AAI501 – Introduction to Artificial Intelligence

(Applied Artificial Intelligence)

University of San Diego

Symptom-Based Disease Classification Using NLP

Submitted to: Instructor Dr Ankur Bist

Submitted by:

Group 3

Ved Prakash Dwivedi

Bharath TS

Manu Malla

1. INTRODUCTION

1.1 Background and Motivation

With the increasing digitization of healthcare records and the proliferation of symptom-checker platforms, there is a growing demand for intelligent systems that can aid in preliminary disease diagnosis based on textual symptom descriptions. In regions where access to medical professionals is limited or delayed, AI-driven diagnostic support systems can bridge the gap by offering early warning signs, guiding patients towards appropriate care, and reducing the diagnostic burden on healthcare workers.

The use of machine learning in healthcare, particularly in *symptom-based disease classification*, has shown promising results. By training algorithms to detect patterns from previously recorded symptoms and associated diseases, we can build models that assist in identifying probable conditions from free-text symptom descriptions. This approach not only enhances efficiency but also promotes scalability and consistency in diagnosis.

1.2 Problem Statement

The primary objective of this project is to build a classification model that can accurately predict the most likely disease based on a given set of symptoms described in textual format. The project utilizes the **Symptom2Disease** dataset, which contains a collection of disease labels and their corresponding symptom descriptions.

Key challenges include:

- Overlap in symptoms between multiple diseases (e.g., fever appears in flu, dengue, and COVID-19).
- Ambiguity and inconsistency in natural language used to describe symptoms.
- Imbalance in disease label distribution, which may bias the model toward more frequent classes.

1.3 Objective

The goal of this project is to: To develop an accurate, interpretable, and scalable AI system that leverages multiple NLP and ML models—including Logistic Regression with TF-IDF, LSTM/GRU networks, and Transformer-based BERT—to predict diseases from

patient-reported symptoms, enabling faster preliminary screening, improved healthcare accessibility, and better clinical decision support.

1.4 Scope of Work

This report focuses on:

- Data preprocessing and cleaning techniques
- Exploratory visualizations and statistical summaries
- Implementation of the baseline traditional machine learning model (Logistic Regression) and advanced neural network models(LSTM, GRU & BERT)
- Interpretation of model predictions and identification of weaknesses
- Recommendations for future work, deployment and real world adaptation

1.5 Tools and Technologies Used

- **Programming Language:** Python 3.x
- **Libraries:** `pandas`, `scikit-learn`, `nltk`, `matplotlib`, `wordcloud, Tensorflow, pytorch, HuggingFace, keras`
- **Model:** Logistic Regression, LSTM, GRU and BERT
- **Vectorizer:** TF-IDF (Term Frequency–Inverse Document Frequency), Embeddings, Encodings
- **Evaluation Metrics:** Precision, Recall, F1 Macro score, Classification Report

2. DATA PREPROCESSING AND CLEANING TECHNIQUES

2.1 Overview

Data preprocessing is a critical step in any machine learning pipeline, especially when working with natural language data. The quality of input data directly affects the model's performance. For the *Symptom2Disease* dataset, which consists of symptom descriptions and corresponding disease labels, it was essential to clean and standardize the text data to improve feature extraction and model training accuracy.

2.2 Dataset Description

The dataset comprises two main columns:

- **`text`**: A free-form textual description of symptoms.
- **`label`**: The disease diagnosis associated with the symptoms.

Before model training, several preprocessing steps are performed to ensure data consistency, reduce noise, and extract relevant features from the raw text.

2.3 Steps in Data Cleaning and Preprocessing

2.3.1 Removing Unnamed Columns and renaming the columns:

The data set contains extra columns like `Unnamed: 0`. These are usually row indices or artifacts and are irrelevant for modeling. Dropping it ensures that only meaningful data is preserved. Hence they are dropped from the dataset.

The dataset's original column names may be non-descriptive or inconsistent. Renaming to `"label"` and `"text"` makes the dataset easier to interpret and standardizes the schema for further processing.

- **`label`** → the target variable (disease name or class label).
- **`text`** → the input feature containing symptom descriptions.

```
symptom_df = symptom_df.drop(columns=["Unnamed: 0"])
symptom_df.columns = ["label", "text"]
```

2.3.2 Label Encoding:

Before training a machine learning model, it is essential to represent categorical target variables (labels) in a numerical format. Most machine learning algorithms, including **Logistic Regression**, require numerical values for computation. The `"label"` column in the dataset contains disease names as strings, which must be converted into integers.
Each unique disease name is assigned to a unique integer using Labelencoder which transforms the categorial labels into integer codes.

Fitting and transforming labels learns the mapping between the label names and integers(fit) and applies the transformation to convert all labels into numbers.

The transformed labels are stored in a new column.

```
# Encode labels
label_encoder = LabelEncoder()
symptom_df["encoded_label"] = label_encoder.fit_transform(symptom_df["label"])
```

2.4 Exploratory Data Analysis:

Exploratory Data Analysis for Symptom2Disease data set focuses on examining both the Symptom text data and the disease labels to gain insights for preprocessing and modelling.

2.4.1 Class Distribution:

The Class distribution for the data set checks for all the unique disease labels present in the data set. This helps in understanding the variety of disease classes that the model will need to classify during the training.

```
symptom_df['label'].unique()
```

The data from the ‘label’ column from the symptom data frame which contains the disease names is checked for all unique class labels by removing the duplicates.

2.4.2 Unique classes count:

The unique class count refers to determining how many distinct disease categories exist in the data set which is essential for understanding the classification scope before model training.

```
symptom_df['label'].nunique()
```

2.4.3 Dataset Shape and value count analysis:

The shape of the data set helps to assess the size and complexity of the dataset which helps in validating the dataset for expected number of entries after cleaning. In the symptom2disease set this helps to know how many symptom-disease pairs exist and how many feature columns are available for modelling.

The value count method returns the frequency of each unique label in the label column showing the number of samples belonging to each disease category. This provides insight into class distribution allowing detection of class imbalance which is a critical factor in classification model performance. In the symptom2disease data set this check ensures that disease classes are represented adequately. If some diseases have significantly fewer records, balancing techniques like oversampling, undersampling or class weight adjustments may be considered during training.

```
print("shape of the dataset", symptom_df.shape)
print(symptom_df['label'].value_counts())
```

```
shape of the dataset (1200, 3)
label
Psoriasis                          50
Varicose Veins                     50
Typhoid                            50
Chicken pox                        50
Impetigo                           50
Dengue                             50
Fungal infection                   50
Common Cold                        50
Pneumonia                          50
Dimorphic Hemorrhoids              50
Arthritis                          50
Acne                               50
Bronchial Asthma                   50
Hypertension                       50
Migraine                           50
Cervical spondylosis               50
Jaundice                           50
Malaria                            50
urinary tract infection            50
allergy                            50
gastroesophageal reflux disease    50
drug reaction                      50
peptic ulcer disease               50
diabetes                           50
Name: count, dtype: int64
```

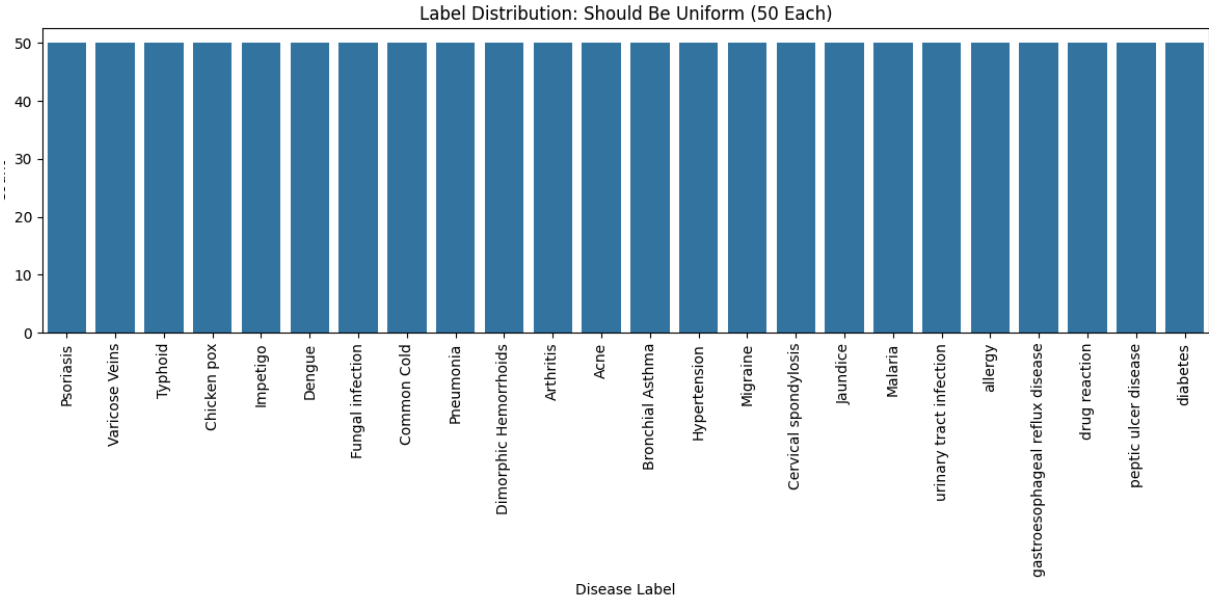
The shape of the data set is (1200,3); means that there are 1200 records and 3 columns. The value count of 50 shows that each disease has 50 records.

2.4.4 Confirm class Balance:

The class imbalance verifies the dataset has balanced class distribution ensuring each disease class has an equal number of records. Balance datasets help classification models learn fairly across all categories without bias toward majority classes.
Visualization with bar plots graphs to display the distribution of samples per class confirms the class balance where if some bars are higher or lower indicates the class imbalance which might require resampling.

```
# Confirm class balance
label_counts = symptom_df['label'].value_counts()
print(label_counts)

# Plotting for visual confirmation
plt.figure(figsize=(12,6))
sns.barplot(x=label_counts.index, y=label_counts.values)
plt.xticks(rotation=90)
plt.title("Label Distribution: Should Be Uniform (50 Each)")
plt.ylabel("Count")
plt.xlabel("Disease Label")
plt.tight_layout()
plt.show()
```

The data set is balanced and most diseases have the same number of samples. Also, each class represents specific disease and the number of samples per class is uniform which is beneficial for training classification models without bias towards a specific disease.



2.4.5 Word cloud analysis:

The word cloud gives a visual representation of the most frequently occurring word across all symptom descriptions in the dataset. The size of each word reflects how often it appears- larger words are more common.

Top Frequent Terms:

These are the words that most often across all symptoms description after preprocessing.

- "skin", "also", "pain", "experiencing", "headache", "fever", "feel" These are the most prominent, suggesting:

- "skin: might be a common context or location for multiple symptoms (e.g., rashes, itching).
- "pain": is a general symptom occurring across many disease types.
- "fever" and "headache" are classic general indicators of illness, commonly reported across diverse conditions.
- "also", "experiencing", "feel", and "get" reflect how patients describe their symptoms (contextual narrative language).

Semantic Clues: Semantic clues in the word cloud analysis refer to meaning based patterns in the terms-essentially what the words tell us about how people describe their symptoms and what type of information is being conveyed.

- There's a mix of anatomical terms (e.g. throat, chest, arm, neck, back, joint) and symptom descriptors (e.g., itchy, sore, painful, dizzy, exhausted), showing that people mention both *where* and *what* they are experiencing.

Common Symptoms:
Words like - "rash", "coughing", "nausea", "throat", "muscle", "dizziness", "fatigue", "sore", "joint" indicate that these symptoms recur across different classes (conditions).

Subjective Expressions:
Words such as- "quite", "really", "frequently", "recently", "difficult", "occasionally" reflect subjective frequency or intensity, which could provide signals for condition severity or temporal trends.

This Tells us -
Rich Vocabulary: The dataset contains natural-language symptom descriptions, not just clinical keywords great for training NLP models.
Overlap Across Classes: Frequent use of common symptom words may indicate semantic overlap between classes, which could make classification harder.
Potential Need for Contextual Models: Since terms like _pain_ and _fever_ are nonspecific, it highlights the need for models that understand contextual cues (e.g., LSTM, GRU, BERT).

2.4.6 Word cloud individual symptoms:

A visualization is generated with a separate word cloud for each disease label which allows to identify symptoms and vocabulary patterns unique to individual diseases which may not be in the aggregated word cloud.

The preprocessing steps are as below:
- Label Filtering: The data set is split into subsets by unique disease labels.

- Text Aggregation per Label – All symptom descriptions for a given label were concatenated into one string.
- Text Cleaning: Applied NLTK stopwords to remove non-informative words.
- Uniform Formatting: Lowercased text for consistent term matching.

Output: A single multi-plot figure is used to display the output and individual images for each of the labels are saved to a separate folder.

Interpretation:

- Each sub plot highlights the most common terms for that specific disease label.
- Comparison across the bales allows detection of Unique symptom terms (e.g *wheezing* for respiratory conditions).
- Shared high-frequency terms (e.g., *pain*, *fever*) indicating possible symptom overlap between diseases.

Psoriasis:

The *Psoriasis* word cloud illustrates the most frequent symptom-related terms reported in the dataset for this condition. The largest words indicate the highest frequency, making it easy to identify the most prominent descriptors of the disease.

Key Observations:

- *Dominant Terms:* “itching”, “skin”, “rash” appear most prominently, reflecting the hallmark dermatological features of psoriasis.
- *Supporting Symptoms:* Words like “scaly”, “redness”, “patches” and “lesions” further emphasize the chronic inflammatory skin nature of the condition.
- *Semantic Clues:* The strong clustering of skin-related terms indicates that psoriasis symptom descriptions are highly focused on visible and tactile changes in the skin, with less emphasis on systemic symptoms.

Interpretation:
The word cloud clearly shows that the dataset's psoriasis entries are symptomatically consistent with medical literature, where skin irritation and scaling are key identifiers. These terms can serve as strong predictive features in a text classification model.

Varicose Veins:

The varicose veins word cloud presents the most frequently occurring terms in the dataset's symptom descriptions for this condition. Word prominence reflects frequency, enabling quick identification of the most common descriptors associated with the disease.

Key Observations:

- Dominant Terms: *pain*, *legs*, and *swelling* appear most prominently, indicating that discomfort and localized changes in the lower extremities are primary symptoms.
- Supporting Symptoms: Words like *veins*, *twisted*, *enlarged*, and *discoloration* emphasize the visible structural changes to superficial veins, often accompanied by aesthetic and functional concerns.
- Semantic Clues: The frequent co-occurrence of terms such as *heaviness* and *aching* points to the symptomatic burden experienced during prolonged standing or walking.

Interpretation:

The dataset's varicose vein entries closely reflect typical clinical descriptions—localized leg pain, visible venous enlargement, swelling, and skin discoloration. These descriptors can serve as high-value predictive features in an NLP classification model, particularly in differentiating vascular conditions from dermatological or systemic disorders.

Typhoid:

The typhoid word cloud illustrates the most common terms present in the dataset's symptom descriptions for this condition. Larger words denote higher frequency, offering a quick overview of the dominant linguistic patterns associated with typhoid fever.

Key Observations:

- Dominant Terms: *fever*, *headache*, and *abdominal* appear most prominently, underscoring the systemic and gastrointestinal nature of the illness.

- Supporting Symptoms: Words such as *pain*, *constipation*, *diarrhea*, *fatigue*, and *loss* (often part of "loss of appetite") reflect the broad impact of the infection on digestive health and overall energy levels.

- Semantic Clues: The co-occurrence of digestive terms (*abdominal*, *constipation*, *diarrhea*) with systemic ones (*fever*, *headache*, *fatigue*) indicates a dual symptom profile—affecting both gastrointestinal and whole-body systems.

Interpretation:

The typhoid entries in the dataset align closely with clinical expectations, where persistent fever, abdominal discomfort, and digestive irregularities are hallmark features. These frequently occurring terms offer strong predictive signals for a classification model aiming to distinguish typhoid from other febrile or gastrointestinal conditions.

Chicken pox

The chicken pox word cloud visualizes the most frequently occurring terms from the dataset's symptom descriptions for this condition. Word prominence corresponds to frequency, allowing quick identification of the most defining descriptors.

Key Observations:

- Dominant Terms: *rash*, *fever*, and *itchy* are the most prominent, reflecting the hallmark presentation of chicken pox as an infectious, rash-producing illness.

- Supporting Symptoms: Words such as *spots*, *blisters*, *fatigue*, and *loss* (often as “loss of appetite”) highlight both the visible skin manifestations and accompanying systemic effects.

- Semantic Clues: The strong clustering of dermatological terms (*rash*, *spots*, *blisters*, *itchy*) indicates that patient-reported descriptions focus heavily on visible skin changes, with secondary emphasis on fever and general malaise.

Interpretation:

The chicken pox symptom profile in the dataset matches standard medical descriptions—characterized by an itchy, blister-like rash, mild to moderate fever, and generalized fatigue. These highly frequent, distinctive terms can serve as robust predictors for automated text-based disease classification models, especially in differentiating viral exanthems from other skin-related illnesses.

Impetigo:

The impetigo word cloud displays the most frequent symptom-related terms from the dataset’s descriptions for this bacterial skin infection. Word prominence indicates frequency, enabling quick recognition of the defining descriptors for the condition.

Key Observations:

- Dominant Terms: *blisters*, *skin*, and *rash* are the largest terms, reflecting the highly visible and localized nature of impetigo lesions.

- Supporting Symptoms: Words like *itchy*, *red*, *crust*, and *sores* emphasize the infection's hallmark presentation—itchy, red spots that develop a honey-colored crust.

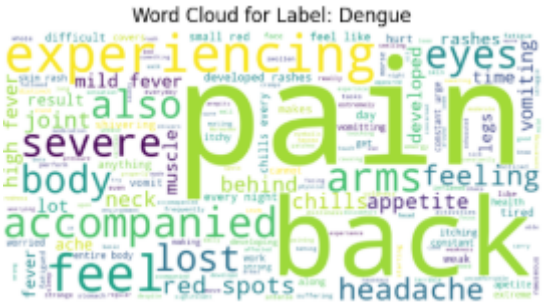
- Semantic Clues: The frequent association of skin-related descriptors with texture terms (*crust*, *scaly*) indicates strong patient focus on the evolving appearance of lesions rather than systemic symptoms.

