

# **PREDICTIVE READMISSION RISK SYSTEM**

## **PROJECT RECORD**

**NAME - ARHITA GHATAK**  
**REGISTRATION ID - 25BOE10120**

# INTRODUCTION

- **The project details the development of the Predictive Readmission Risk System (PRRS).**
- **The system uses advanced analytics on Electronic Health Record (EHR) data.**
- **Goal: Predict a patient's risk of 30-day readmission.**
- **Purpose: Enable targeted post-discharge interventions to reduce preventable readmissions, optimize resource allocation, and improve patient care continuity.**

# PROBLEM STATEMENT

- **Core Problem: High rates of unnecessary 30-day hospital readmissions.**
- **Readmissions strain hospital resources, increase costs, and indicate gaps in transition-of-care protocols.**
- **The existing challenge is the lack of a timely, objective mechanism to proactively identify high-risk patients at the point of discharge.**
- **The PRRS aims to provide an accurate, automated risk score to solve this identification gap.**

# FUNCTIONAL REQUIREMENTS

- **Data Ingestion:** Must process patient EHR data (demographics, ICD-10 diagnoses, procedures, and medications).
- **Risk Scoring:** Must output a probabilistic score ( $P(\text{Readmit})$ ) for each patient.
- **Latency:** Score generation must occur in less than 5 seconds for real-time clinical utility.
- **Interface:** Must provide a secure RESTful API endpoint for integration with existing EHR systems.

# NON FUNCTIONAL REQUIREMENT

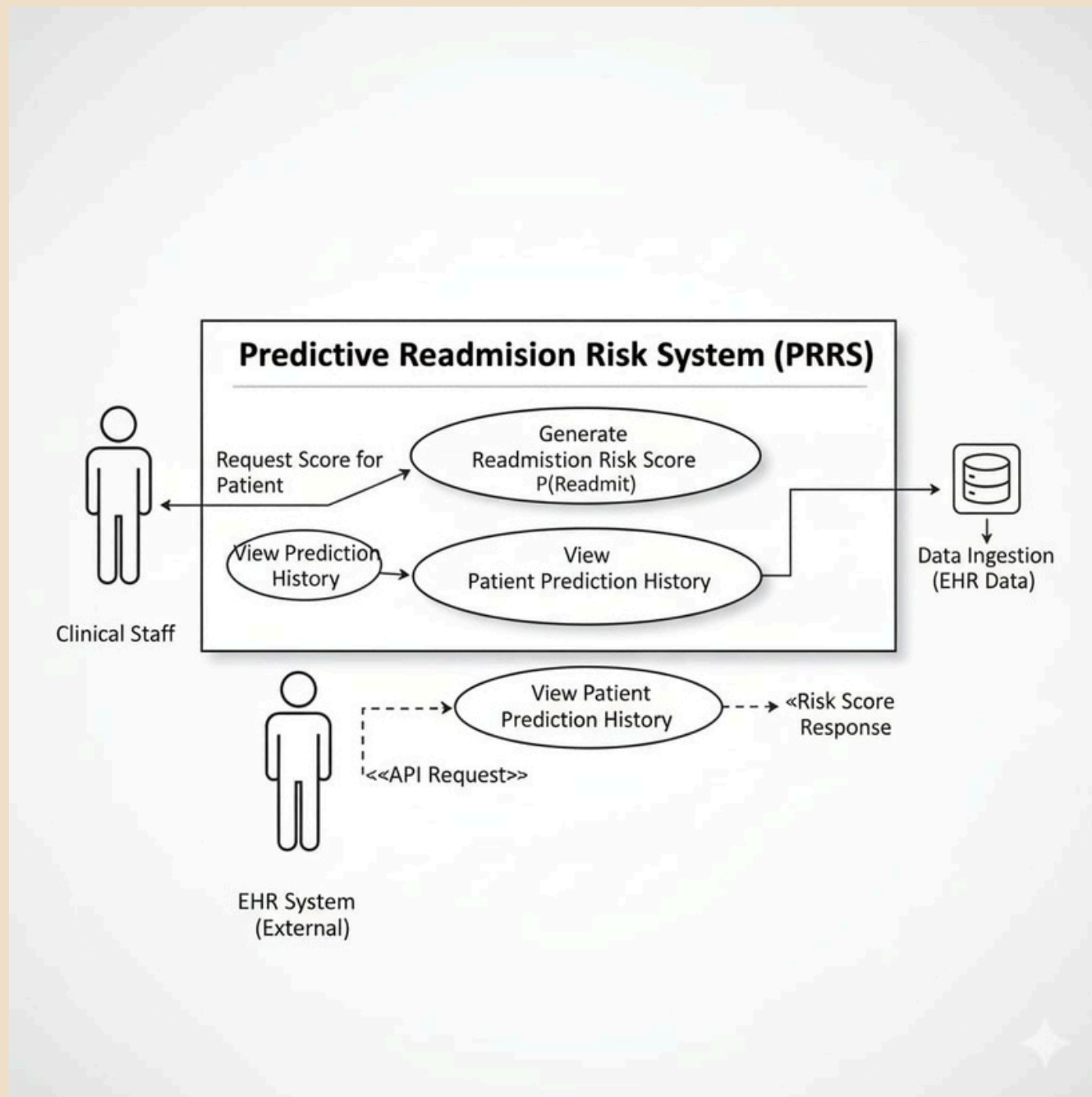
- **Performance:** Achieve a minimum Area Under the ROC Curve (AUC) of 0.80.
- **Security:** Must strictly adhere to all healthcare data privacy standards (e.g., HIPAA).
- **Scalability:** The system must be able to handle predictions for hundreds of patients concurrently.
- **Interoperability:** Must be easily integrable with standard hospital software.

