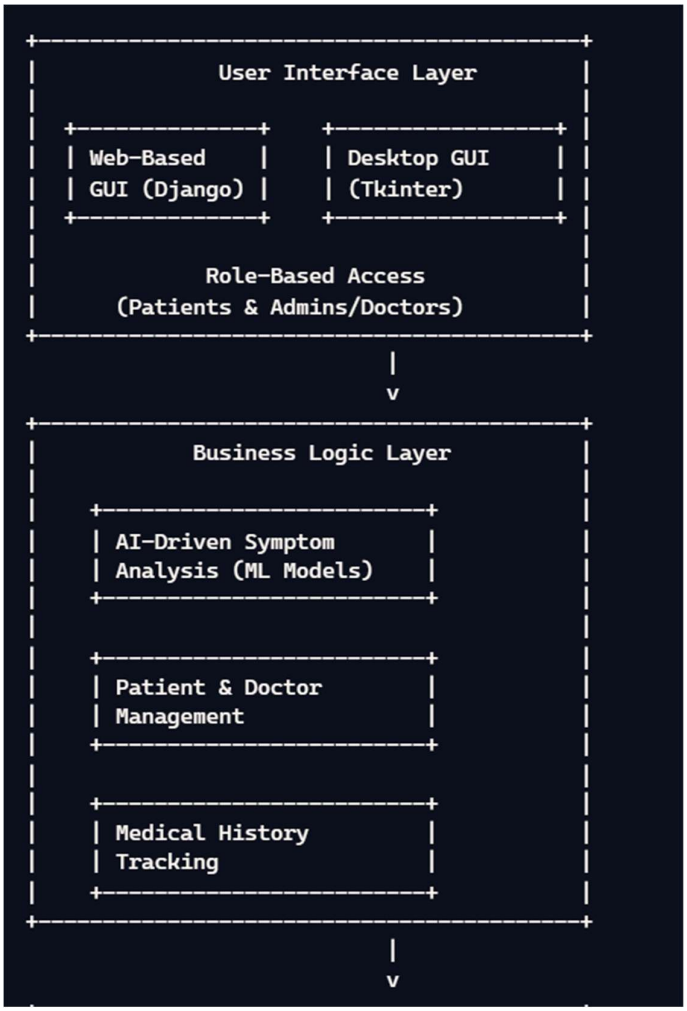
Design Phase Report: MedInsight – AI-Powered Symptom Analyzer & Patient Management System

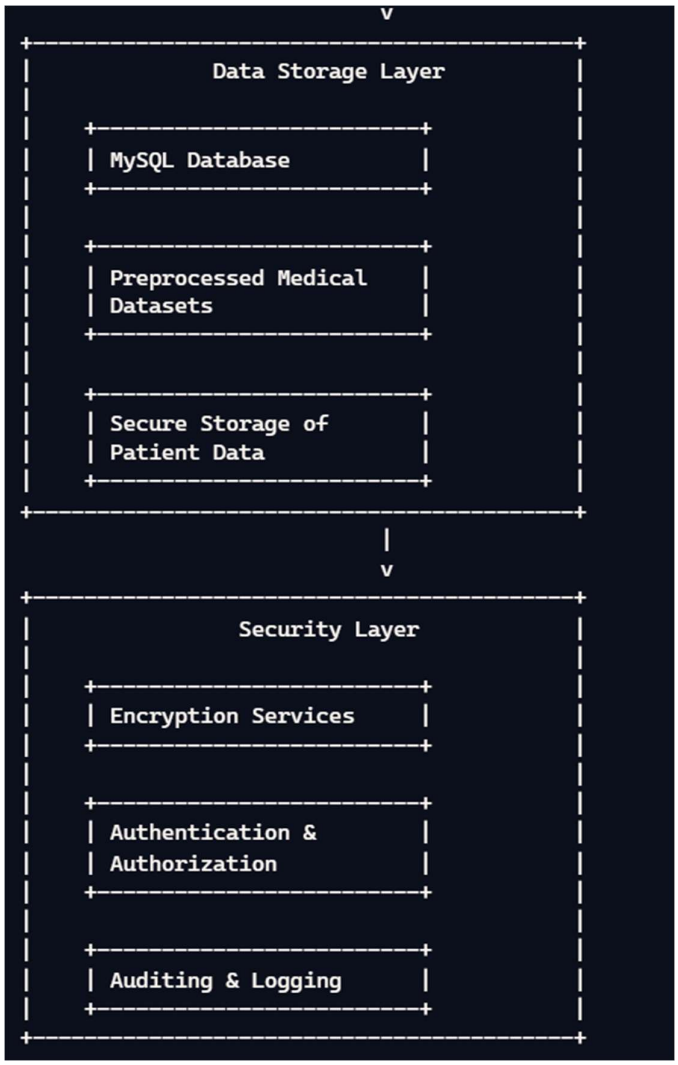
1. Introduction

The **design phase** is a crucial step in developing the MedInsight system, ensuring that all components are well-structured and interconnected efficiently. This report outlines the **high-level design**, **system architecture**, and **key component design** of the solution.