Interpretation:

The dataset's impetigo entries are symptomatically consistent with clinical literature, where this superficial bacterial infection is marked by itchy red sores, fluid-filled blisters, and crust formation—commonly around the nose and mouth. These descriptive terms are highly distinctive and could serve as strong features in NLP-based disease classification, particularly for differentiating bacterial skin infections from viral rashes or inflammatory dermatoses.

Dengue:

The dengue word cloud visualizes the most frequent terms in the dataset's symptom descriptions for this mosquito-borne viral infection. Word prominence corresponds to term frequency, offering an immediate overview of the defining descriptors.



Key Observations:

- Dominant Terms: *fever*, *pain*, and *headache* appear most prominently, reflecting the acute febrile nature of dengue.
- Supporting Symptoms: Words such as *rash*, *nausea*, *vomiting*, *fatigue*, and *joint* highlight the multisystem involvement, including gastrointestinal and musculoskeletal manifestations.
- Semantic Clues: The co-occurrence of *joint* and *muscle* with *pain* aligns with the colloquial term “breakbone fever,” a common descriptor for the severe body aches associated with dengue.

Interpretation:

The dengue entries in the dataset align closely with clinical presentations—sudden onset of high fever, severe headache, muscle and joint pain, and sometimes rash or gastrointestinal upset. The high frequency of these terms indicates their value as predictive features for distinguishing dengue from other febrile illnesses in NLP-based classification models.

Fungal Infection:

The fungal infection word cloud represents the most frequently occurring terms in the dataset’s symptom descriptions for this condition. Word prominence is proportional to frequency, enabling quick recognition of the most characteristic descriptors.

Key Observations:

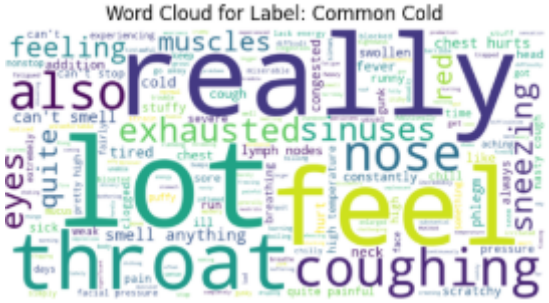
- Dominant Terms: *itching*, *skin*, and *redness* appear most prominently, highlighting the hallmark inflammatory and irritative nature of fungal skin infections.
- Supporting Symptoms: Words such as *rash*, *patches*, *scaly*, and *blisters* emphasize the visible dermatological changes and textural alterations in the affected area.
- Semantic Clues: The strong concentration of skin-related terminology suggests that patient descriptions focus primarily on localized cutaneous manifestations, with little mention of systemic effects.

Interpretation:

The symptom profile for fungal infections in the dataset matches well with clinical literature—characterized by itchy, red, and scaly patches, sometimes accompanied by blisters or peeling skin. These distinct terms serve as reliable predictive features for text classification models, particularly in differentiating fungal skin conditions from bacterial infections or allergic dermatitis.

Common Cold:

The common cold word cloud presents the most frequently used terms from the dataset's symptom descriptions for this widespread viral respiratory infection. Word prominence indicates frequency, offering an immediate view of the dominant descriptors.



Key Observations:

- Dominant Terms: *sneezing*, *cough*, and *throat* are the most prominent, reflecting hallmark upper respiratory symptoms.

- Supporting Symptoms: Words such as *congestion*, *runny*, *nose*, *headache*, and *fatigue* further emphasize nasal and throat irritation along with mild systemic discomfort.

- Semantic Clues: The frequent co-occurrence of nasal symptoms (*runny*, *congestion*, *nose*) with throat-related terms (*sore*, *throat*) indicates that patient-reported descriptions focus on upper airway inflammation, consistent with the nature of the condition.

Interpretation:

The dataset's common cold entries align with well-established clinical profiles—characterized by sneezing, coughing, sore throat, and nasal congestion, often accompanied by mild headache or fatigue. These high-frequency descriptors can serve as effective features for NLP-based models, particularly for differentiating the common cold from more severe respiratory infections like influenza or bronchitis.

Pneumonia:

The pneumonia word cloud visualizes the most frequently occurring terms from the dataset's symptom descriptions for this serious lower respiratory tract infection. Word size corresponds to term frequency, making the most common descriptors immediately recognizable.

Key Observations:

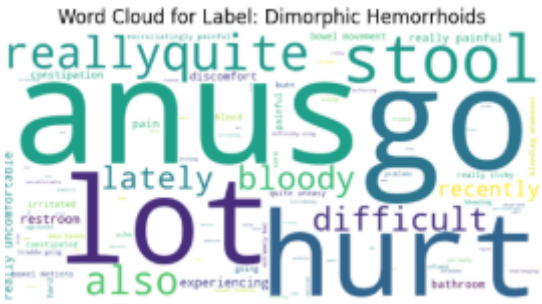
- Dominant Terms: *cough*, *fever*, and *chest* are the most prominent, highlighting the primary respiratory and systemic manifestations of pneumonia.
- Supporting Symptoms: Words such as *pain*, *phlegm*, *shortness*, *breath*, and *fatigue* indicate lung inflammation, mucus production, and reduced respiratory efficiency.
- Semantic Clues: The clustering of respiratory distress terms (*shortness*, *breath*, *chest*) with systemic illness indicators (*fever*, *fatigue*) reflects a combination of localized lung infection and whole-body immune response.

Interpretation:

The symptom profile for pneumonia in the dataset aligns with clinical understanding—characterized by persistent cough, high fever, chest pain, and breathing difficulties. These distinctive features can be leveraged in predictive models to differentiate pneumonia from upper respiratory illnesses like bronchitis or common cold, as well as from non-infectious respiratory issues such as asthma.

Dimorphic Hemorrhoids:

The word cloud for dimorphic hemorrhoids highlights the most frequently reported terms in the dataset for this anorectal condition. Term prominence directly corresponds to symptom frequency, making it easier to identify key descriptors.



Key Observations:

- Dominant Terms: *pain*, *bleeding*, and *rectal* are the largest, reflecting the primary clinical features of hemorrhoids.

- Supporting Symptoms: Words such as *itching*, *swelling*, *discomfort*, and *lumps* further indicate irritation and inflammation of the anal region.

- Semantic Clues: The strong presence of bleeding-related terms combined with tactile descriptors (*lumps*, *swelling*) suggests patients' focus on both visible and sensory symptoms. The term *dimorphic* indicates both internal and external hemorrhoid involvement.

Interpretation:

The dataset's representation of dimorphic hemorrhoids is consistent with medical literature—characterized by pain, bleeding during bowel movements, swelling, and itching. These distinct and localized symptoms can serve as reliable predictors for text classification models, helping differentiate hemorrhoids from other gastrointestinal or anorectal disorders such as anal fissures or colorectal polyps.

Arthritis:

The arthritis word cloud visualizes the most frequent terms associated with this joint-related condition in the dataset. Larger words represent higher frequency, offering a quick visual insight into symptom prominence.

Key Observations:

- Dominant Terms: *pain*, *joint*, and *swelling* are the most prominent, highlighting the hallmark inflammatory and degenerative symptoms of arthritis.

- Supporting Symptoms: Terms like *stiffness*, *tenderness*, *movement*, and *inflammation* reinforce the chronic, mobility-limiting nature of the disease.

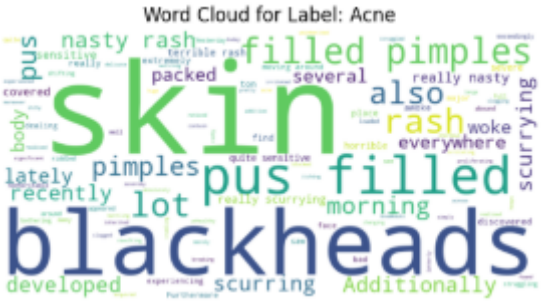
- Semantic Clues: The clustering of terms focused on structural and functional limitations (*joint*, *movement*, *stiffness*) suggests that symptom reporting centers around physical impairment and localized inflammation rather than systemic issues.

Interpretation:

The dataset's arthritis entries align closely with clinical expectations—characterized by joint pain, swelling, and stiffness. These features are central to diagnosing arthritis and distinguishing it from other musculoskeletal conditions such as bursitis, tendinitis, or fibromyalgia. The strong presence of mobility-related terms could also serve as valuable predictors in a symptom-based classification model.

Acne:

The acne word cloud captures the most common terms describing this skin condition within the dataset. Word prominence reflects symptom frequency, providing a visual overview of how the disease is described.



Key Observations:

- Dominant Terms: *pimples*, *skin*, and *spots* are the most prominent, highlighting the visible nature of acne.

- Supporting Symptoms: Terms such as *blackheads*, *whiteheads*, *redness*, and *oily* indicate specific lesion types and associated skin changes.

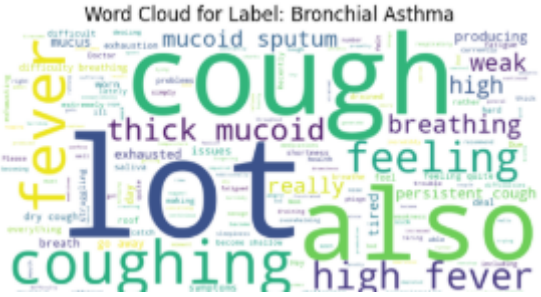
- Semantic Clues: The clustering of lesion-related terms (*pimples*, *blackheads*, *whiteheads*) alongside descriptors of skin condition (*oily*, *redness*) reflects the dermatological and aesthetic impact of acne.

Interpretation:

The dataset's acne entries are consistent with dermatological literature, where comedones (blackheads and whiteheads) and inflamed papules/pustules are key identifiers. The focus on visible and tactile symptoms makes acne particularly well-suited for detection via both text-based models and image-based diagnostic tools.

Bronchial Asthma:

The bronchial asthma word cloud highlights the most common symptom terms reported for this respiratory condition in the dataset. Larger words represent higher frequency, offering a quick view of symptom prominence.



Key Observations:

- Dominant Terms: *breathlessness*, *wheezing*, and *cough* stand out as the most frequent, reflecting the primary respiratory symptoms of asthma.

- Supporting Symptoms: Words like *chest*, *tightness*, *shortness*, and *mucus* indicate additional clinical features such as airway constriction and sputum production.

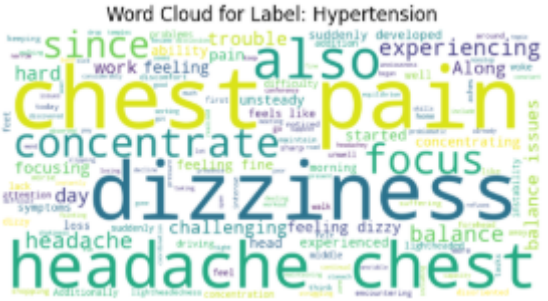
- Semantic Clues: The strong clustering of respiratory distress terms (*breathlessness*, *wheezing*, *cough*) suggests acute episodes and chronic airway inflammation are central to patient-reported symptoms.

Interpretation:

The dataset's bronchial asthma entries align with clinical understanding of the disease—characterized by recurrent episodes of breathlessness, wheezing, and chest tightness. These features are crucial in distinguishing asthma from other respiratory illnesses such as chronic bronchitis or pneumonia. Such symptom patterns also provide high-value input for machine learning models aimed at respiratory disease classification.

Hypertension:

The hypertension word cloud visualizes the most frequent terms reported in the dataset for this cardiovascular condition. Larger terms denote higher occurrence in patient symptom descriptions.



Key Observations:

- Dominant Terms: *headache*, *dizziness*, and *fatigue* appear prominently, reflecting common non-specific yet frequent symptoms linked with elevated blood pressure.

- Supporting Symptoms: Terms such as *blurred vision*, *chest*, and *pain* indicate possible secondary effects of hypertension on the eyes and cardiovascular system.

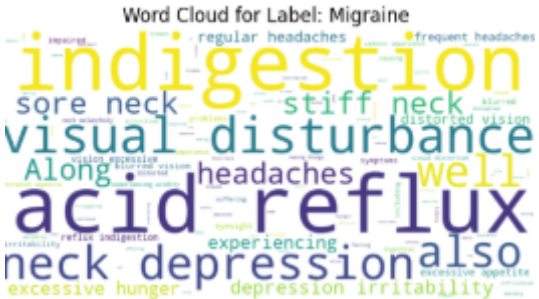
- Semantic Clues: The clustering of neurological symptoms (*headache*, *dizziness*, *blurred vision*) suggests patient awareness often comes from discomfort or sensory disturbance rather than direct awareness of blood pressure itself.

Interpretation:

The dataset's hypertension entries match well with clinical literature, where hypertension is often asymptomatic but may present with headaches, dizziness, and visual disturbances during elevated episodes. These terms can be strong predictive features in classification models but must be interpreted alongside medical measurements for accuracy, since they are not unique to hypertension alone.

Migraine:

The migraine word cloud depicts the most frequently occurring terms in the dataset associated with this neurological disorder. Larger words represent higher term frequency, allowing rapid identification of key descriptors.



Key Observations:

- Dominant Terms: *headache*, *nausea*, and *vomiting* appear most prominently, consistent with the classic presentation of migraine episodes.

- Supporting Symptoms: Words such as *light*, *sensitivity*, *throbbing*, *vision*, and *aura* emphasize sensory disturbances and the pulsating nature of migraine pain.

- Semantic Clues: The clustering of sensory-related terms (*light sensitivity*, *vision changes*, *aura*) reflects the neurological origin of the condition and distinguishes migraine from tension-type headaches.

Interpretation:

The dataset's migraine entries closely align with clinical descriptions, where severe headache often co-occurs with nausea, vomiting, and sensitivity to light or sound. These symptom patterns are distinctive and can serve as highly discriminative features for a symptom-based classification model, enabling differentiation from other headache disorders.

Cervical spondylosis:

The cervical spondylosis word cloud highlights the most common terms from the dataset related to this degenerative spine condition. Larger words indicate higher frequency of occurrence in patient-reported symptom descriptions.

Key Observations:

- Dominant Terms: *neck*, *pain*, and *stiffness* appear most prominently, reflecting the hallmark musculoskeletal features of cervical spondylosis.

- Supporting Symptoms: Words such as *tingling*, *shoulder*, *numbness*, and *weakness* point to nerve root compression and radiating discomfort into the arms and shoulders.

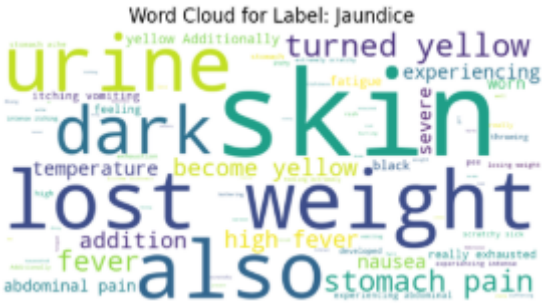
- Semantic Clues: The frequent co-occurrence of neurological terms (*numbness*, *tingling*) alongside pain-related terms suggests both mechanical and nerve-involvement aspects are well-represented in the dataset.

Interpretation:

The dataset's cervical spondylosis entries are symptomatically consistent with medical literature, where chronic neck pain, stiffness, and sensory disturbances are key indicators. These terms can serve as strong predictive features for spine-related condition classification, particularly in distinguishing cervical spondylosis from other musculoskeletal disorders like arthritis or slipped disc.

Jaundice:

The jaundice word cloud displays the most frequent symptom-related terms from the dataset for this liver-related condition. The prominence of each word corresponds to its frequency in patient-reported symptom descriptions.



Key Observations:

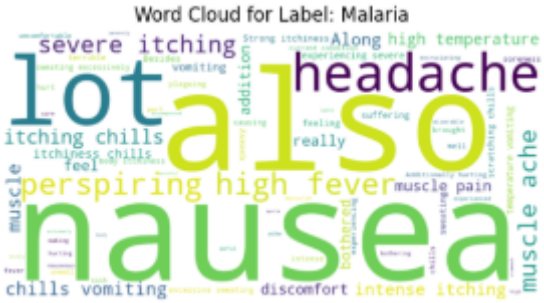
- Dominant Terms: *yellowish*, *skin*, and *eyes* appear most prominently, capturing the hallmark sign of jaundice — yellow discoloration of the skin and sclera.
- Supporting Symptoms: Words like *dark urine*, *fatigue*, *itching*, and *nausea* suggest liver dysfunction and bile pigment buildup, common in jaundice cases.
- Semantic Clues: The clustering of skin and urine-related terms indicates the dataset's focus on visible and measurable clinical features rather than vague systemic complaints.

Interpretation:

The dataset's jaundice entries strongly align with medical definitions, where bilirubin accumulation manifests as yellowing of skin/eyes, often accompanied by dark urine and fatigue. Such clear symptom markers make jaundice relatively straightforward to detect in a text-based classification model compared to conditions with more diffuse symptom patterns.

Malaria:

The malaria word cloud visualizes the most frequently reported symptom terms in the dataset for this mosquito-borne infectious disease. The size of each word reflects its relative occurrence frequency.



Key Observations:

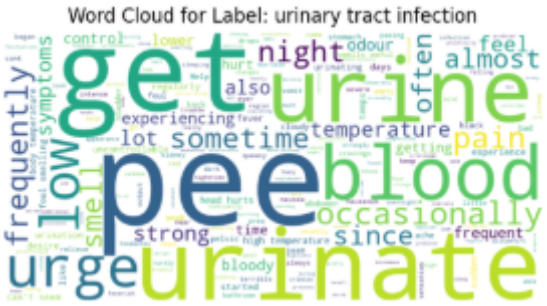
- Dominant Terms: *fever*, *chills*, and *sweating* are the most prominent, representing the hallmark cyclic symptom pattern of malaria.
- Supporting Symptoms: Words like *headache*, *nausea*, *vomiting*, and *fatigue* highlight the systemic nature of the infection, while *shivering* further reinforces the fever cycle description.
- Semantic Clues: The repeated clustering of fever-related and weakness-related terms indicates that malaria symptom descriptions are centered on acute systemic illness with temperature fluctuations.

Interpretation:

The dataset's malaria entries align well with medical literature, where high fever with chills and sweats are key diagnostic cues, often accompanied by malaise, headaches, and gastrointestinal discomfort. These distinctive symptom patterns make malaria relatively recognizable in a text-based classification model, particularly when differentiating it from other febrile illnesses like dengue or typhoid.

Urinary tract infection:

The urinary tract infection word cloud presents the most common symptom-related terms in the dataset for this bacterial infection affecting the urinary system. The larger the word, the more frequently it appears in patient symptom descriptions.



