

if sys.platform != "win32":

    # Unix only

    def confstr(name: str | int, /) -> str | None: ...

    def getloadavg() -> tuple[float, float, float]: ...

    def sysconf(name: str | int, /) -> int: ...

if sys.platform == "linux":

    def getrandom(size: int, flags: int = 0) -> bytes: ...

def urandom(size: int, /) -> bytes: ...

if sys.platform != "win32":

    def register\_at\_fork(

        \*,

        before: Callable[..., Any] | None = ...,

        after\_in\_parent: Callable[..., Any] | None = ...,

        after\_in\_child: Callable[..., Any] | None = ...,

    ) -> None: ...

if sys.platform == "win32":

    class \_AddedDllDirectory:

        path: str | None

        def \_\_init\_\_(self, path: str | None, cookie: \_T, remove\_dll\_directory: Callable[[\_T], object]) -> None: ...

        def close(self) -> None: ...

        def \_\_enter\_\_(self) -> Self: ...

        def \_\_exit\_\_(self, \*args: Unused) -> None: ...

    def add\_dll\_directory(path: str) -> \_AddedDllDirectory: ...

if sys.platform == "linux":

    MFD\_CLOEXEC: int

    MFD\_ALLOW\_SEALING: int

    MFD\_HUGETLB: int

    MFD\_HUGE\_SHIFT: int

    MFD\_HUGE\_MASK: int

    MFD\_HUGE\_64KB: int

    MFD\_HUGE\_512KB: int

    MFD\_HUGE\_1MB: int

    MFD\_HUGE\_2MB: int

    MFD\_HUGE\_8MB: int

    MFD\_HUGE\_16MB: int

    MFD\_HUGE\_32MB: int

    MFD\_HUGE\_256MB: int

    MFD\_HUGE\_512MB: int

    MFD\_HUGE\_1GB: int

    MFD\_HUGE\_2GB: int

    MFD\_HUGE\_16GB: int

    def memfd\_create(name: str, flags: int = ...) -> int: ...

    def copy\_file\_range(src: int, dst: int, count: int, offset\_src: int | None = ..., offset\_dst: int | None = ...) -> int: ...

def waitstatus\_to\_exitcode(status: int) -> int: ...

if sys.platform == "linux":

    def pidfd\_open(pid: int, flags: int = ...) -> int: ...

if sys.version\_info >= (3, 12) and sys.platform == "linux":

    PIDFD\_NONBLOCK: Final = 2048

if sys.version\_info >= (3, 12) and sys.platform == "win32":

    def listdrives() -> list[str]: ...

    def listmounts(volume: str) -> list[str]: ...

    def listvolumes() -> list[str]: ...

if sys.version\_info >= (3, 10) and sys.platform == "linux":

    EFD\_CLOEXEC: int

    EFD\_NONBLOCK: int

    EFD\_SEMAPHORE: int

    SPLICE\_F\_MORE: int

    SPLICE\_F\_MOVE: int

    SPLICE\_F\_NONBLOCK: int

    def eventfd(initval: int, flags: int = 524288) -> FileDescriptor: ...

    def eventfd\_read(fd: FileDescriptor) -> int: ...

    def eventfd\_write(fd: FileDescriptor, value: int) -> None: ...

    def splice(

        src: FileDescriptor,

        dst: FileDescriptor,

        count: int,

        offset\_src: int | None = ...,

        offset\_dst: int | None = ...,

        flags: int = 0,

    ) -> int: ...

if sys.version\_info >= (3, 12) and sys.platform == "linux":

    CLONE\_FILES: int

    CLONE\_FS: int

    CLONE\_NEWCGROUP: int  # Linux 4.6+

    CLONE\_NEWIPC: int  # Linux 2.6.19+

    CLONE\_NEWNET: int  # Linux 2.6.24+

    CLONE\_NEWNS: int

    CLONE\_NEWPID: int  # Linux 3.8+

    CLONE\_NEWTIME: int  # Linux 5.6+

    CLONE\_NEWUSER: int  # Linux 3.8+

    CLONE\_NEWUTS: int  # Linux 2.6.19+

    CLONE\_SIGHAND: int

    CLONE\_SYSVSEM: int  # Linux 2.6.26+

    CLONE\_THREAD: int

    CLONE\_VM: int

    def unshare(flags: int) -> None: ...

    def setns(fd: FileDescriptorLike, nstype: int = 0) -> None: ...

if sys.version\_info >= (3, 13) and sys.platform != "win32":

    def posix\_openpt(oflag: int, /) -> int: ...

    def grantpt(fd: FileDescriptorLike, /) -> None: ...

    def unlockpt(fd: FileDescriptorLike, /) -> None: ...

    def ptsname(fd: FileDescriptorLike, /) -> str: ...

if sys.version\_info >= (3, 13) and sys.platform == "linux":

    TFD\_TIMER\_ABSTIME: Final = 1

    TFD\_TIMER\_CANCEL\_ON\_SET: Final = 2

    TFD\_NONBLOCK: Final[int]

    TFD\_CLOEXEC: Final[int]

    POSIX\_SPAWN\_CLOSEFROM: Final[int]

    def timerfd\_create(clockid: int, /, \*, flags: int = 0) -> int: ...

    def timerfd\_settime(

        fd: FileDescriptor, /, \*, flags: int = 0, initial: float = 0.0, interval: float = 0.0

    ) -> tuple[float, float]: ...

    def timerfd\_settime\_ns(fd: FileDescriptor, /, \*, flags: int = 0, initial: int = 0, interval: int = 0) -> tuple[int, int]: ...

    def timerfd\_gettime(fd: FileDescriptor, /) -> tuple[float, float]: ...

    def timerfd\_gettime\_ns(fd: FileDescriptor, /) -> tuple[int, int]: ...

if sys.version\_info >= (3, 13) or sys.platform != "win32":

    # Added to Windows in 3.13.

    def fchmod(fd: int, mode: int) -> None: ...

if sys.platform != "linux":

    if sys.version\_info >= (3, 13) or sys.platform != "win32":

        # Added to Windows in 3.13.

        def lchmod(path: StrOrBytesPath, mode: int) -> None: ...