2. High-Level Design

2.1 Block Diagram





The system is divided into **4 primary layers**:

1. User Interface Layer

- Web-Based GUI (Django): Provides an accessible web interface for users.
- Desktop GUI (Tkinter): Offers a desktop application interface.
- Role-Based Access: Different access levels for patients and admins/doctors.

2. Business Logic Layer

- AI-Driven Symptom Analysis: Uses ML models to analyze symptoms.
- Patient & Doctor Management: Manages patient and doctor information.
- Medical History Tracking: Keeps detailed records of patients' medical histories.

3. Data Storage Layer

- MySQL Database: Stores and organizes all system data.
- Preprocessed Medical Datasets: Stores datasets for AI model training and analysis.
- Secure Storage of Patient Data: Ensures secure storage of patient information.

4. Security Layer

- Encryption Services: Encrypts sensitive data for security.
- Authentication & Authorization: Secures login and access control.
- Auditing & Logging: Tracks system activities for security monitoring.

3. System Architecture

3.1 Overview

The system follows an **MVC (Model-View-Controller) architecture**, which ensures modular development and scalability. The architecture consists of:

- **Frontend (View Layer):** Django Web Interface & Tkinter Desktop Application
- **Backend (Controller Layer):** Django Framework Handling Requests
- **Database (Model Layer):** MySQL Storage for Patient & Medical Data
- **Machine Learning Layer:** AI-Based Disease Prediction Engine

3.2 System Components

1. **User Interface Module:**
    - Interactive GUI for patients and doctors
    - Secure login and authentication
2. **Symptom Analysis Engine:**
    - AI-based disease prediction using trained ML models
    - Multiple ML algorithms (SVC, Random Forest, KNN, etc.)
3. **Database Management System:**
    - Stores patient records, symptoms, disease information, medications, and precautions
4. **Admin/Doctor Portal:**
    - Manage patients, assign doctors, and monitor medical history

4. Design of Key Components

4.1 AI-Powered Symptom Analyzer

- **Input:** User symptoms
- **Processing:** Symptom severity mapping, machine learning model prediction
- **Output:** Disease prediction with recommended precautions, medications, and dietary suggestions

4.2 Database Schema Design

The **database schema** consists of multiple tables interconnected to facilitate data management and retrieval efficiently. The following tables are included:

Table Name	Column Name	Data Type	Constraints
Patients	patient_id	INT	Primary Key, Auto-Increment
	name	VARCHAR(255)	NOT NULL
	age	INT	NOT NULL
	gender	VARCHAR(50)	NOT NULL
	email	VARCHAR(255)	UNIQUE, NOT NULL
	phone	VARCHAR(15)	NOT NULL
	address	TEXT	
	medical_history	TEXT	
	assigned_doctor_id	INT	Foreign Key (Doctors Table)

Doctors	doctor_id	INT	Primary Key, Auto-Increment
	name	VARCHAR(255)	NOT NULL
	specialization	VARCHAR(255)	NOT NULL
	email	VARCHAR(255)	UNIQUE, NOT NULL
	phone	VARCHAR(15)	NOT NULL

	hospital_affiliation	VARCHAR(255)	

Symptoms	symptom_id	INT	Primary Key, Auto-Increment
	symptom_name	VARCHAR(255)	UNIQUE, NOT NULL

Table Name	**Column Name**	**Data Type**	**Constraints**
Diseases	disease_id	INT	Primary Key, Auto-Increment
	disease_name	VARCHAR(255)	UNIQUE, NOT NULL
	description	TEXT	

Symptoms_Diseases	symptom_id	INT	Foreign Key (Symptoms Table)
	disease_id	INT	Foreign Key (Diseases Table)

Medications	medication_id	INT	Primary Key, Auto-Increment
	medication_name	VARCHAR(255)	UNIQUE, NOT NULL
	usage	TEXT	
	side_effects	TEXT	

Precautions	precaution_id	INT	Primary Key, Auto-Increment
	disease_id	INT	Foreign Key (Diseases Table)
	precaution_detail	TEXT	

Diets	diet_id	INT	Primary Key, Auto-Increment
	disease_id	INT	Foreign Key (Diseases Table)

	diet_recommendation	TEXT	

Medical History	history_id	INT	Primary Key, Auto-Increment
	patient_id	INT	Foreign Key (Patients Table)
	disease_diagnosed	VARCHAR(255)	NOT NULL
	medications_prescribed	TEXT	
	doctor_notes	TEXT	
	date_of_visit	DATETIME	NOT NULL