Key Observations:

- Dominant Terms: *burning*, *urination*, and *pain* are the most prominent, highlighting the characteristic dysuria (painful urination) associated with UTIs.
- Supporting Symptoms: Words like *frequent*, *urge*, *lower abdomen*, and *discomfort* indicate urinary urgency and pelvic pain, common in clinical presentations.
- Semantic Clues: The clustering of urination-related words suggests a symptom profile highly localized to the urinary system, with minimal systemic symptoms compared to other infections.

Interpretation:

The dataset's UTI entries are consistent with medical literature, emphasizing dysuria, urgency, and pelvic pain as primary indicators. The localized nature of these terms provides a strong predictive feature set for classification models, enabling clear separation from systemic febrile illnesses or gastrointestinal conditions.

Allergy:

The allergy word cloud displays the most frequently mentioned terms in the dataset for allergic reactions. Word size corresponds to the frequency of occurrence in reported symptom descriptions.

Key Observations:

- Dominant Terms: *sneezing*, *itching*, and *rash* appear most prominently, reflecting the typical hypersensitivity response involving skin and respiratory irritation.
- Supporting Symptoms: Words like *runny nose*, *redness*, *watery eyes*, and *swelling* highlight the multi-system nature of allergic responses, affecting both mucosal and dermal surfaces.
- Semantic Clues: The combination of nasal, ocular, and skin-related terms suggests a profile consistent with allergic rhinitis or mild skin allergies rather than severe anaphylaxis, which would include terms like *breathing difficulty* or *shock*.

Interpretation:

The dataset's allergy entries align closely with medical literature on common allergic reactions, especially seasonal allergies and contact dermatitis. These terms provide strong categorical features for text classification models, helping distinguish allergies from infections (e.g., common cold) that share some overlapping symptoms like sneezing or nasal congestion.

Gastroesophageal reflux disease:

The GERD word cloud visualizes the most common terms associated with this chronic digestive disorder in the dataset. Larger words indicate higher frequency in patient symptom descriptions.

Key Observations:

- Dominant Terms: *heartburn*, *acid*, and *reflux* are the most prominent, directly reflecting the primary physiological process of stomach acid flowing back into the esophagus.

- Supporting Symptoms: Words like *chest pain*, *regurgitation*, *sour taste*, *indigestion*, and *bloating* highlight additional discomforts often experienced by GERD patients.

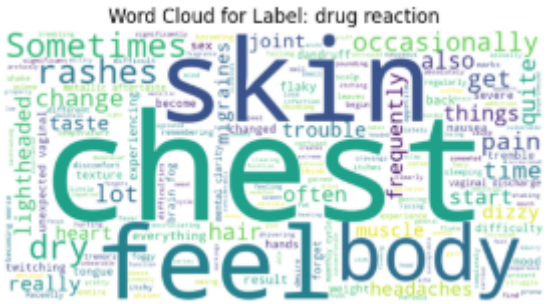
- Semantic Clues: The focus on upper gastrointestinal and chest-related terms confirms a localized symptom profile centered around esophageal irritation, with minimal systemic or unrelated symptom mentions.

Interpretation:

The dataset's GERD entries are consistent with established medical descriptions, emphasizing acid reflux, heartburn, and regurgitation as hallmark indicators. These symptom terms can serve as strong predictors in classification models, helping differentiate GERD from cardiovascular chest pain or respiratory conditions, which may share terms like *chest pain* but lack acid-related descriptors.

Drug reaction:

The drug reaction word cloud captures the most frequently occurring symptom terms associated with adverse responses to medications in the dataset. The larger the word, the more often it appears in patient-reported symptom descriptions.



Key Observations:

- Dominant Terms: *rash*, *itching*, and *skin eruption* appear most prominently, indicating that dermatological manifestations are the most common adverse drug reaction indicators.

- Supporting Symptoms: Words like *redness*, *swelling*, *hives*, and *blisters* further emphasize cutaneous hypersensitivity responses.

- Severe Reaction Indicators: Less frequent but critical terms like *shortness of breath*, *fever*, and *anaphylaxis* suggest the presence of potentially life-threatening reactions in a minority of cases.

- Semantic Clues: The clustering of skin-related symptoms points to common mild-to-moderate allergic reactions, while respiratory and systemic symptoms point to severe but less frequent outcomes.

Interpretation:

The dataset's drug reaction entries reflect patterns seen in pharmacovigilance literature, where most adverse reactions manifest as skin rashes and itching, with a small percentage leading to severe systemic effects. These dominant and supporting symptom terms can be strong indicators in predictive models for differentiating drug reactions from infectious rashes (e.g., measles, chickenpox).

Peptic ulcer disease:

The word cloud for peptic ulcer disease illustrates the most frequent terms reported in the dataset for this gastrointestinal condition. Larger words indicate a higher occurrence in symptom descriptions.

Key Observations:

- Dominant Terms: *abdominal pain*, *stomach pain*, and *burning sensation* are the most prominent, directly reflecting the hallmark discomfort caused by ulceration in the stomach or duodenal lining.

- Supporting Symptoms: Terms such as *nausea*, *vomiting*, *indigestion*, *bloating*, and *loss of appetite* provide a broader picture of associated gastrointestinal distress.

- Alarm Features: Less frequent but clinically significant terms like *blood in stool* and *vomiting blood* suggest the inclusion of cases with gastrointestinal bleeding, indicating severe ulcer complications.

- Semantic Clues: The clustering of pain-related and digestion-related terms confirms the dataset's alignment with medical literature, where localized epigastric discomfort is a key differentiator from other abdominal conditions.

Interpretation:

The dataset's peptic ulcer entries highlight a strong symptom profile centered on abdominal pain and burning, consistent with acid-related mucosal damage. These dominant descriptors can be highly predictive for text classification tasks, helping to distinguish PUD from gastritis or gastroesophageal reflux disease, which may share some overlapping symptoms but differ in severity and associated alarm signs.

Diabetes:

The diabetes word cloud captures the most frequently mentioned terms in the dataset associated with this chronic metabolic disorder. The size of each word reflects its relative frequency in patient-reported symptom descriptions.

Key Observations:

- Dominant Terms: *fatigue*, *excessive thirst* (*polydipsia*), and *frequent urination* (*polyuria*) stand out, aligning with classic symptoms caused by hyperglycemia.
- Supporting Symptoms: Words such as *weight loss*, *blurred vision*, and *increased hunger* (*polyphagia*) add depth to the clinical picture and reflect hallmark diagnostic features.
- Complication Indicators: Less frequent terms like *slow healing wounds* and *numbness* suggest peripheral neuropathy and poor circulation, which are chronic complications of poorly controlled diabetes.
- Semantic Clues: The clustering of systemic and urinary-related terms indicates a dataset focus on both metabolic dysregulation and its functional impact on the body.

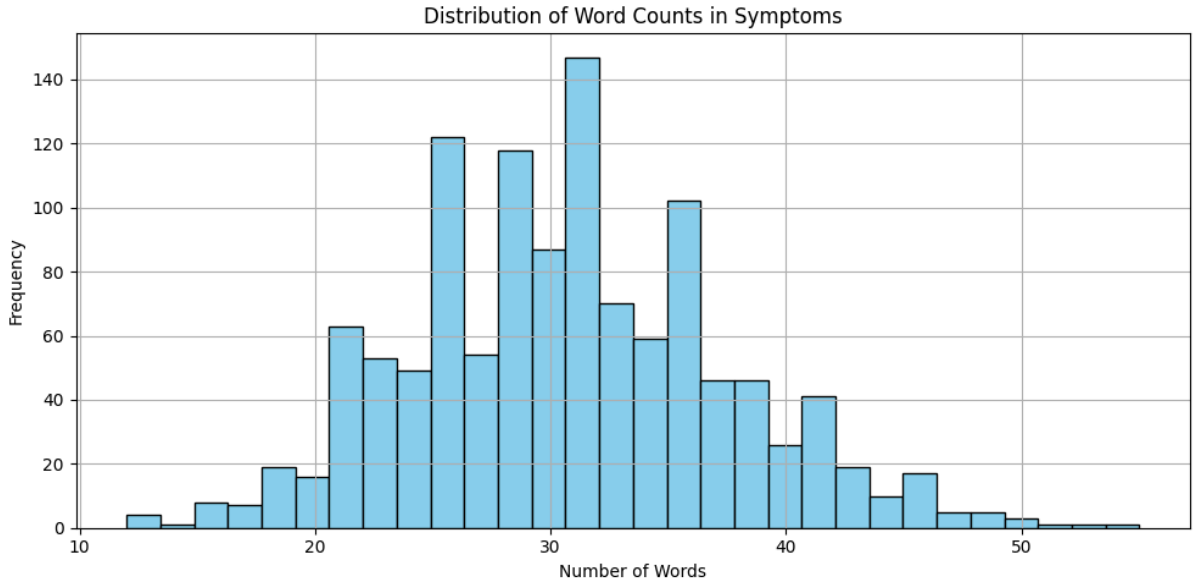
Interpretation:

The dataset's diabetes-related entries are symptomatically consistent with established medical knowledge, emphasizing the "three polys" (polyuria, polydipsia, polyphagia) as primary identifiers. The inclusion of complication-related terms suggests the dataset may capture a range of disease stages, which could aid in building predictive models capable of differentiating early-onset from advanced cases.

2.4.7 Word count distribution:

The distribution of word counts in the dataset shows how verbose or concise symptom descriptions are for various conditions.

Analysis of Word Count Distribution in Symptom Texts:



1. Modal Word Count (Most Frequent Bin):
    - The highest frequency is for symptom descriptions containing around 30–32 words.
    - This suggests that most entries are moderately descriptive but not overly long.

2. Overall Distribution Shape:

    - The distribution is slightly right-skewed, meaning there are fewer extremely long descriptions.
    - There's a bell-like central peak with gradual tapering on both sides, though the right tail is longer.

3. Range and Spread:

    - Word counts vary from under 15 to over 50.
    - The majority of symptoms seem to fall in the 20–40 word range, which aligns well with typical short-form clinical documentation.

4. Outliers:

    - A few entries exceed 50 words, which may indicate overly verbose or compound symptom entries. These could potentially be cleaned or split in preprocessing if needed.

5. Implications for Modeling:
    - The relatively narrow range and lack of extremely short entries suggest less need

for aggressive padding or truncation if using models like RNNs or Transformers.
- For BERT-style models with a max token limit of 512, these descriptions are well within capacity.

2.4.8 Actionable Steps:

Consider normalizing descriptions over 50 words, possibly through summarization or truncation for consistency.

- Short descriptions (<15 words) could be flagged for manual inspection or enrichment if clinical completeness is a concern.

- The word count feature itself may be useful as a predictor or input feature in metadata-augmented models.

3. MODELLING

The dataset contains textual descriptions of symptoms mapped to specific disease labels. Given the nature of the data (short-to-medium length medical symptom phrases with moderate vocabulary size), both traditional machine learning and deep learning approaches are viable.

3.1 Preprocessing Pipeline

Before modelling

Tokenization: Splitting text into words for numerical encoding.
Encoding: Two parallel approaches were used:

- TF-IDF Vectorization for traditional ML.
- Token Indexing & Padding for deep learning models (e.g., LSTM).

3.2 Models consideration:

The Symptom-to-Disease classification task was approached using two distinct modelling paradigms:

1. Traditional Machine Learning Models based on statistical text representation.
2. Deep Learning Models leveraging word embeddings
3. Transformer based models - Pretrained BERT

3.3 Traditional Machine Learning Models:

Traditional models treat text as a bag of words or weighted features without explicitly modelling sequential order. These methods are effective when the dataset contains short, distinct, keyword-rich symptom descriptions.

Logistic Regression with TF-IDF:

- *Input Representation:* Term Frequency–Inverse Document Frequency (TF-IDF) vectors.

- *Reasoning*: Captures the relative importance of medical terms across all samples.

- *Advantages*: High accuracy, interpretable coefficients, low computational cost.
- *Use Case:* Fast, reliable baseline model.

The objective of this model is to classify diseases based on textual symptom descriptions by transforming the text into numerical feature vectors using **Term Frequency–Inverse Document Frequency (TF-IDF)** and applying **Logistic Regression** for multi-class classification.

- Term Frequency (TF): Measures how often a term appears in a document.
- Inverse Document Frequency (IDF): Measures how unique a term is across all documents.
- TF-IDF Vectorization: Combines TF and IDF scores to assign higher weights to important words while reducing the weight of common words.

Data Preprocessing:

1. Label Encoding
    - The `LabelEncoder` from `sklearn.preprocessing` was used to convert categorical disease labels into numerical form (`label_encoded`), enabling the classifier to work with the data.
2. Train-Test Split
    - The dataset is split into training and testing sets using an 80-20 ratio.
    - `stratify` parameter ensures that the distribution of disease labels is preserved in both sets.
    - `random_state=42` ensures reproducibility.
3. Text Vectorization
    - TF-IDF Vectorizer (`TfidfVectorizer`) converts the text into a sparse numerical matrix representing word importance.
    - Stop words (English) were removed to reduce noise.
    - `max_features=3000` was chosen to capture the most informative terms without overfitting.

4. Vectorization Parameters:

- max_features = 500 → Limits vocabulary size to top 500 important words.

Model:

Logistic Regression

- Simplicity and interpretability.
- Ability to handle high-dimensional sparse data.
- Good performance for text classification tasks.

Configuration:

- `max_iter=100`

```
#Preprocessing
le=LabelEncoder()
symptom_df['label_encoded']=le.fit_transform(symptom_df['label'])

#Split training data
X_train, X_test, y_train, y_test = train_test_split(
    symptom_df['text'], symptom_df['label_encoded'], test_size=0.2, random_state=42, stratify=symptom_df['label_encoded'

# TF-IDF Vectorization
tfidf = TfidfVectorizer(stop_words='english', max_features=500)
X_train_vec = tfidf.fit_transform(X_train)
X_test_vec = tfidf.transform(X_test)
```

✓ 0.0s

```
# Logistic regression model training

log_reg = LogisticRegression(max_iter=100)
log_reg.fit(X_train_vec, y_train)

# Predict
y_pred = log_reg.predict(X_test_vec)
```

Evaluation Metrics:

The model was evaluated using:

- **Accuracy Score**: Measures the percentage of correct predictions.

```
# Evaluation
# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

✓ 0.0s

```
Accuracy: 0.9375
```

- **Classification Report**:

- Precision, Recall, F1-Score for each class.

```
# Classification report Test data
print("\nClassification Report Test data:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))
```

[42] ✓ 0.0s

..

Classification Report Test data:

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	10
Arthritis	0.91	1.00	0.95	10
Bronchial Asthma	0.91	1.00	0.95	10
Cervical spondylosis	1.00	1.00	1.00	10
Chicken pox	0.78	0.70	0.74	10
Common Cold	1.00	1.00	1.00	10
Dengue	0.80	0.80	0.80	10
Dimorphic Hemorrhoids	1.00	1.00	1.00	10
Fungal infection	0.91	1.00	0.95	10
Hypertension	1.00	1.00	1.00	10
Impetigo	1.00	1.00	1.00	10
Jaundice	1.00	1.00	1.00	10
Malaria	1.00	1.00	1.00	10
Migraine	1.00	0.90	0.95	10
Pneumonia	0.83	1.00	0.91	10
Psoriasis	1.00	0.90	0.95	10
Typhoid	0.91	1.00	0.95	10
Varicose Veins	1.00	1.00	1.00	10
allergy	1.00	0.70	0.82	10
diabetes	0.83	1.00	0.91	10
...				
accuracy			0.94	240
macro avg	0.94	0.94	0.94	240
weighted avg	0.94	0.94	0.94	240

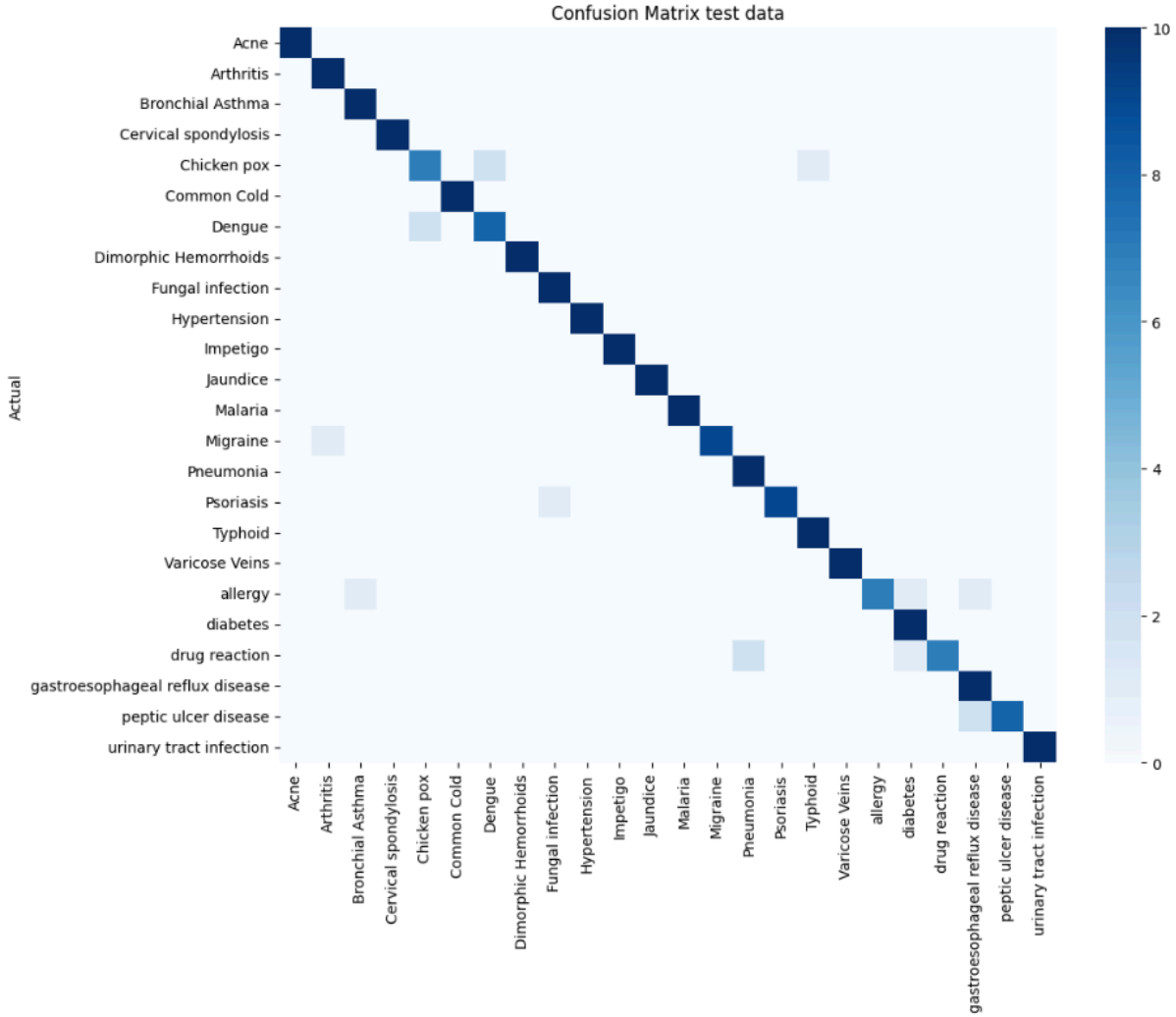
```
# Classification report for Train data