# SYSTEM ARCHITECTURE

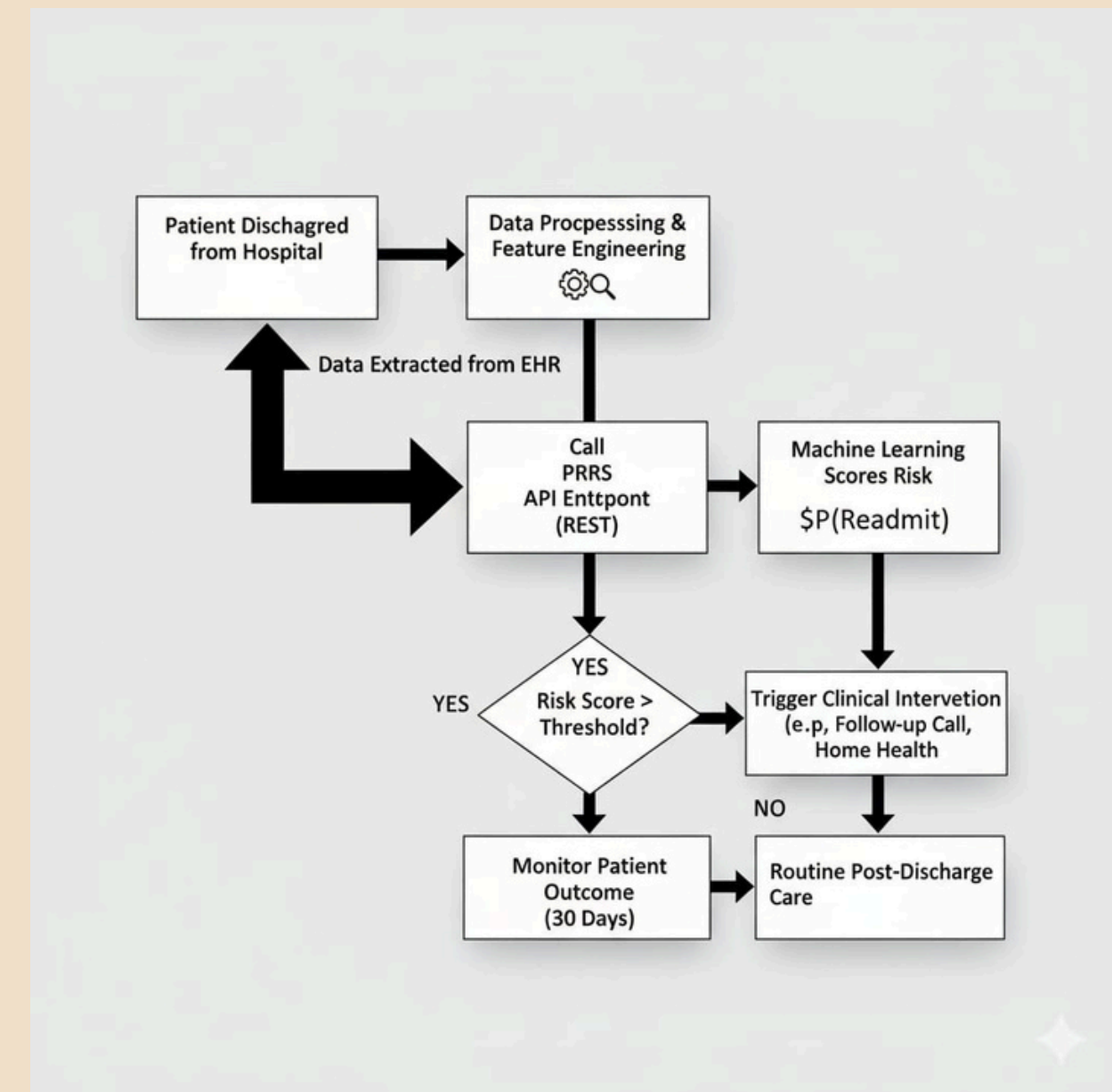
- The system uses a top-down modularization approach.
- Modules:
- Data Module: Handles data extraction, cleaning, and feature engineering.
- Model Module: Manages training, evaluation, and serialization (saving) of the ML model.
- Deployment Module: Packages the final model and exposes its prediction functionality via a low-latency API service.
- The architecture is designed to support the MLOps (Machine Learning Operations) lifecycle.