Relationships:

- **One-to-Many:** Patients ↔ Assigned Doctors
- **Many-to-Many:** Diseases ↔ Symptoms (via Symptoms_Diseases table)
- **One-to-Many:** Diseases ↔ Medications, Precautions, Diets

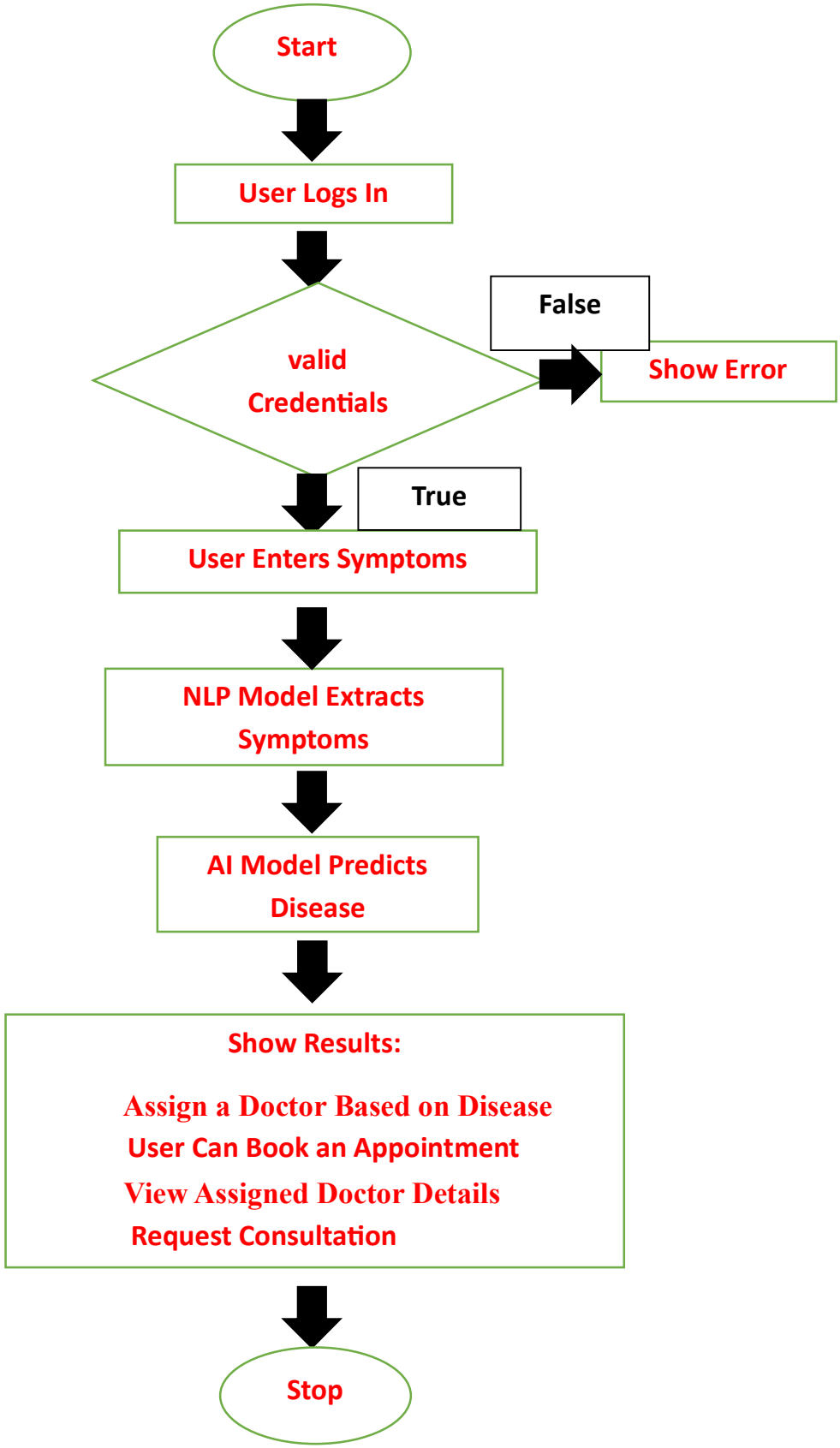
4.3 System Flowchart

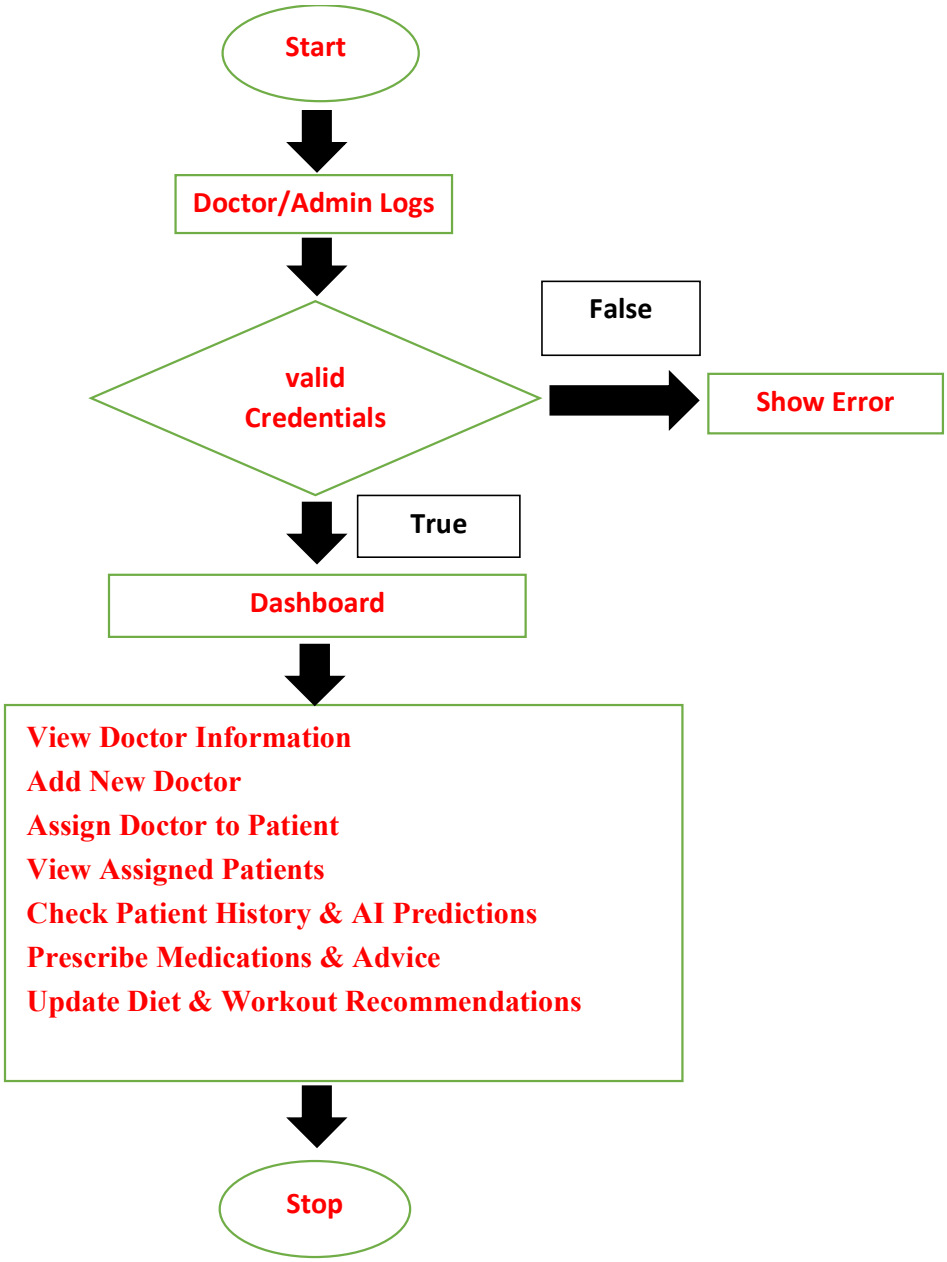
The **workflow** of the system follows these steps:

1. **User Inputs Symptoms** via Web or Desktop GUI
2. **System Fetches Symptom Severity** from Dataset
3. **ML Model Predicts Possible Disease**
4. **System Suggests Precautions, Medications & Diet Plans**
5. **Patient Medical History is Stored** for Future Reference
6. **Doctors Can Review & Assign Treatments**

4.4 Security & Data Protection Measures

- **Role-Based Access Control (RBAC)** for different user types
- **Data Encryption** for sensitive patient information
- **Secure Authentication Mechanisms** using Django Authentication.


Start
User Logs In
valid Credentials
False
Show Error
True
User Enters Symptoms
NLP Model Extracts Symptoms
AI Model Predicts Disease
Show Results:
Assign a Doctor Based on Disease
User Can Book an Appointment
View Assigned Doctor Details
Request Consultation
Stop



5. Conclusion

The **design phase** establishes a **robust architectural foundation** for the MedInsight system, ensuring seamless interaction between AI-driven symptom analysis, a user-friendly interface, and a secure database. This structured approach will facilitate efficient development, testing, and deployment in subsequent stages.