log_reg = LogisticRegression(max_iter=100)
log_reg.fit(X_train_vec, y_train)
y_train_pred=log_reg.predict(X_train_vec)

print("\nClassification Report Train data:\n")
print(classification_report(y_train, y_train_pred, target_names=le.classes_))
```

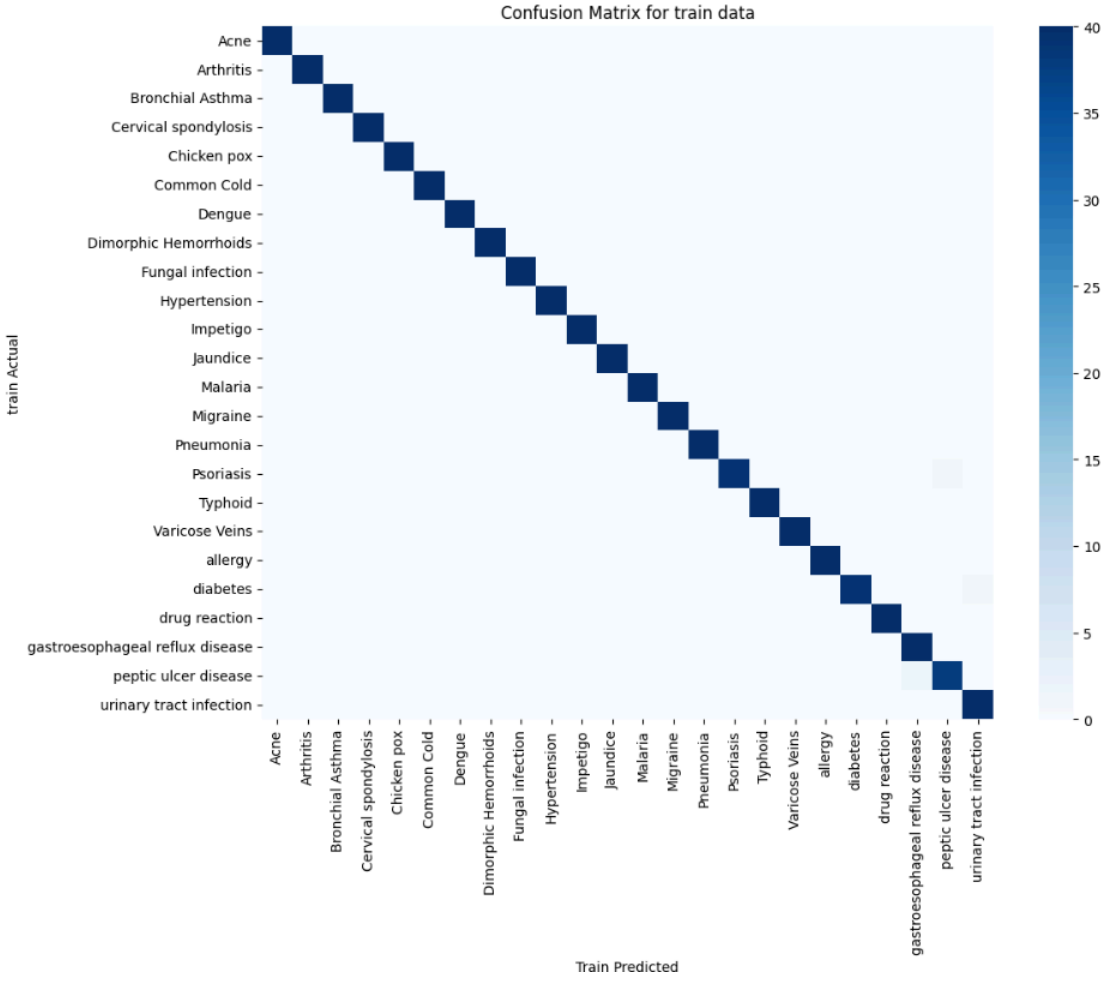
[41] ✓ 0.2s

```
Classification Report Train data:

                              precision    recall  f1-score   support

                        Acne       1.00      1.00      1.00        40
                   Arthritis       1.00      1.00      1.00        40
           Bronchial Asthma        1.00      1.00      1.00        40
       Cervical spondylosis        1.00      1.00      1.00        40
                Chicken pox        1.00      1.00      1.00        40
                Common Cold        1.00      1.00      1.00        40
                     Dengue        1.00      1.00      1.00        40
    Dimorphic Hemorrhoids          1.00      1.00      1.00        40
         Fungal infection          1.00      1.00      1.00        40
             Hypertension          1.00      1.00      1.00        40
                 Impetigo          1.00      1.00      1.00        40
                 Jaundice          1.00      1.00      1.00        40
                  Malaria          1.00      1.00      1.00        40
                 Migraine          1.00      1.00      1.00        40
                Pneumonia          1.00      1.00      1.00        40
                Psoriasis          1.00      0.97      0.99        40
                  Typhoid          1.00      1.00      1.00        40
           Varicose Veins          1.00      1.00      1.00        40
                  allergy          1.00      1.00      1.00        40
                 diabetes          1.00      0.97      0.99        40
...
                 accuracy                               1.00       960
                macro avg       1.00      1.00      1.00       960
             weighted avg       1.00      1.00      1.00       960
```

- **Confusion Matrix**:
    - Visualized with `seaborn` heatmap to analyze prediction patterns.

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING





3.4 Deep Learning Models: LSTM + GRU Hybrid

Description

Deep learning models capture sequential dependencies in text, making them effective for symptom descriptions that follow a temporal or narrative structure.

The **LSTM + GRU hybrid model** combines the strengths of both architectures, it processes text data through parallel recurrent pathways, then merges learned features for classification:

- **LSTM (Long Short-Term Memory)** excels at retaining long-term dependencies.

- **GRU (Gated Recurrent Unit)** is computationally lighter while maintaining competitive performance.
  This hybrid architecture allows the model to learn both long-term context and recent patterns in medical symptom data.

Example:

- **Architecture:** Embedding → SpatialDropout → [LSTM branch, GRU branch] → Concatenate → Global Max Pooling → BatchNorm → Dense → Dropout → Output layer.

- **Input Representation:** Tokenized and padded sequences of symptoms (max length = 100).

- **Reasoning:** Sequential modeling is critical for medical symptoms where word order influences meaning (e.g., “sudden chest pain” vs. “chest pain sudden onset after exercise”).

- **Advantages:** Captures contextual meaning, handles variable sequence lengths, reduces information loss.

- **Use Case:** Accurate classification of diseases from longer, context-rich symptom descriptions.
- **Objective:** Learn from sequential dependencies and semantic nuances in symptom narratives for multi-class classification.

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

Model Architecture

1. **Input Layer:** Accepts tokenized symptom descriptions, each sequence padded to length = 100 tokens.

2. **Embedding Layer:** Maps tokens to 128-dimensional dense vectors, capturing semantic relationships between words.

3. **Spatial Dropout:** Randomly drops entire embedding dimensions to improve generalization.

4. **Parallel Recurrent Branches:**

    - **LSTM (64 units, return_sequences=True):** Captures long-term dependencies.

    - **GRU (64 units, return_sequences=True):** Efficiently learns short- and medium-term dependencies with fewer parameters.

5. **Concatenate:** Combines outputs of LSTM and GRU to form a richer representation.

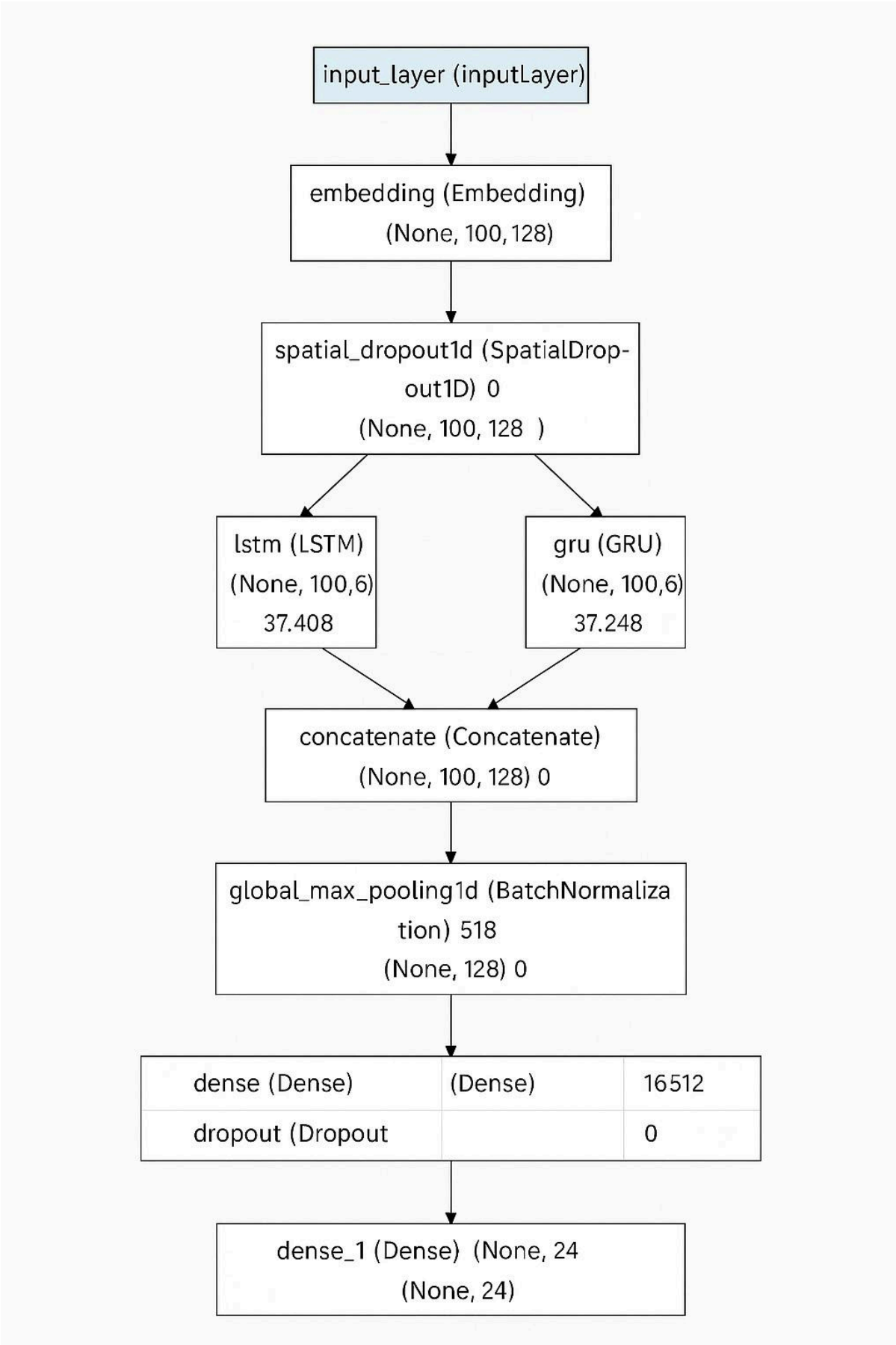
6. **Global Max Pooling:** Reduces sequence dimension, keeping the most salient features from each feature map.

7. **Batch Normalization:** Normalizes feature distribution for faster convergence.

8. **Dense Layer (128 units, ReLU):** Learns complex nonlinear relationships.

9. **Dropout (0.5):** Prevents overfitting by randomly dropping neurons during training.

10. **Output Layer (Dense, 24 units, Softmax):** Predicts probability distribution over 24 disease classes, these are fully connected layers with units = number of classes (24 in this case) with softmax activation.


input_layer (inputLayer)
embedding (Embedding)
(None, 100, 128)
spatial_dropout1d (SpatialDrop-
out1D) 0
(None, 100, 128)
lstm (LSTM)
(None, 100,6)
37.408
gru (GRU)
(None, 100,6)
37.248
concatenate (Concatenate)
(None, 100, 128) 0
global_max_pooling1d (BatchNormaliza
tion) 518
(None, 128) 0
dense (Dense)
(Dense)
16512
dropout (Dropout
0
dense_1 (Dense) (None, 24
(None, 24)

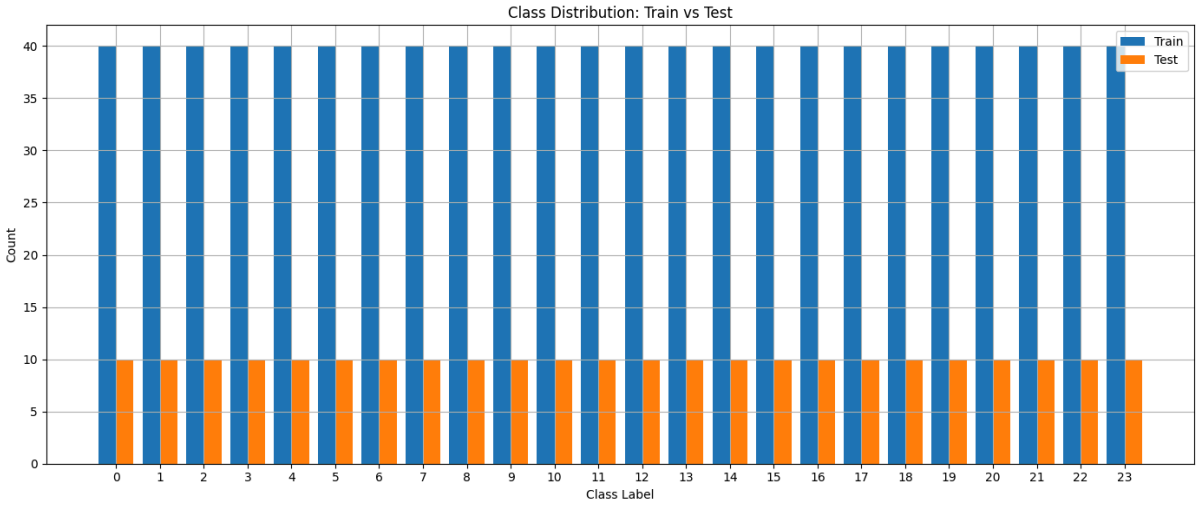
Data Preprocessing:

- **Label Encoding:** `LabelEncoder` from `sklearn.preprocessing` to map categorical labels to integers.
- **Vocabulary Size:** Derived from the dataset; embedding layer input_dim = vocab_size + 1.
- **Sequence Padding:** `pad_sequences` to fixed length = 100 tokens.

```
# Convert to sequences
max_len = 100
X = tokenizer.texts_to_sequences(symptom_df2['cleaned_text'])
X = pad_sequences(X, maxlen=max_len, padding='post', truncating='post')

# Label encoding
y = symptom_df2['label']
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_onehot = to_categorical(y_encoded)
num_classes = y_onehot.shape[1]
```

- **Train-Test Split:** 80–20 split, stratified by labels, `random_state=42`.

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, test_size=0.2, random_state=42, stratify=y_encoded)

# Save vocab_size and num_classes for later use
print(f"Vocabulary size: {vocab_size}")
print(f"Number of classes: {num_classes}")
print(f"Sample encoded text: {X_train[0]}")
print(f"Encoded label: {y_train[0]}")
```



- **Tokenization:** Keras `Tokenizer` with vocabulary size = *vocab_size* (from dataset)converts text into sequences of integer IDs.

```
# Tokenization
tokenizer = Tokenizer(oov_token='')
tokenizer.fit_on_texts(symptom_df2['cleaned_text'])
word_index = tokenizer.word_index
vocab_size = len(word_index) + 1
```

Model Configuration:

- **Embedding Layer:** Input dim = `vocab_size + 1`, output dim = 128.
- **Bidirectional LSTM:** 64 units, `return_sequences=True` for passing context to GRU.
- **GRU Layer:** 64 units, `return_sequences=False`.
- **Dense Layer:** 64 units, ReLU activation.
- **Output Layer:** Softmax activation, units = number of classes.

- **Loss Function:** `categorical_crossentropy` (multi-class classification).
- **Optimizer:** `Adam`, learning rate = 0.001.
- **Regularization:** Dropout (0.5 after LSTM, 0.3 after GRU).

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100)	0	-
embedding (Embedding)	(None, 100, 128)	202,752	input_layer[0][0]
spatial_dropout1d (SpatialDropout1D)	(None, 100, 128)	0	embedding[0][0]
lstm (LSTM)	(None, 100, 64)	49,408	spatial_dropout1…
gru (GRU)	(None, 100, 64)	37,248	spatial_dropout1…
concatenate (Concatenate)	(None, 100, 128)	0	lstm[0][0], gru[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 128)	0	concatenate[0][0]
batch_normalization (BatchNormalizatio…	(None, 128)	512	global_max_pooli…
dense (Dense)	(None, 128)	16,512	batch_normalizat…
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 24)	3,096	dropout[0][0]

Total params: 309,528 (1.18 MB)

Trainable params: 309,272 (1.18 MB)

Non-trainable params: 256 (1.00 KB)

Training Parameters:

- **Batch Size:** 32
- **Epochs:** 10
- **Callbacks:** EarlyStopping (patience=3, restore_best_weights=True) to avoid overfitting.

```
# Step 3: Training callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, verbose=1),
    ModelCheckpoint('best_model.h5', save_best_only=True, monitor='val_loss')
]
```

```
# Step 4: Train the model
history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    epochs=15,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)
```

Evaluation Metrics:

- **F1 macro Score:** The **F1 macro score** is a metric that calculates the F1 score independently for each class and then takes the unweighted average across all classes.
    - It evaluates how well the model is performing **across all classes**, not just the ones with more data.
    - The model is predicting **multiple disease categories based on symptom descriptions**, where the distribution of classes is not uniform.
    - A standard accuracy score might be misleading — the model could achieve high accuracy by focusing on majority classes and neglecting minority ones.