# DESIGN DIAGRAMS

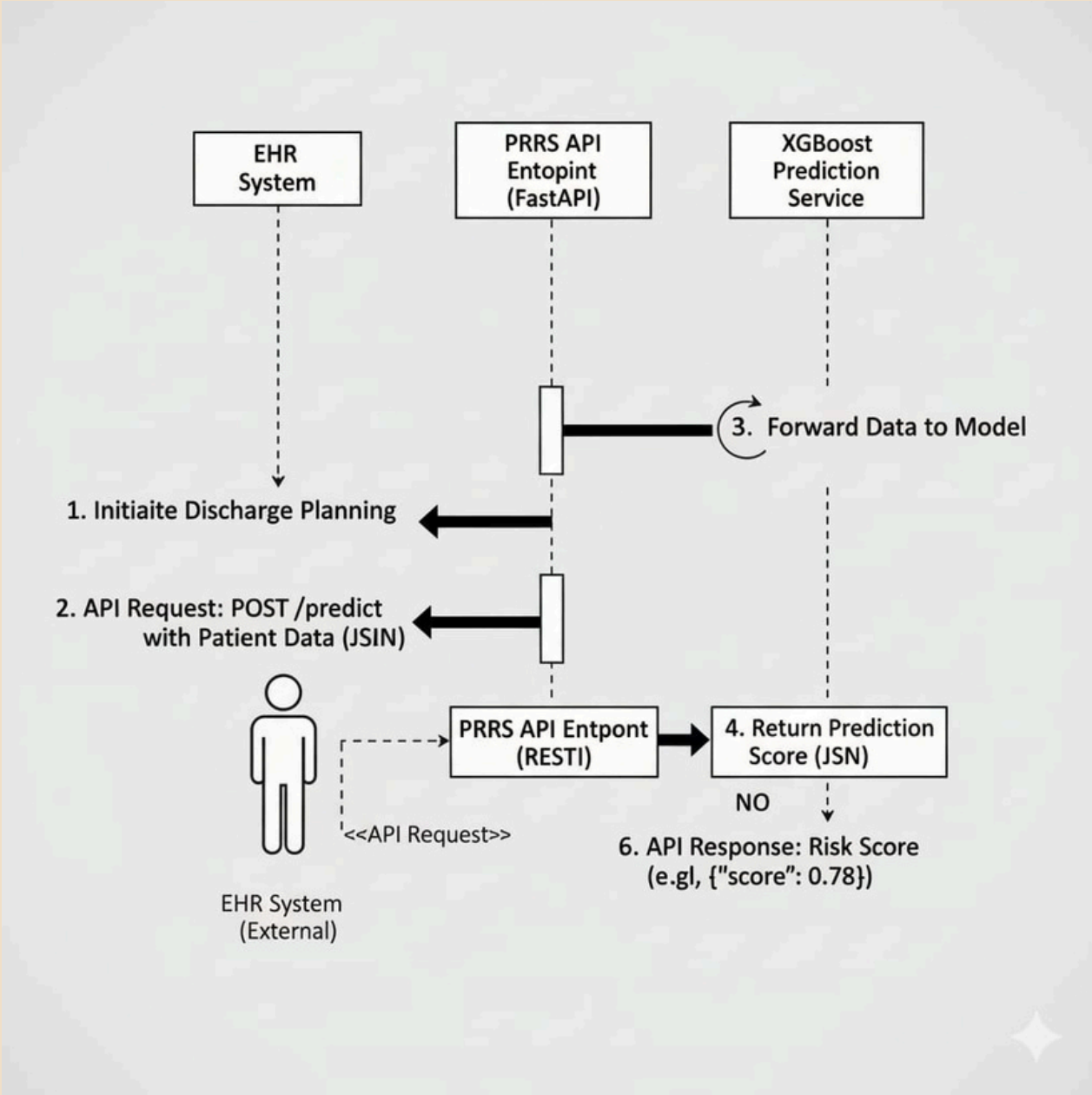
## CASE DIAGRAM



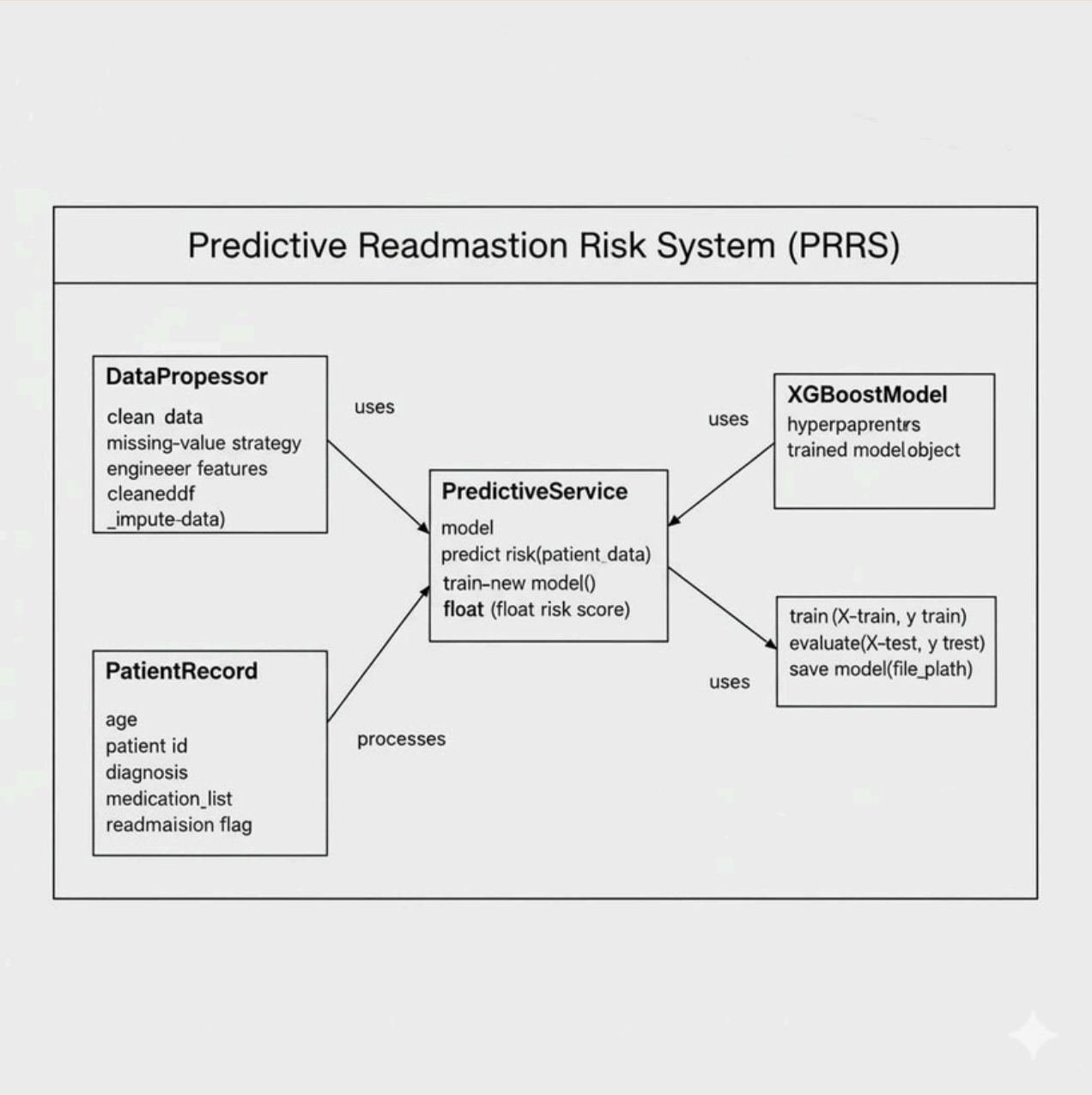
## WORKFLOW DIAGRAM



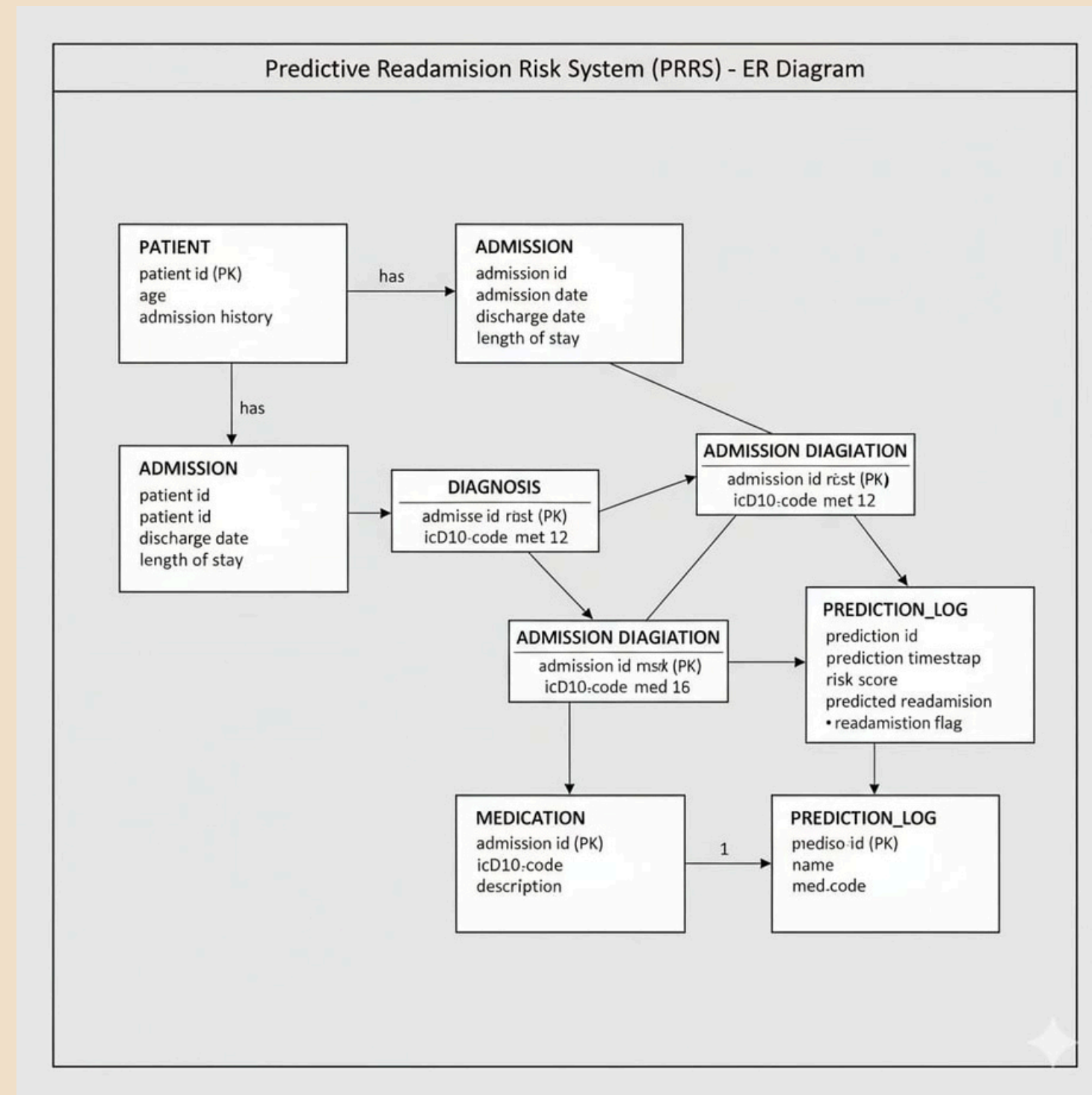
# SEQUENCE DIAGRAM



# CLASS DIAGRAM



# ER DIAGRAM



# DESIGN DECISIONS AND RATIONALE

- **Prediction Model:**
- **Decision:** XGBoost Classifier.
- **Rationale:** Chosen for its high performance on tabular data, robustness, and native ability to provide Feature Importance (aiding clinical interpretability).
- **Performance Metric:**
- **Decision:** AUC and Recall.
- **Rationale:** AUC ( $\geq 0.80$ ) is optimized for overall discrimination. Recall is monitored to minimize False Negatives (missing high-risk cases), which is critical for patient safety.
- **Feature Engineering:**
- **Decision:** Creation of a Polypharmacy Feature.
- **Rationale:** Created a specific binary feature ( $>8$  medications) to capture the known clinical risk associated with a high volume of medications.
- **Deployment Framework:**