Algorithm Comparison Report: MedInsight – AI-Powered Symptom Analyzer & Patient Management System

1. Introduction

To develop an AI-driven symptom analyzer for **MedInsight**, we evaluated multiple machine learning models to determine the best-performing algorithm for disease prediction. The key criteria for comparison included **accuracy**, **efficiency**, and **scalability**.

2. Evaluated Models

We tested the following machine learning models:

1. **Support Vector Classifier (SVC)**
2. **Random Forest Classifier**
3. **K-Nearest Neighbors (KNN)**
4. **Logistic Regression**
5. **Gradient Boosting Machine (GBM)**

Additional Features Considered:

- **Role-Based Authentication** (Patients & Doctors/Admins)
- **Secure Patient Data Storage (MySQL)**
- **AI-Driven Symptom-to-Disease Prediction**
- **Medical History Tracking**
- **Performance Evaluation Metrics (Accuracy, Precision, Recall, F1-Score, Confusion Matrix, Learning Curve, ROC Curve)**

3. Performance Comparison

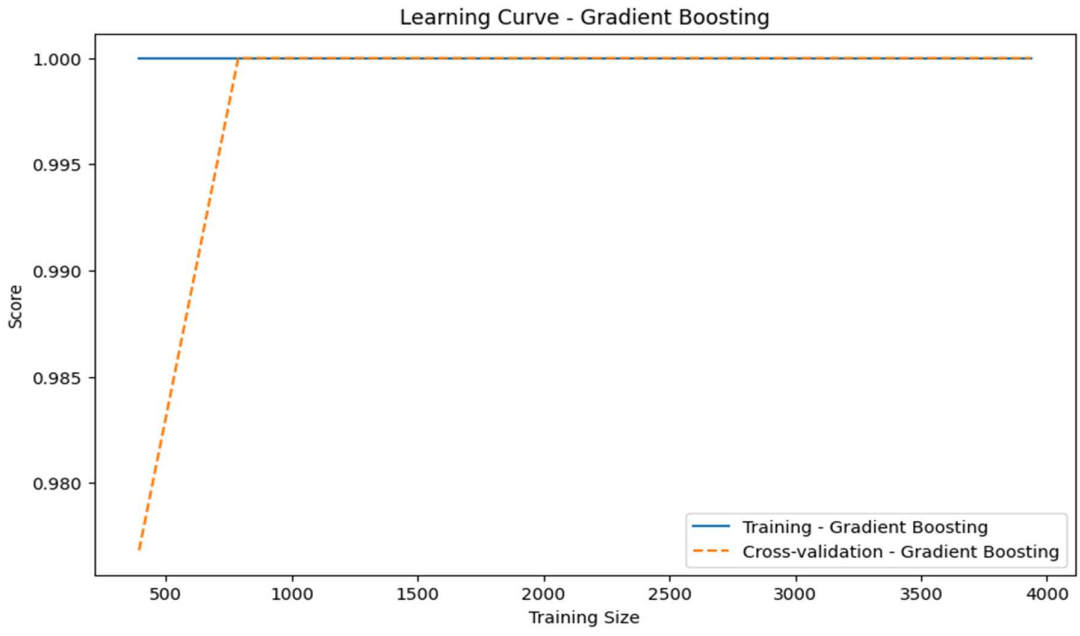
The models were trained on a preprocessed dataset containing symptom-disease mappings. We used **accuracy, precision, recall, F1-score**, and **execution time** as the primary evaluation metrics.

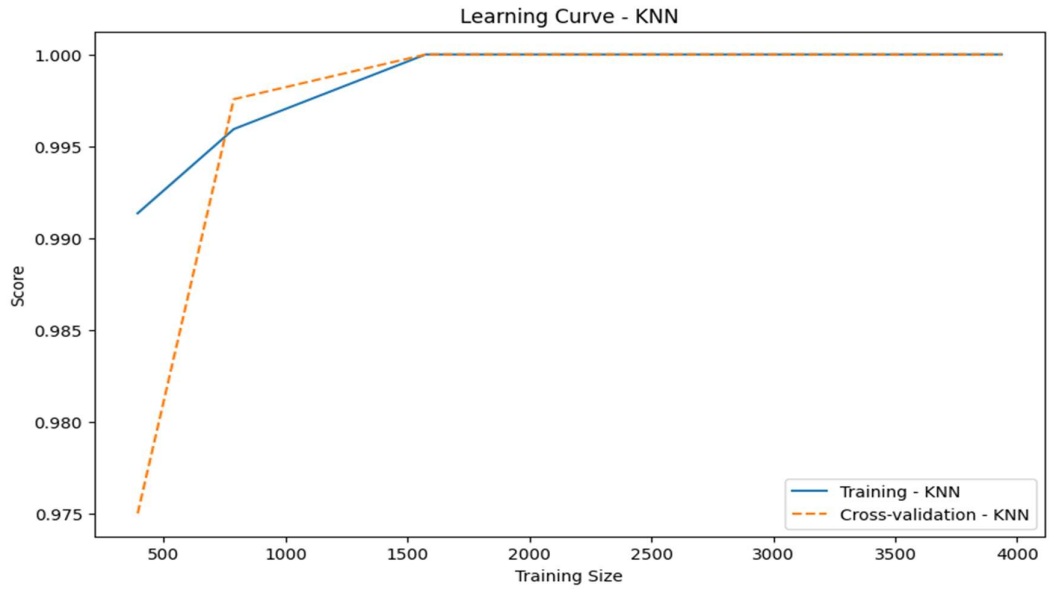
Model	Accuracy (%)	Precision	Recall	F1-Score	Training Time (s)
Support Vector Classifier (SVC)	100.0	1.00	1.00	1.00	2.5