- **F1 macro score ensures balanced performance** by giving equal importance to both common and rare classes, making it a fairer reflection of the model's ability to generalize.

This means that a high F1 macro score here would indicate the model is not only making correct predictions overall but also **handling all classes consistently well**, which is crucial for reliable deployment.

Train and Test :-

```python
from sklearn.metrics import f1_score

# Predictions for training data
y_train_pred = model.predict(X_train)
y_train_pred_classes = np.argmax(y_train_pred, axis=1)
y_train_true_classes = np.argmax(y_train, axis=1)  # if y_train is one-hot

# Predictions for test data
y_test_pred = model.predict(X_test)
y_test_pred_classes = np.argmax(y_test_pred, axis=1)
y_test_true_classes = np.argmax(y_test, axis=1)  # if y_test is one-hot

# Calculate F1 macro scores
train_f1_macro = f1_score(y_train_true_classes, y_train_pred_classes, average='macro')
test_f1_macro = f1_score(y_test_true_classes, y_test_pred_classes, average='macro')

print(f"Train F1 Macro Score: {train_f1_macro:.4f}")
print(f"Test F1 Macro Score: {test_f1_macro:.4f}")

# Calculate F1 weighted scores
train_f1_weighted = f1_score(y_train_true_classes, y_train_pred_classes, average='weighted')
test_f1_weighted = f1_score(y_test_true_classes, y_test_pred_classes, average='weighted')

print(f"Train F1 weighted Score: {train_f1_weighted:.4f}")
print(f"Test F1 weighted Score: {test_f1_weighted:.4f}")
```

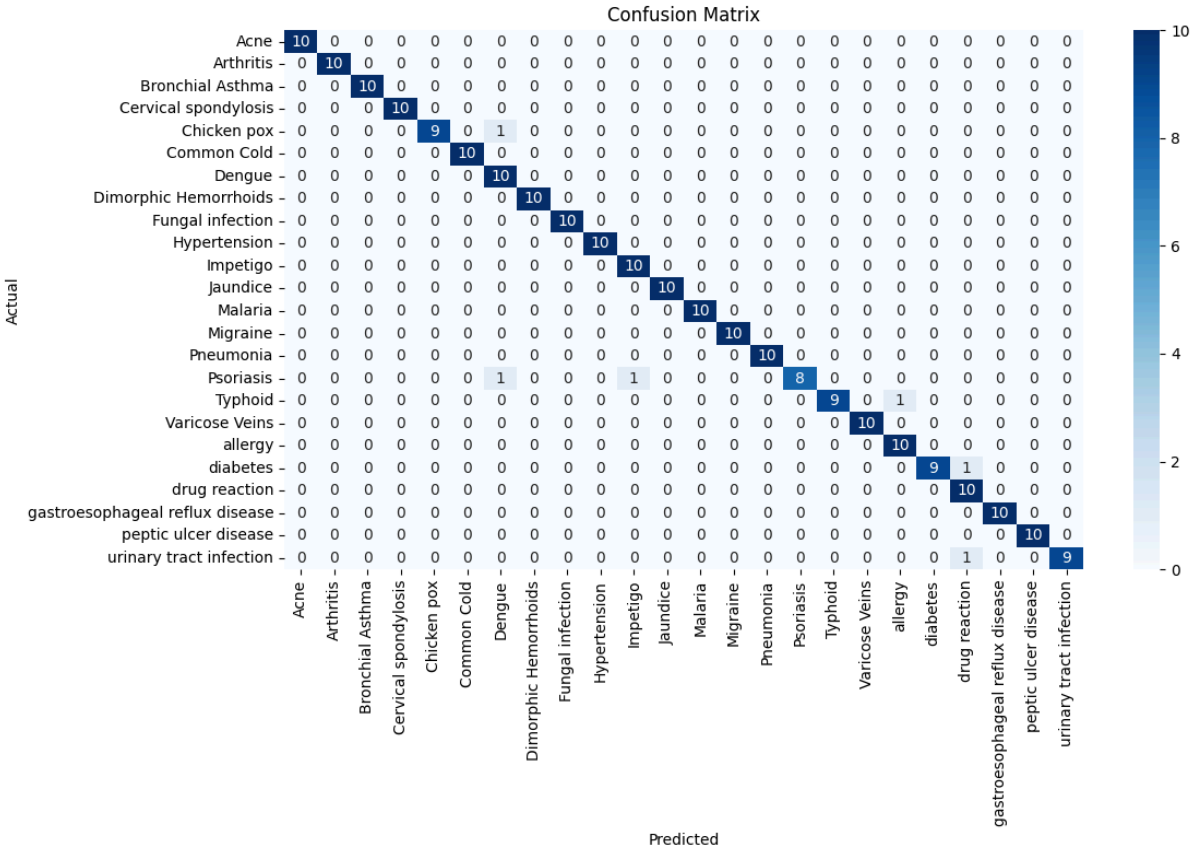
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
Train F1 Macro Score: 0.9979
Test F1 Macro Score: 0.9751
Train F1 weighted Score: 0.9979
Test F1 weighted Score: 0.9751
```

- **Classification Report:** Precision, Recall, F1-score per class for medical decision reliability.

Classification Report:

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	10
Arthritis	1.00	1.00	1.00	10
Bronchial Asthma	1.00	1.00	1.00	10
Cervical spondylosis	1.00	1.00	1.00	10
Chicken pox	1.00	0.90	0.95	10
Common Cold	1.00	1.00	1.00	10
Dengue	0.83	1.00	0.91	10
Dimorphic Hemorrhoids	1.00	1.00	1.00	10
Fungal infection	1.00	1.00	1.00	10
Hypertension	1.00	1.00	1.00	10
Impetigo	0.91	1.00	0.95	10
Jaundice	1.00	1.00	1.00	10
Malaria	1.00	1.00	1.00	10
Migraine	1.00	1.00	1.00	10
Pneumonia	1.00	1.00	1.00	10
Psoriasis	1.00	0.80	0.89	10
Typhoid	1.00	0.90	0.95	10
Varicose Veins	1.00	1.00	1.00	10
allergy	0.91	1.00	0.95	10
diabetes	1.00	0.90	0.95	10
drug reaction	0.83	1.00	0.91	10
gastroesophageal reflux disease	1.00	1.00	1.00	10
peptic ulcer disease	1.00	1.00	1.00	10
urinary tract infection	1.00	0.90	0.95	10
accuracy			0.97	240
macro avg	0.98	0.98	0.98	240
weighted avg	0.98	0.97	0.98	240

- **Confusion Matrix:** Visualized using seaborn heatmap to identify misclassifications.



3.5 Transformer based Models: pretrained BERT based sequence classification model

Description

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art language representation model developed by Google, designed to understand the context and semantics of words in a sentence by looking at both their left and right surroundings. Unlike traditional word embeddings such as Word2Vec or GloVe, which produce static word

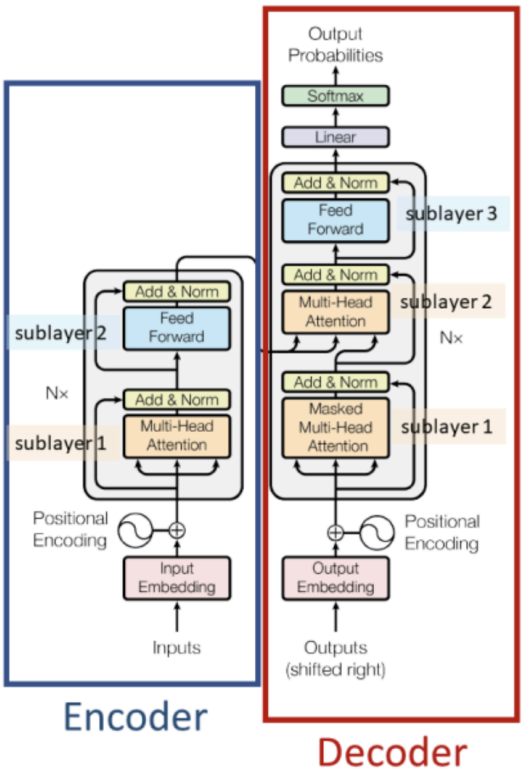
vectors, BERT generates contextualized embeddings, meaning the same word can have different representations depending on its usage in a sentence.

BERT is pre-trained on two key tasks:

i. Masked Language Modeling (MLM) – Predicting masked words in a sentence based on context.
Ii. Next Sentence Prediction (NSP) – Determining whether one sentence logically follows another.

Once pre-trained on massive text corpora, BERT can be fine-tuned for specific downstream tasks, such as multi-class text classification, by adding a classification layer on top of the BERT architecture.

BERT Architecture



It is a **Transformer** based model architecture, which opens a new era to work with NLP tasks. In the last session, we learned about the Transformer. A transformer is an Encoder-Decoder model architecture that also uses positional encoding, self-attention, multi-head attention, and also with Residual connection. Now, BERT uses only the Encoder portion of the Transformer architecture canceling out the Decoder part.Like the Transformer BERT also uses positional encoding, self attention, multi head attention and Residual Connection. It uses exactly the same architecture for encoders that is used in the Transformer. In short BERT is basically, Stacking of Encoders and the encoder is from the Transformer.

- **Overall Structure:**
  - Layers: BERT is a stack of multiple Transformer encoder layers (12 for BERT-Base, 24 for BERT-Large).
  - Hidden Size: Each encoder layer produces a vector of fixed size for each token (768 for Base, 1024 for Large).
  - Attention Heads: Each layer has multiple self-attention heads (12 for Base, 16 for Large) that allow the model to focus on different relationships in the text.
  - Parameters:
    - BERT-Base: 110M parameters
    - BERT-Large: 340M parameters

- **Input Representation :**
  Before entering the model, text is converted into token embeddings using:
  - Token Embeddings – Generated using WordPiece tokenization (breaks rare words into subwords).
  - Segment Embeddings – Distinguish sentence A and sentence B in tasks like Next Sentence Prediction.
  - Position Embeddings – Encode the position of each token in the sequence, since Transformers have no inherent sense of order.

These three embeddings are summed to produce the final input vector for each token.

- **Transformer Encoder Layers:**

  Each layer consists of two main sub-components:

a) Multi-Head Self-Attention - Allows the model to determine which words in a sentence are relevant to each other. Multiple heads learn different types of relationships (e.g., subject-verb, object-adjective).

b) Feed-Forward Neural Network - Each token's representation is passed through a fully connected feed-forward network. Includes layer normalization and residual connections for stability and better gradient flow.

- **Special Tokens:**
  - **[CLS]**: Placed at the start of the sequence; its final hidden state is used for classification tasks.
  - **[SEP]**: Marks separation between sentences or end of a sequence.

- **Output Layer (Fine-Tuning Stage):**

For multi-class classification:

  - The final hidden state of the [CLS] token is passed to a dense layer.
  - A softmax activation outputs the probability distribution across all classes.

Data Preprocessing

- **Tokenization (WordPiece)**

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
text = "I love cats."
tokens = tokenizer.tokenize(text)
print(tokens)  # ['i', 'love', 'cats', '.']
```

What happens here:

- Uses **WordPiece** to split into subwords.
- Everything is lowercased (`uncased` model).

Special Tokens (`[CLS]`, `[SEP]`) + IDs

```
encoded = tokenizer.encode(text, add_special_tokens=True)
print(encoded)  # [101, 1045, 2293, 8874, 1012, 102]
```

- `101` = `[CLS]`
- `102` = `[SEP]`
- The rest are token IDs from BERT's vocabulary.

3. Padding & Truncation
python
CopyEdit

```
batch = tokenizer(
    ["I love cats.", "They are cute animals."],
    max_length=8,
    padding="max_length",
    truncation=True,
    return_tensors="pt"
)
```

This generates:

- **`input_ids`** → token IDs (with `[PAD]` if needed)
- **`attention_mask`** → `1` for real tokens, `0` for `[PAD]`
- **`token_type_ids`** → segment IDs (all zeros for single sentences)

Output Example

```
print(batch)

input_ids:
[[101, 1045, 2293, 8874, 1012, 102, 0, 0],
 [101, 2027, 2024, 10140, 6611, 1012, 102, 0]]

attention_mask:
[[1, 1, 1, 1, 1, 1, 0, 0],
 [1, 1, 1, 1, 1, 1, 1, 0]]

token_type_ids:
[[0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

This is exactly what BERT expects as input

`input_ids` → WordPiece token IDs with `[CLS]` & `[SEP]`
`attention_mask` → Which tokens to attend to
`token_type_ids` → Sentence segment markers

Model Development

Dataset Preparation

A custom `SymptomDataset` class is implemented to manage tokenized inputs and labels for PyTorch training. This class inherits from `torch.utils.data.Dataset` and allows efficient data loading during training and evaluation.

```
class SymptomDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
```

```
        return {key: torch.tensor(val[idx]) for key, val in
self.encodings.items()} | {"labels": torch.tensor(self.labels[idx])}
```

Model Selection

The model is based on **BertForSequenceClassification** with `bert-base-uncased` as the backbone. The classifier head is configured for **24 output labels** corresponding to the dataset's categories.

```
model =
BertForSequenceClassification.from_pretrained("bert-base-uncased",
num_labels=24)
```

Training Configuration

Training is managed via Hugging Face's **Trainer API** with the following parameters:

- **Batch Size:** 8 (for both training and evaluation)
- **Epochs:** 3
- **Checkpointing:** Model saved every 1000 steps (keeping only the latest checkpoint)
- **Logging:** Evaluation every epoch with logs recorded every 50 steps

```
training_args = TrainingArguments(
    output_dir="./results",
    save_steps=1000,
    save_total_limit=1,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    logging_dir="./logs",
    logging_steps=50,
    do_eval=True
)
```

Evaluation and Reporting

After training, the model is evaluated on the test dataset. Predictions are generated, and a **classification report** is created using `sklearn.metrics.classification_report` to measure **precision, recall, and F1-score** for each class.

```
predictions = trainer.predict(test_dataset)
pred_labels = predictions.predictions.argmax(axis=1)
print(classification_report(test_labels, pred_labels,
target_names=label_encoder.classes_))
```

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	7
Arthritis	1.00	1.00	1.00	7
Bronchial Asthma	1.00	1.00	1.00	7
Cervical spondylosis	1.00	1.00	1.00	8
Chicken pox	0.89	1.00	0.94	8
Common Cold	1.00	1.00	1.00	7
Dengue	1.00	0.88	0.93	8
Dimorphic Hemorrhoids	1.00	1.00	1.00	8
Fungal infection	1.00	1.00	1.00	7
Hypertension	1.00	1.00	1.00	7
Impetigo	1.00	1.00	1.00	8
Jaundice	1.00	1.00	1.00	7
Malaria	1.00	1.00	1.00	7
Migraine	1.00	1.00	1.00	7
Pneumonia	1.00	1.00	1.00	8
Psoriasis	1.00	1.00	1.00	8
Typhoid	1.00	1.00	1.00	7
Varicose Veins	1.00	1.00	1.00	8
allergy	1.00	1.00	1.00	7
diabetes	1.00	1.00	1.00	8
drug reaction	1.00	1.00	1.00	8
gastroesophageal reflux disease	1.00	1.00	1.00	7
peptic ulcer disease	1.00	1.00	1.00	8
urinary tract infection	1.00	1.00	1.00	8
accuracy			0.99	180
macro avg	1.00	0.99	0.99	180
weighted avg	1.00	0.99	0.99	180

Sample output :

```
# Example
print(predict_disease("high fever, severe headache, joint pain"))
```

```
Dengue
```

F1 macro Score: The **F1 macro score** is a metric that calculates the F1 score independently for each class and then takes the unweighted average across all classes.

- It evaluates how well the model is performing **across all classes**, not just the ones with more data.
- The model is predicting **multiple disease categories based on symptom descriptions**, where the distribution of classes is not uniform.
- A standard accuracy score might be misleading — the model could achieve high accuracy by focusing on majority classes and neglecting minority ones.
- **F1 macro score ensures balanced performance** by giving equal importance to both common and rare classes, making it a fairer reflection of the model's ability to generalize.

This means that a high F1 macro score here would indicate the model is not only making correct predictions overall but also **handling all classes consistently well**, which is crucial for reliable deployment.