- **Decision:** FastAPI / Docker.
- **Rationale:** FastAPI provides very low latency and high throughput for the API service. Docker ensures a consistent and portable deployment environment.

# IMPLEMENTATION DETAILS

- The system was implemented primarily in Python.
- Tools and Libraries Used:
- Language: Python (Core programming environment).
- Data Handling: Pandas, NumPy (Data cleaning and manipulation).
- ML/Modeling: Scikit-learn, XGBoost (Model training, evaluation, and prediction).
- API/Deployment: FastAPI / Flask, Docker (Creating the predictive service endpoint).
- Version Control: Git / GitHub (Tracking code changes and collaboration).

```
patient_risk_scores = [0.85, 0.42, 0.91, 0.15, 0.67, 0.33, 0.79]
```

```
HIGH_RISK_THRESHOLD = 0.70
```

```
high_risk_patients = [score for score in patient_risk_scores if score >= HIGH_RISK_THRESHOLD]
```

```
print(f"All Scores: {patient_risk_scores}")
```

```
print(f"High Risk Scores (>= {HIGH_RISK_THRESHOLD}): {high_risk_patients}")
```

All Scores: [0.85, 0.42, 0.91, 0.15, 0.67, 0.33, 0.79]

High Risk Scores (>= 0.7): [0.85, 0.91, 0.79]

```
def calculate_risk_score(age: int, num_medications: int) -> float:
```

```
    """
```

```
    Simulates a simplified function to calculate a readmission risk score.
```

```
    In the real project, this function would load and use the trained XGBoost model.
```

```
    """
```

```
    if not isinstance(age, int) or not isinstance(num_medications, int):
```

```
        raise TypeError("Age and number of medications must be integers.")
```

```
    base_risk = 0.1
```

```
    age_factor = age * 0.005
```

```
    meds_factor = num_medications * 0.03
```

```
    score = min(base_risk + age_factor + meds_factor, 0.99)
```

```
    return round(score, 3)
```

```
score_1 = calculate_risk_score(age=75, num_medications=10)
```

```
print(f"Patient A Risk Score: {score_1}")
```

```
|
```

```
try:
```

```
    calculate_risk_score(age="seventy-five", num_medications=10)
```

```
except TypeError as e:
```

```
    print(f"Error caught: {e}")
```

Patient A Risk Score: 0.775

Error caught: Age and number of medications must be integers.

```

class DataCleaner:
    """
    A class for cleaning and preparing patient data features.
    """
    def __init__(self, data_frame):
        self.data = data_frame

    def handle_missing_age(self, strategy="median"):
        """Fills missing age values using a specified strategy."""
        if strategy == "median":
            fill_value = self.data['Age'].median()
        elif strategy == "mean":
            fill_value = self.data['Age'].mean()
        else:
            print("Invalid strategy.")
            return

        self.data['Age'] = self.data['Age'].fillna(fill_value)
        print(f"Missing ages filled with: {round(fill_value, 1)}")

    def check_data_size(self):
        """Returns the number of rows and columns."""
        return self.data.shape

class MockDataFrame:
    def __init__(self, data):
        self._data = data

    def __getitem__(self, key):