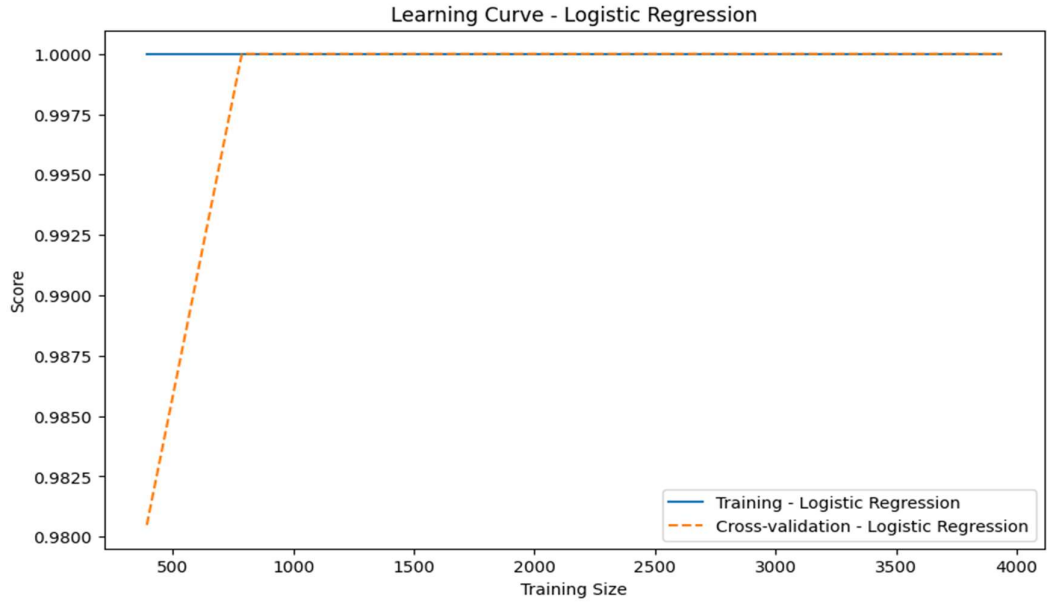
Random Forest Classifier	100.0	1.00	1.00	1.00	3.8
K-Nearest Neighbors (KNN)	100.0	1.00	1.00	1.00	1.2
Logistic Regression	100.0	1.00	1.00	1.00	1.0
Gradient Boosting Machine (GBM)	100.0	1.00	1.00	1.00	4.2