```
from sklearn.metrics import f1_score

# Macro F1 = average F1 across all classes (treats all classes equally)
macro_f1 = f1_score(test_labels, pred_labels, average='macro')

# Weighted F1 = average F1 weighted by number of samples in each class
weighted_f1 = f1_score(test_labels, pred_labels, average='weighted')

print(f"Macro F1-score: {macro_f1:.4f}")
print(f"Weighted F1-score: {weighted_f1:.4f}")
```

```
Macro F1-score: 0.9948
Weighted F1-score: 0.9944
```

4. Model Comparison and Conclusion

i. Model Process

Step / Feature	Logistic Regression + TF-IDF	LSTM / GRU (Deep NN)	BERT Classifier
Text Representation	Bag-of-words weighted by TF-IDF. Loses order information. Sparse vector representation.	Word embeddings (e.g., Word2Vec, GloVe) or trainable embedding layer. Preserves sequential order of words. Dense, continuous vector representation.	Contextual embeddings via Transformer encoder. Each token's meaning changes based on context in the sentence.
Feature Extraction	Manual feature engineering via n-grams. No learning of relationships between words.	Learns sequential dependencies via recurrent connections (LSTM/GRU cells). Can capture long-term dependencies but still limited for very long texts.	Self-attention captures global context across the entire sequence simultaneously. Handles very long dependencies better than RNNs.
Preprocessing	Lowercasing, tokenization, stopword removal, TF-IDF vectorization.	Tokenization, optional stopword handling, mapping to embeddings, padding/truncation.	WordPiece tokenization, adding [CLS] & [SEP], attention masks, token type IDs. Minimal manual preprocessing.
Training Complexity	Very fast to train, low compute needs.	Moderate compute needs; sequential nature means slower training compared to TF-IDF.	High compute demand; large number of parameters; GPU recommended.
Model Size	Very small (few MBs).	Medium (depends on embedding + recurrent layers).	Large (~420MB for bert-base-uncased).
Interpretability	High — coefficients can be inspected.	Moderate — some understanding from feature visualization but less transparent.	Low — highly complex attention patterns; explainability tools needed.

ii. Performance Comparison

Metric	Logistic Regression + TF-IDF	LSTM / GRU	BERT
Avg F1 Macro Score	0.944	0.975	0.989
Error Reduction (vs TF-IDF)	—	~66% fewer errors	~90% fewer errors
Strengths	Simple, fast, interpretable. Good for smaller datasets.	Captures sequence order & moderate context. Strong for structured text patterns.	Captures deep semantic meaning & global context. Strongest for nuanced language.
Weaknesses	Cannot understand context or word order.	Struggles with very long dependencies; training is slower.	Heavy computational cost; requires large memory and inference time.

iii. Conclusion

1. **Logistic Regression + TF-IDF**
    - Best when **speed, simplicity, and interpretability** are most important.
    - Performance is strong but capped because it ignores word order and context.
2. **LSTM / GRU**
    - Significant improvement because it models **word order and sequential patterns**.
    - Still limited compared to Transformer models for long-range dependencies.
    - Good trade-off for moderately complex NLP tasks when BERT-level compute is unavailable.

3. **BERT Classifier**

    - **Highest F1 (0.989)** due to contextual embeddings and self-attention capturing both local and global dependencies.

    - Excels at nuanced multi-class classification where **subtle word meaning shifts** matter.

    - Downside: **compute heavy**, larger latency, and harder to interpret.

Final Verdict:

For production, the choice depends on constraints:

- If **accuracy is critical** and compute resources allow, **BERT** is the clear winner.

- If **moderate compute** is available and you still need strong performance, **LSTM/GRU** is a solid middle ground.

- If **speed, low resource usage, and interpretability** matter most, **TF-IDF + Logistic Regression** is still very effective.

APPENDIX

A.1 Data Preparation and Analysis

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import pickle
from wordcloud import WordCloud
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
```

Output: Import of blow libraries

pandas – Data manipulation and cleaning

numpy – Numerical computations

matplotlib – Plotting graphs and visualizations

seaborn – Statistical data visualization

scikit-learn – Data splitting and preprocessing (train_test_split, LabelEncoder)

TensorFlow / Keras – Deep learning framework for building and training LSTM models

pickle – Object serialization for saving models/tokenizers

wordcloud – Generating visual word cloud representations

nltk – Natural Language Toolkit for text preprocessing and stopword removal

A.2 Load the dataset:

```
## Step 2: Load and Prepa
symptom_df = pd.read_csv(
```

✓ 0.2s

```
symptom_df.head()
```

Output:

	Unnamed: 0	label	text
0	0	Psoriasis	I have been experiencing a skin rash on my arm...
1	1	Psoriasis	My skin has been peeling, especially on my kne...
2	2	Psoriasis	I have been experiencing joint pain in my fing...
3	3	Psoriasis	There is a silver like dusting on my skin, esp...
4	4	Psoriasis	My nails have small dents or pits in them, and...

A.2 Drop column “Unnamed” and rename “text” column as “label”:

Code:

```
symptom_df = symptom_df.drop(columns=["Unnamed: 0"])
symptom_df.columns = ["label", "text"]
```

✓ 0.0s

Output:

	label	
0	Psoriasis	I have been experiencing a skin rash on my a
1	Psoriasis	My skin has been peeling, especially on my k
2	Psoriasis	I have been experiencing joint pain in my f
3	Psoriasis	There is a silver like dusting on my skin,
4	Psoriasis	My nails have small dents or pits in them, a

A.3: Encode Labels:

Code:

```
# Encode labels
label_encoder = LabelEncoder()
symptom_df["encoded_label"] = label_encoder.fit_transform(symptom_df["label"])
```

Output:

	label	text	encoded_label
0	Psoriasis	I have been experiencing a skin rash on my arm...	15
1	Psoriasis	My skin has been peeling, especially on my kne...	15
2	Psoriasis	I have been experiencing joint pain in my fing...	15
3	Psoriasis	There is a silver like dusting on my skin, esp...	15
4	Psoriasis	My nails have small dents or pits in them, and...	15

A.3: EDA (Folder creation for visualization):

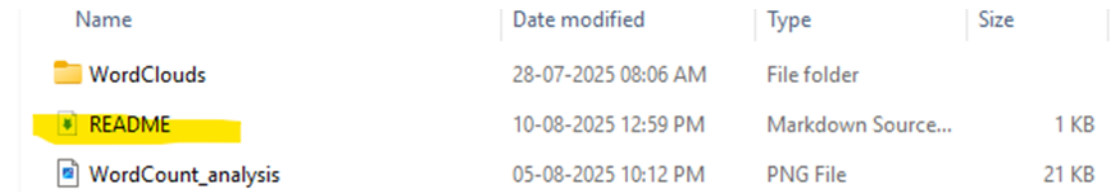
Code:

```python
import os

# Define base path to your working branch directory
base_path = "D:\Masters\M2_AAI-501 Introduction to Artificial Intelligence\Group Project\Symptom2Disease"

# Create visualization folders
os.makedirs(os.path.join(base_path, "Visualizations/WordClouds/PerLabel"), exist_ok=True)

# Optional: Create README file
readme_path = os.path.join(base_path, "Visualizations", "README.md")
with open(readme_path, "w") as f:
    f.write("This folder contains visualizations from symptom-level data, including word clouds.\n")

print("Folder structure created successfully.")
```

✓ 0.0s

Output:

```
Folder structure created successfully.
```



Name	Date modified	Type	Size
WordClouds	28-07-2025 08:06 AM	File folder	
README	10-08-2025 12:59 PM	Markdown Source...	1 KB
WordCount_analysis	05-08-2025 10:12 PM	PNG File	21 KB

B.1: EDA (Class distribution):

Code:

```python
symptom_df['label'].unique()
```

Output:

```
array(['Psoriasis', 'Varicose Veins', 'Typhoid', 'Chicken pox',
       'Impetigo', 'Dengue', 'Fungal infection', 'Common Cold',
       'Pneumonia', 'Dimorphic Hemorrhoids', 'Arthritis', 'Acne',
       'Bronchial Asthma', 'Hypertension', 'Migraine',
       'Cervical spondylosis', 'Jaundice', 'Malaria',
       'urinary tract infection', 'allergy',
       'gastroesophageal reflux disease', 'drug reaction',
       'peptic ulcer disease', 'diabetes'], dtype=object)
```

Code:

```
symptom_df['label'].nunique()
```

Output:

```
24
```

Code:

```
print("shape of the dataset", symptom_df.shape)
print(symptom_df['label'].value_counts())
```

Output:

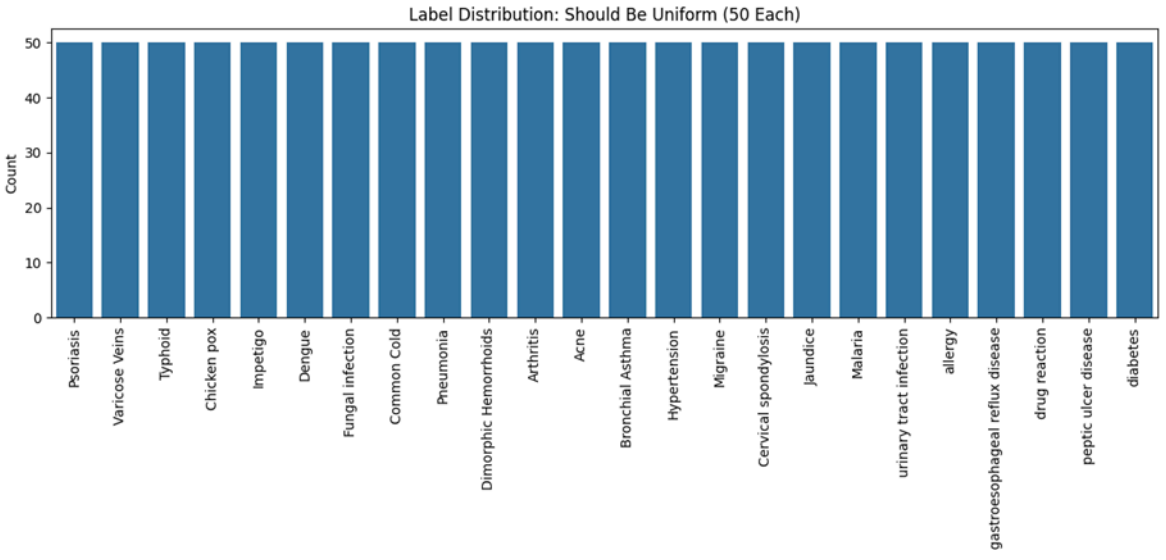
```
shape of the dataset (1200, 3)
label
Psoriasis                          50
Varicose Veins                     50
Typhoid                            50
Chicken pox                        50
Impetigo                           50
Dengue                             50
Fungal infection                   50
Common Cold                        50
Pneumonia                          50
Dimorphic Hemorrhoids              50
Arthritis                          50
Acne                               50
Bronchial Asthma                   50
Hypertension                       50
Migraine                           50
Cervical spondylosis               50
Jaundice                           50
Malaria                            50
urinary tract infection            50
allergy                            50
gastroesophageal reflux disease    50
drug reaction                      50
peptic ulcer disease               50
diabetes                           50
Name: count, dtype: int64
```

B.2: EDA (Class Balance):

Code:

```
label_counts = symptom_df['label'].value_counts()
print(label_counts)

# Plotting for visual confirmation
plt.figure(figsize=(12,6))
sns.barplot(x=label_counts.index, y=label_counts.values)
plt.xticks(rotation=90)
plt.title("Label Distribution: Should Be Uniform (50 Each)"
plt.ylabel("Count")
plt.xlabel("Disease Label")
plt.tight_layout()
plt.show()
```

Output:

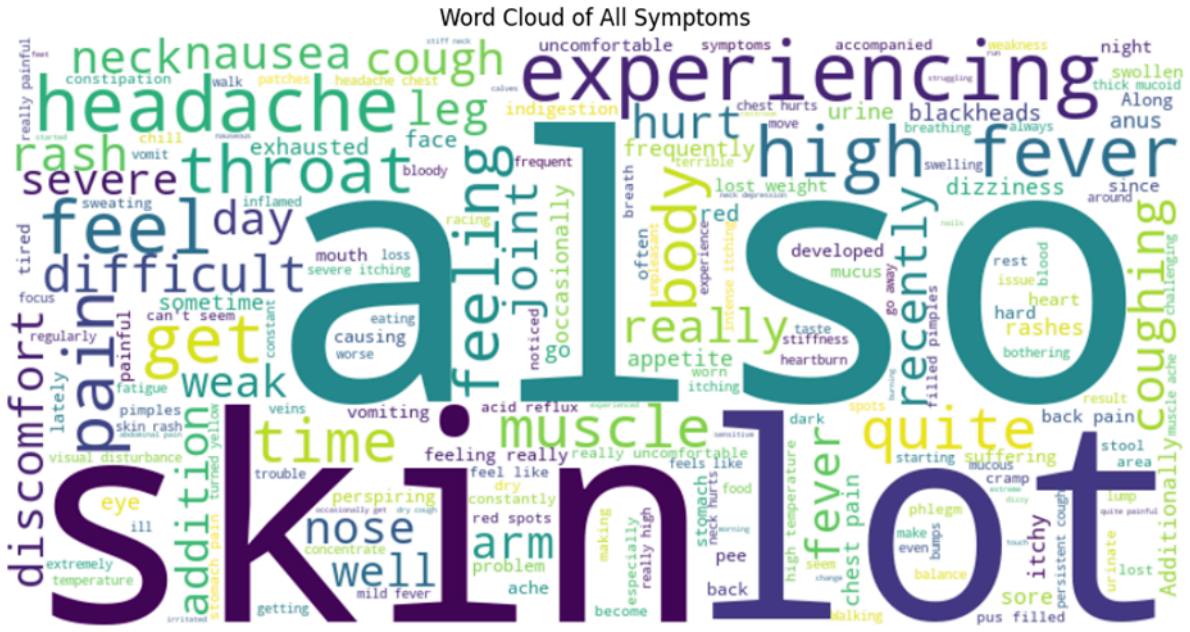


B.3: EDA Word Cloud of all symptom:

Code:

```
# Create folders if they don't exist
import os
combined_wc_path = os.path.join(base_path, "Visualizations/WordClouds/Combined.png")

all_text = ' '.join(symptom_df['text'].values)
stop_words = set(stopwords.words('english'))
wordcloud = WordCloud(width=800, height=400, background_color='white', stopwords=stop_words).generate(all_text)
plt.figure(figsize=(10,5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of All Symptoms")
plt.tight_layout()
plt.savefig(combined_wc_path)
plt.show()
```

Output:



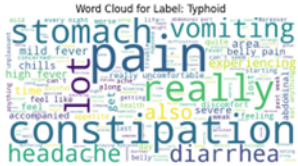
B.4: Word Cloud Individual Symptoms:

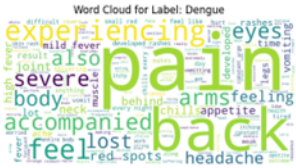
Code:

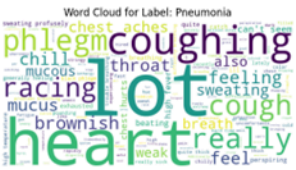
```
per_label_path = os.path.join(base_path,"Visualizations","WordClouds","PerLabel")


# Word clouds per label
from math import ceil
unique_labels = symptom_df['label'].unique()
num_labels = len(unique_labels)
cols = 4
rows = ceil(num_labels / cols)

fig, axes = plt.subplots(rows, cols, figsize=(20, 5 * rows))
axes = axes.flatten()

for i, label in enumerate(unique_labels):
    subset = symptom_df[symptom_df['label'] == label]
    text = ' '.join(subset['text'].values)
    wc = WordCloud(width=800, height=400, background_color='white', stopwords=stop_words).generate(text)
    axes[i].imshow(wc, interpolation='bilinear')
    axes[i].axis('off')
    axes[i].set_title(f"Word Cloud for Label: {label}")

    # Save word cloud to the correct path
    label_filename = f"{label}.png"
    wc.to_file(os.path.join(per_label_path, label_filename))

# Hide empty subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

plt.tight_layout()
plt.show()
```

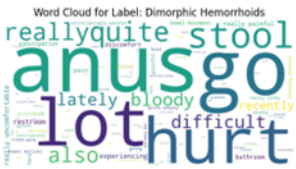
Output:

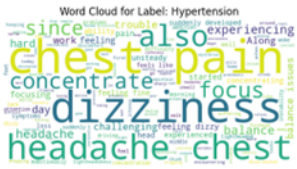
University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

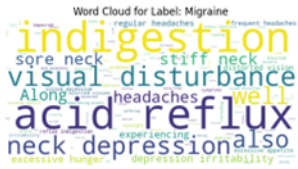


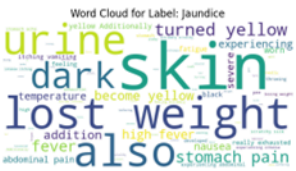


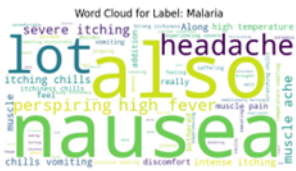


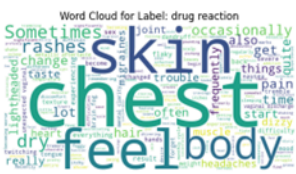










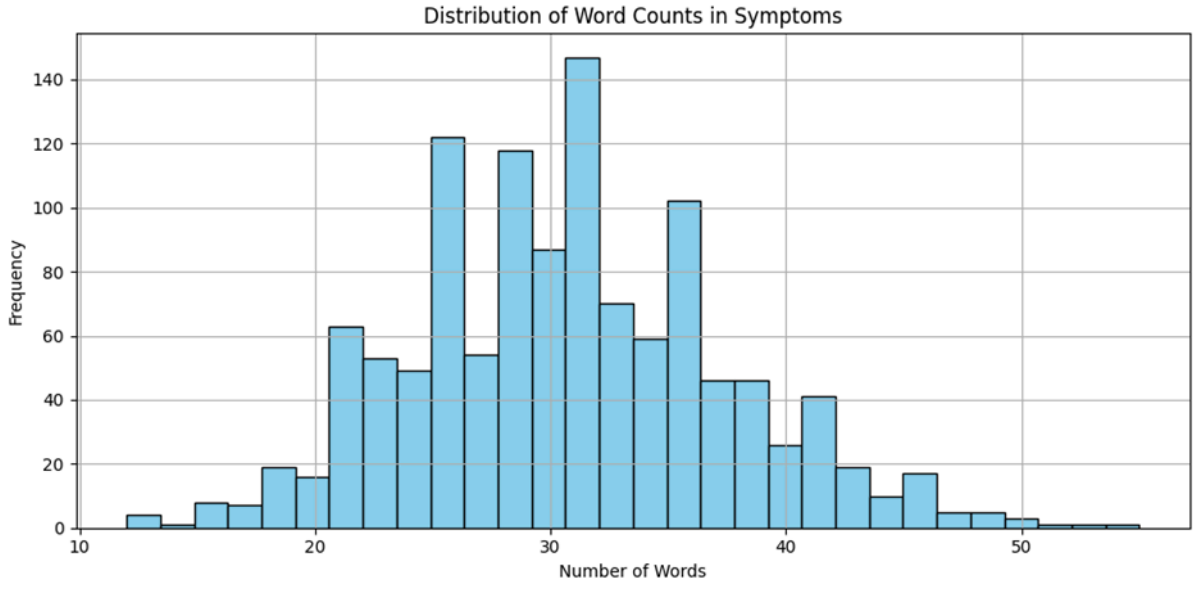


B.5: Word Count analysis:

Code:

```
# Word Count Analysis
symptom_df['word_count'] = symptom_df['text'].apply(lambda x: len(x.split()))
fig = plt.figure(figsize=(10, 5))
plt.hist(symptom_df['word_count'], bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Word Counts in Symptoms')
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.grid(True)

# Save to WordCount_analysis.png in correct path
wc_analysis_path = os.path.join(base_path, "Visualizations", "WordCount_analysis.png")
# os.makedirs(os.path.dirname(wc_analysis_path), exist_ok=True)
save_and_show_plot(fig, wc_analysis_path)
```

Output:



C.1 Modelling approach 1 (Linear regression: Preprocessing)

Code:

```
#Preprocessing
le=LabelEncoder()
symptom_df['label_encoded']=le.fit_transform(symptom_df['label'])

#Split training data
X_train, X_test, y_train, y_test = train_test_split(
    symptom_df['text'], symptom_df['label_encoded'], test_size=0.2, random_state=42, stratify=symptom_df['label_encoded'])

# TF-IDF Vectorization
tfidf = TfidfVectorizer(stop_words='english', max_features=500)
X_train_vec = tfidf.fit_transform(X_train)
X_test_vec = tfidf.transform(X_test)
```

✓ 0.0s

Output:

Label Encoding – Converts categorical disease labels into integers for compatibility with the Logistic Regression classifier.