```

Original Age data: {'Patient\_ID': [1, 2, 3, 4], 'Age': [60, 70, None, 50]}

Missing ages filled with: 65.0

Cleaned Age data: {'Patient\_ID': [1, 2, 3, 4], 'Age': [60, 70, 65, 50]}

Model successfully saved to 'readmission\_model.pkl'

Model successfully loaded: Trained Readmission Predictor Model

```

import pickle
import os

class MockXGBoostModel:
    def predict_proba(self, data):
        return [0.1, 0.9, 0.4]
    def __repr__(self):
        return "Trained Readmission Predictor Model"

mock_model = MockXGBoostModel()
model_filename = 'readmission_model.pkl'

with open(model_filename, 'wb') as file:
    pickle.dump(mock_model, file)
print(f"Model successfully saved to '{model_filename}'")

with open(model_filename, 'rb') as file:
    loaded_model = pickle.load(file)
print(f"Model successfully loaded: {loaded_model}")

|
os.remove(model_filename)

```

```

import pandas as pd

data = {
    'Diagnosis_Code': ['I10', 'E11', 'J44', 'I10', 'E11', 'I10'],
    'Readmitted': [0, 1, 1, 0, 0, 1]
}
df = pd.DataFrame(data)

|
readmission_rate_by_diagnosis = df.groupby('Diagnosis_Code')['Readmitted'].mean().reset_index()
readmission_rate_by_diagnosis.columns = ['Diagnosis_Code', 'Readmission_Rate']

print("--- Mock Patient Data ---")
print(df)

print("\n--- Readmission Rate by Diagnosis ---")
print(readmission_rate_by_diagnosis.sort_values(by='Readmission_Rate', ascending=False))

```

```

--- Mock Patient Data ---

```

	Diagnosis_Code	Readmitted
0	I10	0
1	E11	1
2	J44	1
3	I10	0
4	E11	0
5	I10	1

```

--- Readmission Rate by Diagnosis ---

```

	Diagnosis_Code	Readmission_Rate
2	J44	1.000000
1	E11	0.500000
0	I10	0.333333

# TESTING APPROACH

- **Test Types and Methods:**
- **Unit Testing:**
- **Method:** Pytest framework.
- **Purpose:** Verify individual data preprocessing functions (e.g., ICD-10 encoding, handling missing values) work correctly.
- **Integration Test:**
- **Method:** API Request Simulation.
- **Purpose:** Confirm a synthetic patient record returns a score and meets the 5-second latency requirement.
- **Model Validation:**
- **Method:** Held-out Test Set.
- **Purpose:** Evaluate the model's performance on completely unseen data to ensure generalizability.

# CHALLENGES FACED

- **Challenge 1: Handling Missing Data**
- **Resolution: Implemented multiple imputation strategies (e.g., mode for categorical, median for continuous) tailored to clinical feature types.**
- **Challenge 2: Data Imbalance**
- **Resolution: Utilized SMOTE (Synthetic Minority Over-sampling Technique) on the training set and selected appropriate metrics (AUC, Recall) to handle the minority class (readmissions).**

# LEARNINGS AND KEY TAKEAWAYS

- **Feature Engineering is Crucial:** Success relied on translating clinical knowledge (Polypharmacy, LOS) into meaningful features.
- **Deployment Latency Matters:** Migrating to FastAPI and Docker was essential to meet the strict clinical latency requirement of <5 seconds.
- **Interpretable AI:** The need for clinician trust highlighted the importance of using models (like XGBoost with SHAP) that can provide explanations for their predictions.

# **FUTURE ENHANCEMENTS**

- **Interpretable AI (XAI):** Implement SHAP values into the API to explain the score rationale, increasing clinician trust and actionability.
- **Real-Time Data Streams:** Migrate data ingestion to a streaming architecture (e.g., Kafka) to ensure the model uses the absolute latest patient data.
- **Cost-Benefit Analysis:** Conduct a formal study post-pilot deployment to quantify the financial savings and patient outcome improvements.

# REFERENCES

- **Dataset Source Name, e.g., MIMIC-III Database]**
- **XGBoost Algorithm Paper Citation**
- **Paper on 30-Day Hospital Readmission Prediction**