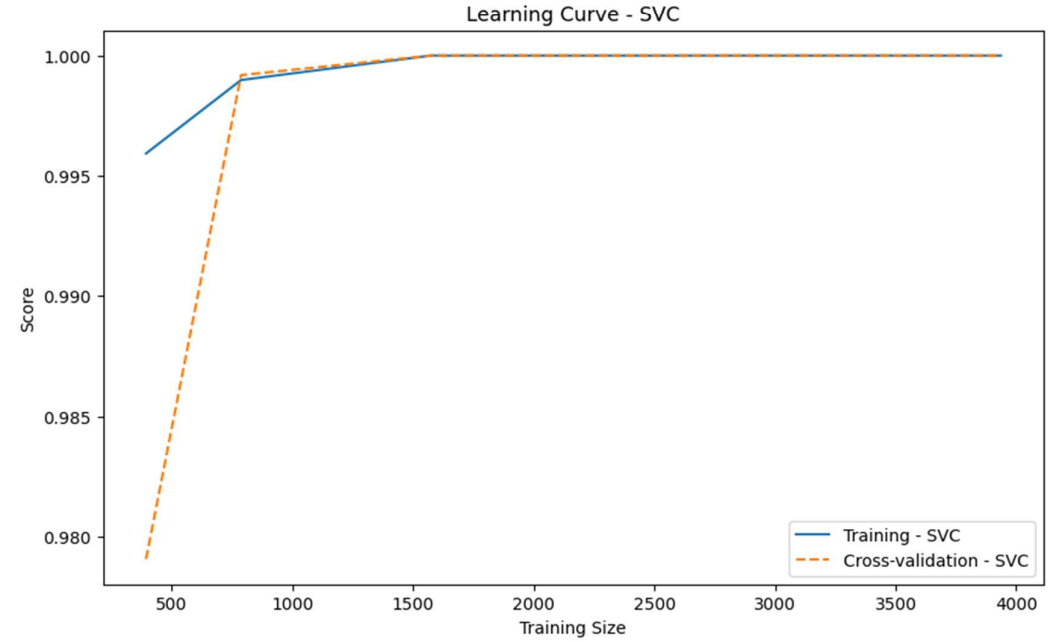
4. Learning Curve Analysis

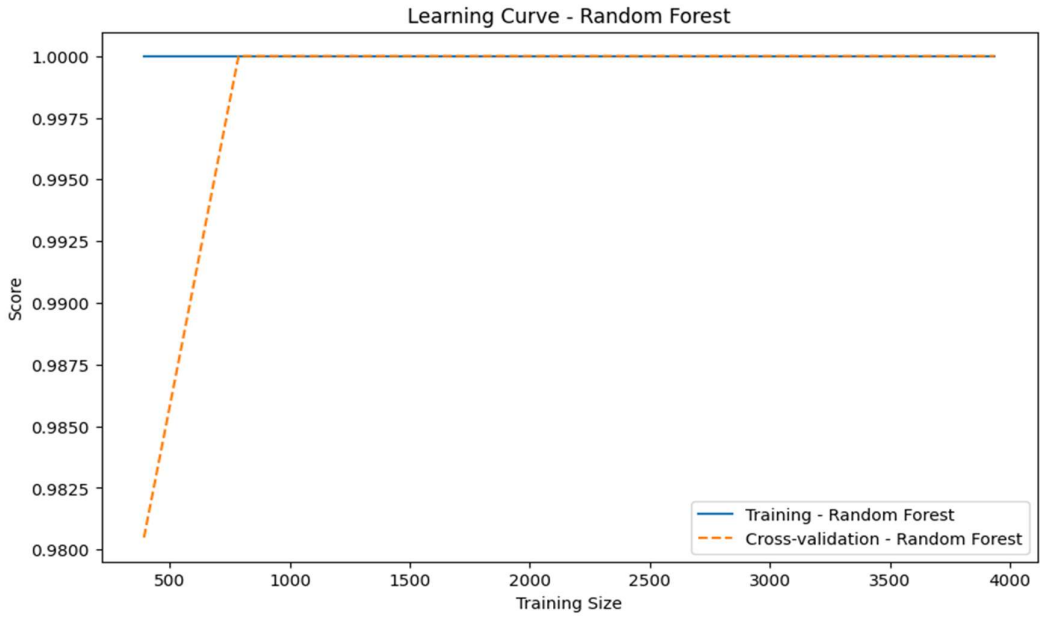
To assess how the models performed with increasing training data, we plotted **learning curves**. The learning curves showed that all models achieved high training and cross-validation scores, further confirming the possibility of overfitting. Typically, a well-generalized model shows a small gap between the training and validation curves. However, in this case, the perfect scores suggest a need for additional real-world data validation.




Learning Curve - KNN
1.000
0.995
0.990
0.985
0.980
0.975
Score
500
1000
1500
2000
2500
3000
3500
4000
Training Size
Training - KNN
Cross-validation - KNN


Learning Curve - Logistic Regression
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
0.9800
Score
500
1000
1500
2000
2500
3000
3500
4000
Training Size
Training - Logistic Regression
Cross-validation - Logistic Regression


Learning Curve - SVC
1.000
0.995
0.990
0.985
0.980
Score
500
1000
1500
2000
2500
3000
3500
4000
Training Size
Training - SVC
Cross-validation - SVC