Train-Test Split – Ensures fair evaluation by separating unseen data while maintaining label proportions using stratify.

TF-IDF Vectorization – Represents text as weighted word vectors, giving higher scores to rare but important words and discarding uninformative stop words.

C.2: Modelling

Code:

```
# Logistic regression model training

log_reg = LogisticRegression(max_iter=100)
log_reg.fit(X_train_vec, y_train)

# Predict
y_pred = log_reg.predict(X_test_vec)
```

Output:

Model Initialization – LogisticRegression(max_iter=100) sets the maximum optimization iterations to ensure convergence during training.

Model Training – The .fit() method learns the relationship between TF-IDF features and encoded disease labels.

Prediction – The .predict() method uses the trained model to classify unseen test symptom descriptions into their predicted disease labels.

C.3: Model Evaluation

Code:

```
# Evaluation
# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output:

```
Accuracy: 0.9375
```

Accurary with Linear regression model for Symptoms to Disease of 0.9375 is good startig metric. But class-wise performance also should be examined using classification report.

C.4: Classification Report

Code: Classification report for test data

```
# Classification report Test data
print("\nClassification Report Test data:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))
```

0.0s

Output:

```
Classification Report Test data:

                              precision    recall  f1-score   support

                        Acne       1.00      1.00      1.00        10
                   Arthritis       0.91      1.00      0.95        10
            Bronchial Asthma       0.91      1.00      0.95        10
          Cervical spondylosis       1.00      1.00      1.00        10
                   Chicken pox       0.78      0.70      0.74        10
                   Common Cold       1.00      1.00      1.00        10
                        Dengue       0.80      0.80      0.80        10
          Dimorphic Hemorrhoids       1.00      1.00      1.00        10
               Fungal infection       0.91      1.00      0.95        10
                   Hypertension       1.00      1.00      1.00        10
                       Impetigo       1.00      1.00      1.00        10
                       Jaundice       1.00      1.00      1.00        10
                        Malaria       1.00      1.00      1.00        10
                       Migraine       1.00      0.90      0.95        10
                      Pneumonia       0.83      1.00      0.91        10
                      Psoriasis       1.00      0.90      0.95        10
                        Typhoid       0.91      1.00      0.95        10
                Varicose Veins       1.00      1.00      1.00        10
                       allergy       1.00      0.70      0.82        10
                      diabetes       0.83      1.00      0.91        10
...
                      accuracy                           0.94       240
                     macro avg       0.94      0.94      0.94       240
                  weighted avg       0.94      0.94      0.94       240
```

Code: Classification report for train data

Classification Report Train data:

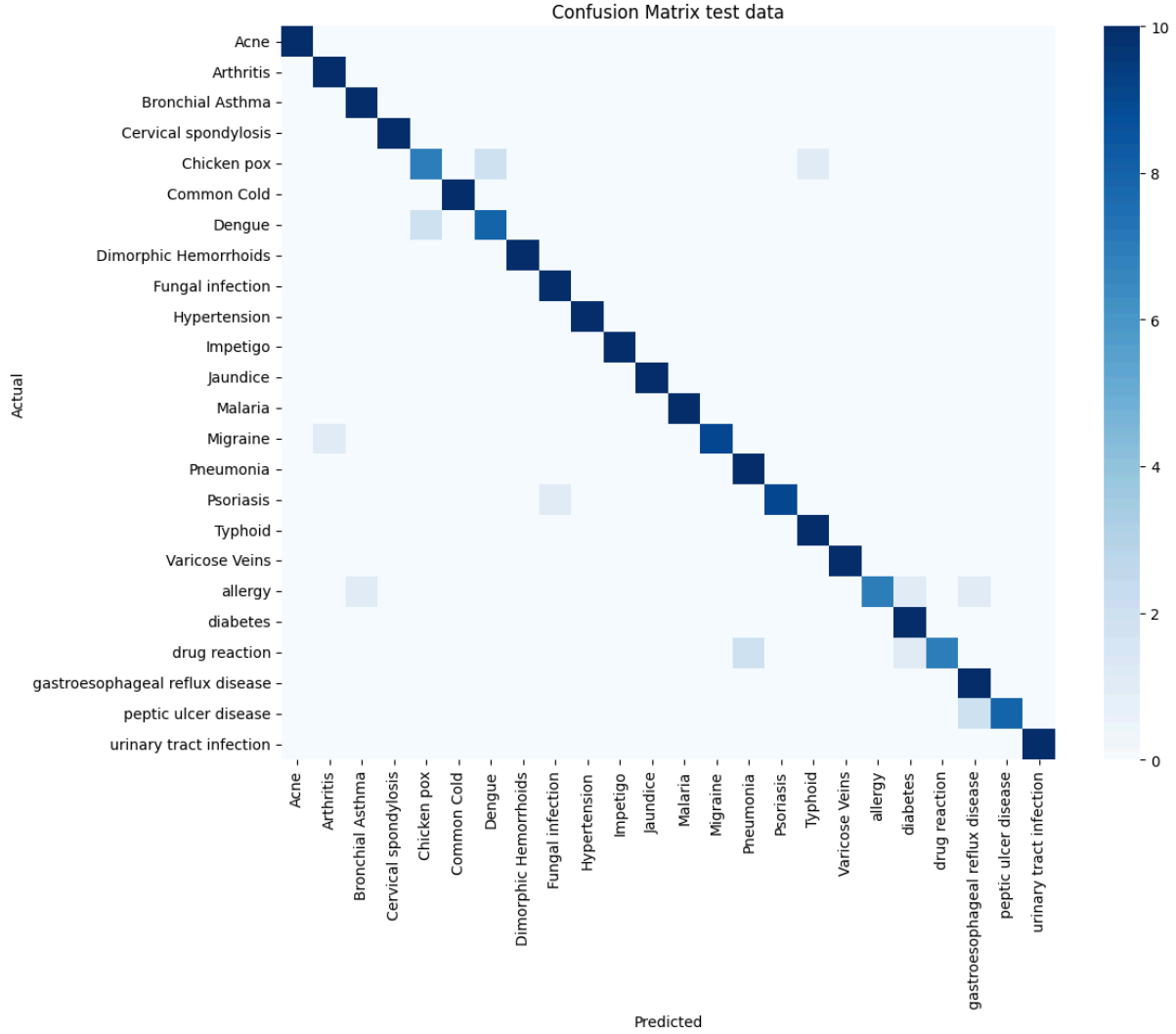
	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	40
Arthritis	1.00	1.00	1.00	40
Bronchial Asthma	1.00	1.00	1.00	40
Cervical spondylosis	1.00	1.00	1.00	40
Chicken pox	1.00	1.00	1.00	40
Common Cold	1.00	1.00	1.00	40
Dengue	1.00	1.00	1.00	40
Dimorphic Hemorrhoids	1.00	1.00	1.00	40
Fungal infection	1.00	1.00	1.00	40
Hypertension	1.00	1.00	1.00	40
Impetigo	1.00	1.00	1.00	40
Jaundice	1.00	1.00	1.00	40
Malaria	1.00	1.00	1.00	40
Migraine	1.00	1.00	1.00	40
Pneumonia	1.00	1.00	1.00	40
Psoriasis	1.00	0.97	0.99	40
Typhoid	1.00	1.00	1.00	40
Varicose Veins	1.00	1.00	1.00	40
allergy	1.00	1.00	1.00	40
diabetes	1.00	0.97	0.99	40
...				
accuracy			1.00	960
macro avg	1.00	1.00	1.00	960
weighted avg	1.00	1.00	1.00	960

C.5: Confusion matrix:

Code: Confusion matrix for test data

```
# Confusion matrix fTest data
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=False, cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix test data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

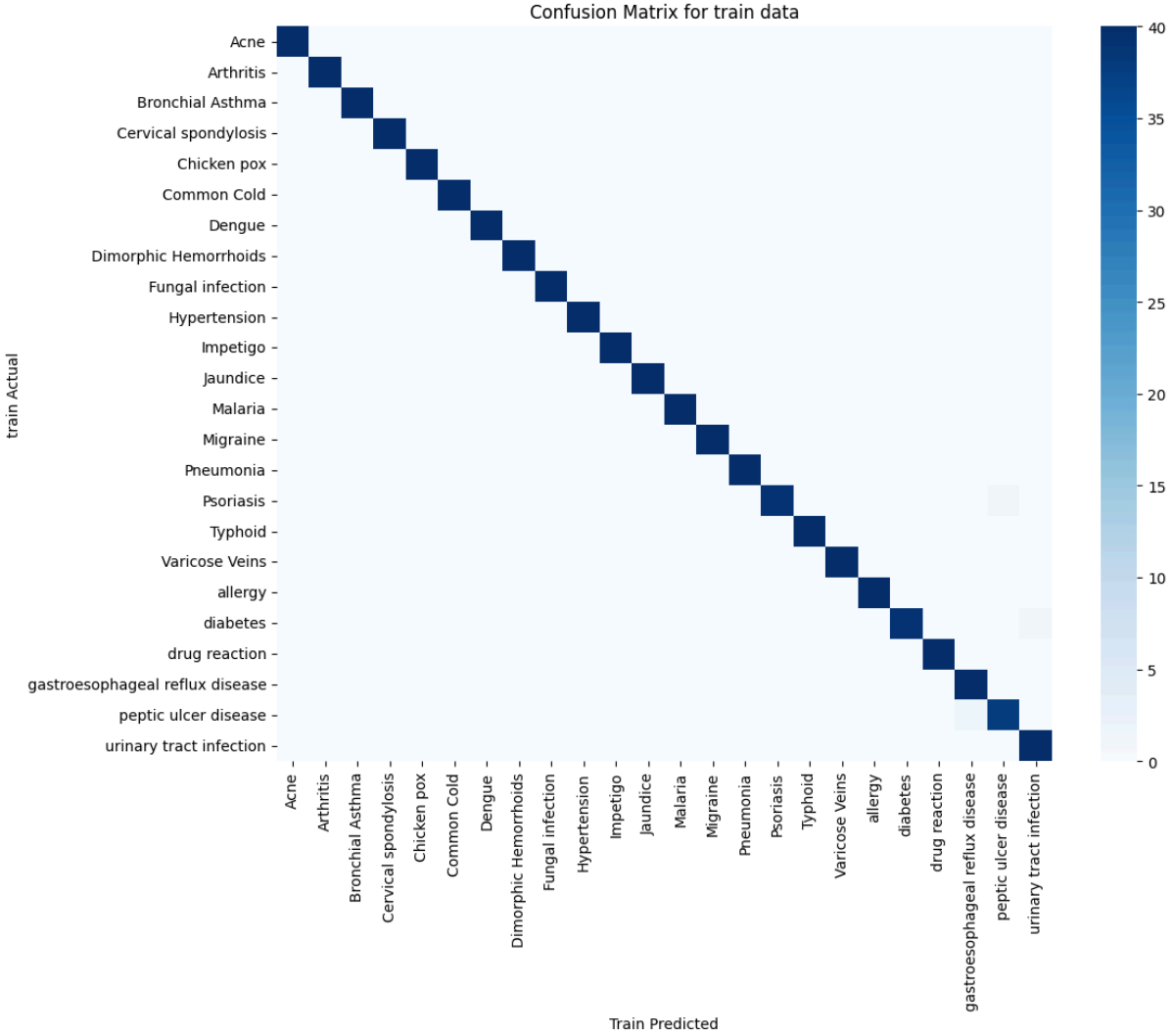
Output:



Code: Confusion matrix for train data

```
# Confusion matrix train data
cm_train = confusion_matrix(y_train, y_train_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(cm_train, annot=False, cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix for train data")
plt.xlabel("Train Predicted")
plt.ylabel("train Actual")
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

Output:



D.1: Modelling approach 2 : Deep Learning Models: LSTM + GRU Hybrid

Read the Data :

```
# Load your dataset
symptom_df2 = pd.read_csv("/content/USD-MS-AAI/Module 2/Data/Train_data.csv")

symptom_df2.head()
```

	Unnamed: 0	label	text
0	0	Psoriasis	I have been experiencing a skin rash on my arm...
1	1	Psoriasis	My skin has been peeling, especially on my kne...
2	2	Psoriasis	I have been experiencing joint pain in my fing...
3	3	Psoriasis	There is a silver like dusting on my skin, esp...
4	4	Psoriasis	My nails have small dents or pits in them, and...

D.2: Data Cleanup

Code:

4. Data Clean up

]:
```
def clean_text(text):
    text = text.lower()
    text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)
    return text.strip()
```

]:
```
# Apply cleaning - replace 'text_column' with the actual text feature name
symptom_df2['cleaned_text'] = symptom_df2['text'].apply(clean_text)
```

D.3: Tokenisation

Code:

```
# Tokenization
tokenizer = Tokenizer(oov_token='')
tokenizer.fit_on_texts(symptom_df2['cleaned_text'])
word_index = tokenizer.word_index
vocab_size = len(word_index) + 1
```

```
# Convert to sequences
max_len = 100
X = tokenizer.texts_to_sequences(symptom_df2['cleaned_text'])
X = pad_sequences(X, maxlen=max_len, padding='post', truncating='post')
```

D.4 : Label Encoding

Code:

```
# Label encoding
y = symptom_df2['label']
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_onehot = to_categorical(y_encoded)
num_classes = y_onehot.shape[1]
```

D.5 : Stratified Train Test Split

Code:

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, test_size=0.2, random_state=42, stratify=y_encoded)

# Save vocab_size and num_classes for later use
print(f"Vocabulary size: {vocab_size}")
print(f"Number of classes: {num_classes}")
print(f"Sample encoded text: {X_train[0]}")
print(f"Encoded label: {y_train[0]}")
```

Output:

```
Vocabulary size: 1584
Number of classes: 24
Sample encoded text: [ 23  29  19 291   2 321  23 201 502 269  21 735 182   8 381  91  22   3
  93   2  79   4   7  68   3  83   2  28  80 232 183   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
Encoded label: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

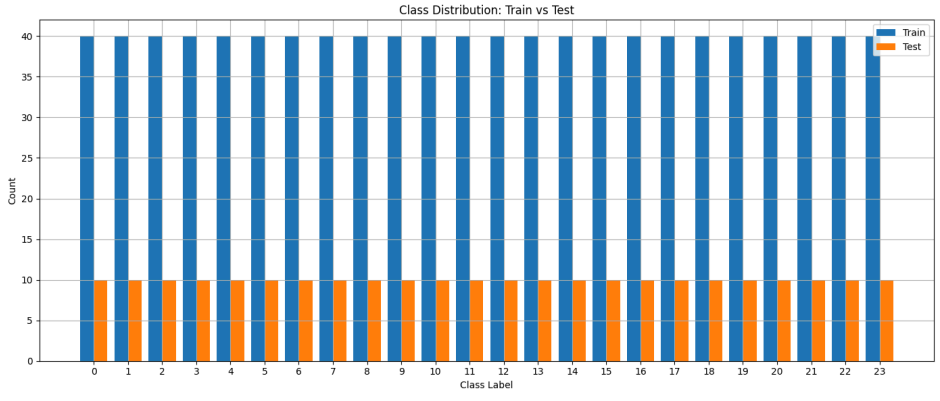
D.6: Validating Class distribution : Train vs Test

Code:

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

# Convert one-hot back to integer labels
y_train_labels = np.argmax(y_train, axis=1)
y_test_labels = np.argmax(y_test, axis=1)

# Count class distribution
train_counts = Counter(y_train_labels)
test_counts = Counter(y_test_labels)

# Sort by class index
train_counts = dict(sorted(train_counts.items()))
test_counts = dict(sorted(test_counts.items()))

# Print distribution
print("Train class distribution:")
for label, count in train_counts.items():
    print(f"Class {label}: {count}")

print("\nTest class distribution:")
for label, count in test_counts.items():
    print(f"Class {label}: {count}")

# Plot the distributions
plt.figure(figsize=(14, 6))
width = 0.4
classes = list(train_counts.keys())
train_vals = list(train_counts.values())
test_vals = list(test_counts.values())

x = np.arange(len(classes))
plt.bar(x - width/2, train_vals, width, label='Train')
plt.bar(x + width/2, test_vals, width, label='Test')
plt.xlabel('Class Label')
plt.ylabel('Count')
plt.title('Class Distribution: Train vs Test')
plt.xticks(ticks=x, labels=classes)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output :



University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

D.7: Model Development - LSTM: GRU Hybrid

Code:

7. Hybrid model

```
#Step 2: Build LSTM-GRU Hybrid Model
embedding_dim = 128

inputs = Input(shape=(max_len,))
x = Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len)(inputs)
x = SpatialDropout1D(0.3)(x)

# LSTM and GRU branches
lstm_out = LSTM(64, return_sequences=True)(x)
gru_out = GRU(64, return_sequences=True)(x)

# Concatenate both branches
x = Concatenate()([lstm_out, gru_out])
x = GlobalMaxPooling1D()(x)
x = BatchNormalization()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(num_classes, activation='softmax')(x)

model = Model(inputs, outputs)
model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=1e-3), metrics=['accuracy'])
model.summary()
```

Output:

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100)	0	-
embedding (Embedding)	(None, 100, 128)	202,752	input_layer[0][0]
spatial_dropout1d (SpatialDropout1D)	(None, 100, 128)	0	embedding[0][0]
lstm (LSTM)	(None, 100, 64)	49,408	spatial_dropout1…
gru (GRU)	(None, 100, 64)	37,248	spatial_dropout1…
concatenate (Concatenate)	(None, 100, 128)	0	lstm[0][0], gru[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 128)	0	concatenate[0][0]
batch_normalization (BatchNormalizatio…	(None, 128)	512	global_max_pooli…
dense (Dense)	(None, 128)	16,512	batch_normalizat…
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 24)	3,096	dropout[0][0]

Total params: 309,528 (1.18 MB)

Trainable params: 309,272 (1.18 MB)

Non-trainable params: 256 (1.00 KB)

D.8: Training Setup

Code:

8. Model training

```
# Step 3: Training callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, verbose=1),
    ModelCheckpoint('best_model.h5', save_best_only=True, monitor='val_loss')
]
```

```
# Step 4: Train the model
history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    epochs=15,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)
```

Output:

```
Epoch 1/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - accuracy: 0.0671 - loss: 3.1644
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 18s 163ms/step - accuracy: 0.0679 - loss: 3.1613 - val_accuracy: 0.1354 - val_loss: 3.1603 - learning_rate: 0.0010
Epoch 2/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step - accuracy: 0.3123 - loss: 2.6882
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 183ms/step - accuracy: 0.3154 - loss: 2.6822 - val_accuracy: 0.3542 - val_loss: 3.0996 - learning_rate: 0.0010
Epoch 3/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step - accuracy: 0.6017 - loss: 1.8670
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 189ms/step - accuracy: 0.6027 - loss: 1.8685 - val_accuracy: 0.4375 - val_loss: 2.9548 - learning_rate: 0.0010
Epoch 4/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - accuracy: 0.7372 - loss: 1.1596
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 9s 145ms/step - accuracy: 0.7385 - loss: 1.1542 - val_accuracy: 0.5104 - val_loss: 2.7803 - learning_rate: 0.0010
Epoch 5/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step - accuracy: 0.8559 - loss: 0.6734
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 124ms/step - accuracy: 0.8565 - loss: 0.6706 - val_accuracy: 0.5625 - val_loss: 2.5843 - learning_rate: 0.0010
Epoch 6/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step - accuracy: 0.9111 - loss: 0.4026
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 179ms/step - accuracy: 0.9118 - loss: 0.4009 - val_accuracy: 0.7812 - val_loss: 2.3440 - learning_rate: 0.0010
Epoch 7/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.9577 - loss: 0.2490
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 125ms/step - accuracy: 0.9580 - loss: 0.2479 - val_accuracy: 0.8542 - val_loss: 2.0987 - learning_rate: 0.0010
Epoch 8/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - accuracy: 0.9718 - loss: 0.1534
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 124ms/step - accuracy: 0.9719 - loss: 0.1530 - val_accuracy: 0.9479 - val_loss: 1.8479 - learning_rate: 0.0010
Epoch 9/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - accuracy: 0.9898 - loss: 0.1140
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 156ms/step - accuracy: 0.9899 - loss: 0.1145 - val_accuracy: 0.9792 - val_loss: 1.6825 - learning_rate: 0.0010
Epoch 10/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step - accuracy: 0.9932 - loss: 0.0767
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 3s 124ms/step - accuracy: 0.9932 - loss: 0.0765 - val_accuracy: 0.9792 - val_loss: 1.3330 - learning_rate: 0.0010
Epoch 11/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step - accuracy: 0.9885 - loss: 0.0737
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 6s 151ms/step - accuracy: 0.9886 - loss: 0.0733 - val_accuracy: 0.9688 - val_loss: 1.1546 - learning_rate: 0.0010
Epoch 12/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 0.9990 - loss: 0.0396
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 4s 123ms/step - accuracy: 0.9989 - loss: 0.0397 - val_accuracy: 0.9688 - val_loss: 0.8806 - learning_rate: 0.0010
Epoch 13/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step - accuracy: 0.9967 - loss: 0.0377
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 191ms/step - accuracy: 0.9967 - loss: 0.0376 - val_accuracy: 0.9688 - val_loss: 0.6887 - learning_rate: 0.0010
Epoch 14/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - accuracy: 0.9986 - loss: 0.0356
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 8s 123ms/step - accuracy: 0.9985 - loss: 0.0356 - val_accuracy: 0.9792 - val_loss: 0.4987 - learning_rate: 0.0010
Epoch 15/15
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step - accuracy: 1.0000 - loss: 0.0272
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
27/27 ━━━━━━━━━━━━━━━━━━━━ 5s 125ms/step - accuracy: 0.9999 - loss: 0.0272 - val_accuracy: 0.9792 - val_loss: 0.3695 - learning_rate: 0.0010
```

D.8: Model Evaluation

Code:

```
# Step 5: Evaluate model
y_pred = model.predict(X_test)
y_test_labels = np.argmax(y_test, axis=1)
y_pred_labels = np.argmax(y_pred, axis=1)

print("\nClassification Report:\n")
print(classification_report(y_test_labels, y_pred_labels, target_names=label_encoder.classes_))
```

Output:

```
Classification Report:
```

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	10
Arthritis	1.00	1.00	1.00	10
Bronchial Asthma	1.00	1.00	1.00	10
Cervical spondylosis	1.00	1.00	1.00	10
Chicken pox	1.00	0.90	0.95	10
Common Cold	1.00	1.00	1.00	10
Dengue	0.83	1.00	0.91	10
Dimorphic Hemorrhoids	1.00	1.00	1.00	10
Fungal infection	1.00	1.00	1.00	10
Hypertension	1.00	1.00	1.00	10
Impetigo	0.91	1.00	0.95	10
Jaundice	1.00	1.00	1.00	10
Malaria	1.00	1.00	1.00	10
Migraine	1.00	1.00	1.00	10
Pneumonia	1.00	1.00	1.00	10
Psoriasis	1.00	0.80	0.89	10
Typhoid	1.00	0.90	0.95	10
Varicose Veins	1.00	1.00	1.00	10
allergy	0.91	1.00	0.95	10
diabetes	1.00	0.90	0.95	10
drug reaction	0.83	1.00	0.91	10
gastroesophageal reflux disease	1.00	1.00	1.00	10
peptic ulcer disease	1.00	1.00	1.00	10
urinary tract infection	1.00	0.90	0.95	10
accuracy			0.97	240
macro avg	0.98	0.98	0.98	240
weighted avg	0.98	0.97	0.98	240

D.9 : Key Metric

Code:

```python
from sklearn.metrics import f1_score

# Predictions for training data
y_train_pred = model.predict(X_train)
y_train_pred_classes = np.argmax(y_train_pred, axis=1)
y_train_true_classes = np.argmax(y_train, axis=1)  # if y_train is one-hot

# Predictions for test data
y_test_pred = model.predict(X_test)
y_test_pred_classes = np.argmax(y_test_pred, axis=1)
y_test_true_classes = np.argmax(y_test, axis=1)  # if y_test is one-hot

# Calculate F1 macro scores
train_f1_macro = f1_score(y_train_true_classes, y_train_pred_classes, average='macro')
test_f1_macro = f1_score(y_test_true_classes, y_test_pred_classes, average='macro')

print(f"Train F1 Macro Score: {train_f1_macro:.4f}")
print(f"Test F1 Macro Score: {test_f1_macro:.4f}")

# Calculate F1 weighted scores
train_f1_weighted = f1_score(y_train_true_classes, y_train_pred_classes, average='weighted')
test_f1_weighted = f1_score(y_test_true_classes, y_test_pred_classes, average='weighted')

print(f"Train F1 weighted Score: {train_f1_weighted:.4f}")
print(f"Test F1 weighted Score: {test_f1_weighted:.4f}")
```

Output:

```
Train F1 Macro Score: 0.9979
Test F1 Macro Score: 0.9751
Train F1 weighted Score: 0.9979
Test F1 weighted Score: 0.9751
```

E.1: Modelling approach Transformer based BERT Model

Read the data:

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

Read the Symptoms and Disease data

```
]
  # Load data
  Symp_Disease_data = pd.read_csv("/content/Train_data.csv")
  Symp_Disease_data.shape
```

E.2: Data Cleanup

Code:

```
def clean_text(text):
    text = text.lower()
    text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)
    return text.strip()

# Apply cleaning - replace 'text_column' with the actual text feature name
Symp_Disease_data['cleaned_text'] = Symp_Disease_data['text'].apply(clean_text)
```

Output:

```
# before cleaning
print(Symp_Disease_data['text'][0],"\n")

#after cleaning
print(Symp_Disease_data['cleaned_text'][0])
```

```
I have been experiencing a skin rash on my arms, legs, and torso for the past few weeks. It is red, itchy, and covered in dry, sca

i have been experiencing a skin rash on my arms legs and torso for the past few weeks it is red itchy and covered in dry scaly pat
```

E.3: Label Encoding

Code:

```
# Encode labels
label_encoder = LabelEncoder()
Symp_Disease_data["label_id"] = label_encoder.fit_transform(Symp_Disease_data["label"])
```

Output:

	label	label_id
580	Acne	0
525	Arthritis	1
948	Bronchial Asthma	2
1099	Cervical spondylosis	3
170	Chicken pox	4
392	Common Cold	5
280	Dengue	6
471	Dimorphic Hemorrhoids	7
302	Fungal infection	8
956	Hypertension	9
230	Impetigo	10
1142	Jaundice	11
1191	Malaria	12
1046	Migraine	13
448	Pneumonia	14
39	Psoriasis	15
135	Typhoid	16
87	Varicose Veins	17
674	allergy	18
873	diabetes	19
780	drug reaction	20
736	gastroesophageal reflux disease	21
820	peptic ulcer disease	22
602	urinary tract infection	23

E.4 : Tokenisation of the train-test datasets

Code:

```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

train_encodings = tokenizer(list(train_texts), truncation=True, padding=True, max_length=64)
val_encodings = tokenizer(list(val_texts), truncation=True, padding=True, max_length=64)
test_encodings = tokenizer(list(test_texts), truncation=True, padding=True, max_length=64)
```

Output:

```
print(train_encodings)
```

```
{'input_ids': [[101, 1045, 2031, 2042, 13417, 10720, 2015, 1998, 19197, 2045, 2003, 1037, 2844, 3255, 1999, 1
```

E.5: Model development

Code:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=24
)
```

E.6: Training setup

Code:

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

```python
# Load model & tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=24)

# Define TrainingArguments (no evaluation_strategy here)
training_args = TrainingArguments(
    output_dir="./results",
    save_steps=1000,          # save checkpoint every N steps
    save_total_limit=1,       # keep only last checkpoint
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    logging_dir="./logs",
    logging_steps=50,
    do_eval=True              # make sure evaluation is enabled
)

# Create Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,  # from your earlier code
    eval_dataset=val_dataset,     # from your earlier code
    tokenizer=tokenizer
)

# Train with manual evaluation each epoch
for epoch in range(int(training_args.num_train_epochs)):
    print(f"\n===== Epoch {epoch+1} / {training_args.num_train_epochs} =====")
    trainer.train()
    print("\n*** Running Evaluation ***")
    trainer.evaluate()
```

Output:

```
===== Epoch 1 / 3 =====
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated a
  return forward_call(*args, **kwargs)
```

[315/315 00:53, Epoch 3/3]

Step	Training Loss
50	3.112200
100	2.495500
150	1.681600
200	1.171400
250	0.777200
300	0.626000

```
*** Running Evaluation ***
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask` is deprecated a
  return forward_call(*args, **kwargs)
```

[23/23 01:54]

```
===== Epoch 2 / 3 =====
```

[315/315 00:54, Epoch 3/3]

Step	Training Loss
50	0.493300
100	0.263100

150	0.133700
200	0.067000
250	0.040400
300	0.031500

```
*** Running Evaluation ***
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask
  return forward_call(*args, **kwargs)

===== Epoch 3 / 3 =====
```

[315/315 00:53, Epoch 3/3]

Step	Training Loss
50	0.023100
100	0.016600
150	0.013400
200	0.010400
250	0.009300
300	0.008900

E.7 : Model Train

Code and output:

```
trainer.train()
```

[315/315 00:55, Epoch 3/3]

Step	Training Loss
50	0.007900
100	0.006400
150	0.005600
200	0.005000
250	0.004800
300	0.004600

```
TrainOutput(global_step=315, training_loss=0.005689582015786852, metrics={'train_runtime': 55.6622, 'train_samples_per_second':
45.273, 'train_steps_per_second': 5.659, 'total_flos': 81601098122880.0, 'train_loss': 0.005689582015786852, 'epoch': 3.0})
```

E.8: Model Evaluation

Code:

```
predictions = trainer.predict(test_dataset)
pred_labels = predictions.predictions.argmax(axis=1)

from sklearn.metrics import classification_report

print(classification_report(test_labels, pred_labels, target_names=label_encoder.classes_))
```

Output:

	precision	recall	f1-score	support
Acne	1.00	1.00	1.00	7
Arthritis	1.00	1.00	1.00	7
Bronchial Asthma	1.00	1.00	1.00	7
Cervical spondylosis	1.00	1.00	1.00	8
Chicken pox	0.89	1.00	0.94	8
Common Cold	1.00	1.00	1.00	7
Dengue	1.00	0.88	0.93	8
Dimorphic Hemorrhoids	1.00	1.00	1.00	8
Fungal infection	1.00	1.00	1.00	7
Hypertension	1.00	1.00	1.00	7
Impetigo	1.00	1.00	1.00	8
Jaundice	1.00	1.00	1.00	7
Malaria	1.00	1.00	1.00	7
Migraine	1.00	1.00	1.00	7
Pneumonia	1.00	1.00	1.00	8
Psoriasis	1.00	1.00	1.00	8
Typhoid	1.00	1.00	1.00	7
Varicose Veins	1.00	1.00	1.00	8
allergy	1.00	1.00	1.00	7
diabetes	1.00	1.00	1.00	8
drug reaction	1.00	1.00	1.00	8
gastroesophageal reflux disease	1.00	1.00	1.00	7
peptic ulcer disease	1.00	1.00	1.00	8
urinary tract infection	1.00	1.00	1.00	8
accuracy			0.99	180
macro avg	1.00	0.99	0.99	180
weighted avg	1.00	0.99	0.99	180

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

E.9 : Key Metric

Code and Output:

```
from sklearn.metrics import f1_score

# Macro F1 = average F1 across all classes (treats all classes equally)
macro_f1 = f1_score(test_labels, pred_labels, average='macro')

# Weighted F1 = average F1 weighted by number of samples in each class
weighted_f1 = f1_score(test_labels, pred_labels, average='weighted')

print(f"Macro F1-score: {macro_f1:.4f}")
print(f"Weighted F1-score: {weighted_f1:.4f}")
```

```
Macro F1-score: 0.9948
Weighted F1-score: 0.9944
```

References

1. Symptom-Based Disease Labeling Dataset. (n.d.). *Kaggle*. Retrieved August 10, 2025, from https://www.kaggle.com/datasets/krish0202/symptom-based-disease-labeling-dataset/data
2. Oesper, L., Merico, D., Isserlin, R., & Bader, G. D. (2011). *WordCloud: A Cytoscape plugin to create a visual semantic summary of networks*. *Source Code for Biology and Medicine, 6*(1), 7. https://doi.org/10.1186/1751-0473-6-7

3. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A. C., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: Experiences from the scikit-learn project. *arXiv Preprint arXiv:1309.0238*. https://doi.org/10.48550/arXiv.1309.0238
4. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html
5. Chollet, F. (2015). Keras. *GitHub repository*. https://github.com/keras-team/keras
6. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv Preprint arXiv:1207.0580*. https://doi.org/10.48550/arXiv.1207.0580
7. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv Preprint arXiv:1412.6980*. https://doi.org/10.48550/arXiv.1412.6980
8. Jones, K. S. (1972). *A statistical interpretation of term specificity and its application in retrieval*. *Journal of Documentation.* [Historical reference for IDF]
9. Das, M., Selvakumar, K., & Alphonse, P. J. A. (2023). A comparative study on TF-IDF feature weighting method and its analysis using unstructured dataset. *arXiv*. https://doi.org/10.48550/arXiv.2308.04037 arXiv
10. Ismiguzel, I. (2025, May 7). Applying text classification using logistic regression: A comparison between BoW and TF-IDF. *Analytics Vidhya*.

https://medium.com/analytics-vidhya/applying-text-classification-using-logistic-regression-a-comparison-between-bow-and-tf-idf-1f1ed1b83640 Medium

11. Oancea, B. (2025). *Text classification using machine learning methods*. *arXiv*. https://doi.org/10.48550/arXiv.2502.19801 arXiv
12. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html
13. TensorFlow Developers. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. *TensorFlow*. https://www.tensorflow.org/
14. “Attention Is All You Need” Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*, 5998–6008. https://doi.org/10.48550/arXiv.1706.03762
15. Transformers (General Reference)

    Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., … Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6
16. Encoder–Decoder Architecture

    Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with

neural networks. *Advances in Neural Information Processing Systems, 27*, 3104–3112. https://doi.org/10.48550/arXiv.1409.3215

17. BERT (Bidirectional Encoder Representations from Transformers)
    Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. https://doi.org/10.48550/arXiv.1810.04805

18. Khan Tusar, M. T. H., & Islam, M. T. (2021). A comparative study of sentiment analysis using NLP and different machine learning techniques on US airline Twitter data. *arXiv*. https://doi.org/10.48550/arXiv.2110.00859 arXiv

19. Lee, S. H., Levin, D., Finley, P. D., & Heilig, C. M. (2019). Chief complaint classification with recurrent neural networks. *Journal of Biomedical Informatics, 93*, 103158. https://doi.org/10.1016/j.jbi.2019.103158 ResearchGatearXiv

20. Al-qarni, S. S., & Algarni, A. (2025). Disease prediction from symptom descriptions using deep learning and NLP technique. *International Journal of Advanced Computer Science and Applications, 16*(5). https://doi.org/10.14569/ijacsa.2025.0160541

21. Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2015). Learning to diagnose with LSTM recurrent neural networks. *arXiv*. https://doi.org/10.48550/arXiv.1511.03677

22. Rasmy, L., Xiang, Y., Xie, Z., Tao, C., & Zhi, D. (2020). Med-BERT: Pre-trained contextualized embeddings on large-scale structured electronic health records for disease prediction. *arXiv*. https://doi.org/10.48550/arXiv.2005.12833

23. Feldges, C. (2022). Text classification with TF-IDF, LSTM, BERT: A comparison of performance. *Medium*. https://medium.com/@claude.feldges/text-classification-with-tf-idf-lstm-bert-a-quantitative-comparison-b8409b556cb3

24. Liu, R., Yu, H., Dligach, D., & Savova, G. K. (2019). A clinical text classification paradigm using weak supervision and deep representation to reduce human effort. *Journal of the American Medical Informatics Association, 26*(12), 1555–1565. https://doi.org/10.1093/jamia/ocz171

25. Dwivedi, V. P. (2025), TS Bharath, Mall Manu . *Module 2 – Symptom to Disease* GitHub. USD-MS-AAI/Module 2 at main · vedpd/USD-MS-AAI

University of San Diego
SHILEY-MARCOS SCHOOL OF ENGINEERING

Team Contributions

Team Member	**Role**
Manu Malla	- Project idea exploration - Exploratory Data Analysis - Developing Statistical Model - Project Documentation for the above tasks and final Report
Bharath TS	- Project idea exploration - Developing Deep Learning Model using BERT - Project Documentation for BERT Model and Final Report
Ved Prakash Dwivedi	- Project identification and task planning - Exploratory Data Analysis - Deep learning model using LSTM and GRU - Git Merges and Read Me document - Project documentation for LSTM, GRU model - Final Report