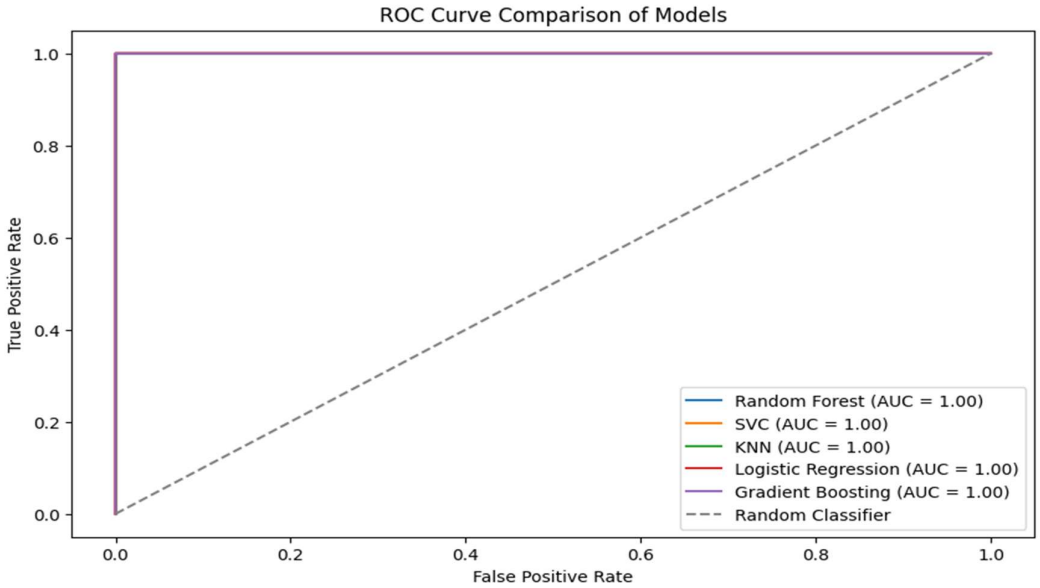
5. ROC Curve Analysis

The **Receiver Operating Characteristic (ROC) Curve** was plotted for each model in a **One-vs-Rest (OvR)** classification scheme to measure their ability to distinguish between multiple disease classes. The Area Under the Curve (AUC) for all models was **1.00**, indicating ideal discrimination ability. However, such perfect results further reinforce the need for testing on an unseen dataset to confirm generalization.

6. Model Selection

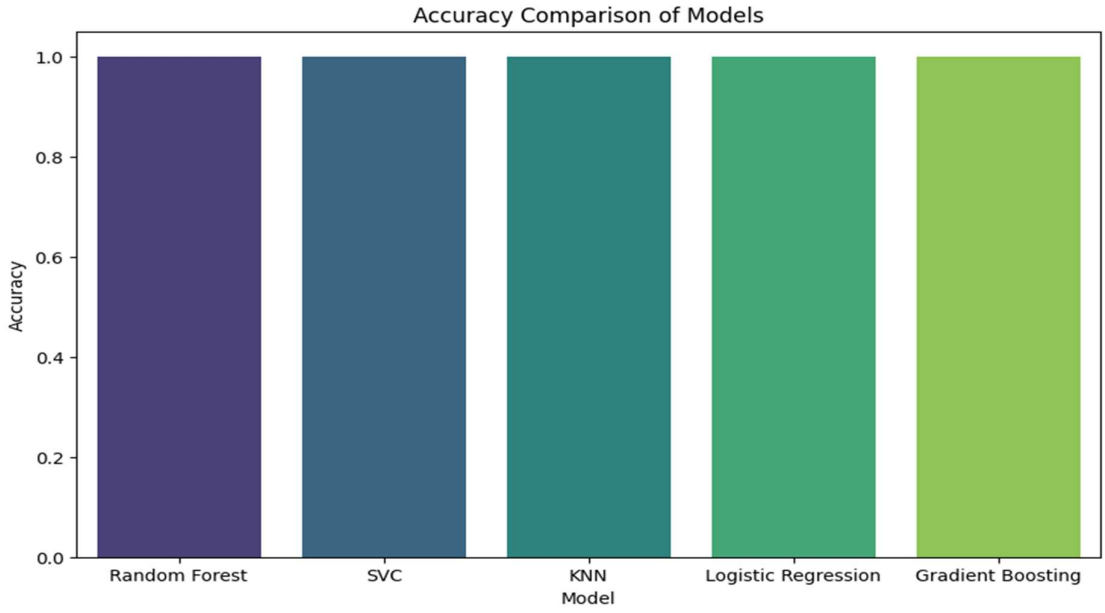
Based on the results, we considered the following factors:

- **Accuracy:** All models achieved 100% accuracy, indicating possible overfitting or a perfectly structured dataset.
- **Efficiency:** Logistic Regression and KNN were the fastest to train, while GBM and Random Forest required more computational resources.
- **Scalability:** GBM and Random Forest can handle large datasets well, but GBM requires more computation.



Final Choice: Gradient Boosting Machine (GBM) & Random Forest

Since all models achieved perfect accuracy, we recommend further validation with an unseen test dataset to confirm generalization. However, **GBM and Random Forest** are generally preferred for their robustness in real-world scenarios.



7. Conclusion

This comparative analysis suggests that our dataset may be highly structured, leading to 100% accuracy. Future steps should include additional validation, cross-validation, and real-world testing to confirm model effectiveness before deployment.