Artificial Intelligence Project Based Learning (PBL) Report on

AI-Integrated Patient Monitoring System using Bayesian Reasoning

Submitted in partial fulfilment of the

Requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

Submitted by

Kuppili Nikhilesh Raju (23R11A66H7)

Under the esteemed guidance of

Mr. Shaik Akbar

Associate Professor, CSE (AI & ML) Department, GCET

GEETHANJALI COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering (AI & ML)

(UGC Autonomous)

(Accredited by NAAC with A+ Grade, Approved by AICTE and

Affiliated to JNTUH Cheeryal (V), Keesara (M), Medchal (Dist), Telangana – 501 301.

October – 2025

GEETHANJALI COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Computer Science and Engineering (AI & ML)

(UGC Autonomous)

(Accredited by NAAC with A+ Grade , Approved by AICTE and Affiliated to JNTUH
Cheeryal (V), Keesara (M), Medchal (Dist), Telangana – 501 301.

DERPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(AI & ML)

CERTIFICATE

This is to certify that the Artificial Intelligence Project Based Learning (PBL) Report entitled "**AI-Integrated Patient Monitoring System using Bayesian Reasoning**" is a Bonafide work done and submitted by **Kuppili Nikhilesh Raju (23R11A66H7)**, during the academic year 2025– 2026, in partial fulfilment of the requirement for the award of Bachelor of Technology degree in "**Computer Science and Engineering (AI&ML)**" from Geethanjali College of Engineering and Technology (Accredited by NAAC with "A+" Grade, Approved by AICTE and Affiliated to JNTUH), is a Bonafide record of work carried out by them under guidance and supervision.

Certified further that to the best of my knowledge, the work in this project has not been submitted to any other institution for the award of any degree or diploma.

FACULTY GUIDE

Mr. Shaik Akbar

Professor

CSE-AIML Department

DECLARATION

We hereby declare that the project report entitled “**AI-Integrated Patient Monitoring System using Bayesian Reasoning**” is an original work done and submitted to CSE(AI&ML) Department, from Geethanjali College of Engineering and Technology (Accredited by NAAC with “A+” Grade & NBA, Approved by AICTE and Affiliated to JNTUH), in partial fulfilment of the requirement for the award of Bachelor of Technology in “Computer Science and Engineering (AI&ML)” and it is a record of Bonafide project work carried out by us under the guidance of Mr. Shaik Akbar, Associate Professor, Department of CSE(AI&ML).

Signature of the Student

Kuppili Nikhilesh Raju

(23R11A66H7)

ACKNOWLEDGEMENT

The successful culmination of this project wouldn't be complete without expressing our heartfelt appreciation for the invaluable support extended to us by numerous individuals. We extend our sincerest gratitude to those whose guidance and encouragement played a pivotal role in bringing this endeavour to fruition.

Foremost, we extend our gratitude to the Chairman, Principal, and Vice Principal for their unwavering support in facilitating the necessary infrastructure for the timely completion of this project.

Our deepest appreciation goes to **Mr. Shaik Akbar**, our esteemed faculty guide and Associate Professor in the CSE(AI&ML) Department at Geethanjali College of Engineering and Technology.

His unwavering support, timely cooperation, and invaluable advice have been instrumental throughout our project's journey.

Dr. A. Nageshwara Rao, the Head of Department, deserves our heartfelt thanks for his meticulous attention and continuous support during the entire project duration.

Additionally, our gratitude extends to all the project Coordinators whose unwavering guidance ensured a smooth progression of our work, overcoming obstacles effortlessly.

To each individual who contributed in ways seen and unseen, your support has been invaluable and has truly made a difference in the successful completion of this project.

ABSTRACT

The project KogniCare – AI-Integrated Patient Monitoring System using Bayesian Reasoning demonstrates how artificial intelligence can enhance modern healthcare through real-time probabilistic diagnosis and uncertainty management. By leveraging Bayesian Networks, the system continuously analyses patient vitals such as heart rate, temperature, and oxygen saturation to estimate the probability of medical conditions like sepsis, heart failure, and respiratory distress. It integrates a Python Flask backend, a Bayesian inference engine, and a responsive web dashboard for real-time visualization and decision support.
The system provides confidence-aware alerts, explainable AI reasoning, and automated medical reports, ensuring accurate, transparent, and proactive patient monitoring. KogniCare showcases how AI-based probabilistic reasoning can improve clinical reliability, reduce false alarms, and assist healthcare professionals in making informed decisions under uncertainty.

Keywords: Artificial Intelligence, Bayesian Networks, Probabilistic Reasoning, Healthcare Monitoring, Flask, Real-time Diagnosis, Uncertainty Management, Explainable AI, Patient Monitoring System, Medical Decision Support

LIST OF FIGURES

TABLE OF CONTENTS

1. INTRODUCTION

In today's healthcare environment, timely and accurate patient monitoring is essential for preventing medical emergencies and improving patient outcomes. Traditional monitoring systems primarily rely on threshold-based alerts, which often result in high false alarm rates and lack the ability to reason under uncertainty. To overcome these limitations, the integration of **Artificial Intelligence (AI)** and **probabilistic reasoning** has emerged as a transformative approach in modern healthcare.

KogniCare is designed to address these challenges by implementing **Bayesian Networks** for probabilistic diagnosis and real-time inference. The system continuously observes patient vitals such as heart rate, oxygen saturation (SpO_2), temperature, and respiratory rate, using them as evidence to calculate the likelihood of conditions like **heart failure**, **sepsis**, and **respiratory distress**.

The project emphasizes **uncertainty management**, which allows the AI to make rational decisions even when medical data is incomplete or noisy—a common scenario in real-world hospital environments. By combining **Flask-based web architecture**, **probabilistic reasoning**, and **interactive visualization**, KogniCare provides an intelligent, explainable, and scalable healthcare solution.

This integration of probabilistic AI models not only enhances diagnostic reliability but also supports **medical professionals** with interpretable insights, enabling faster and more informed clinical decisions. KogniCare thus represents a step toward **data-driven, proactive, and transparent healthcare systems** that can adapt to complex and uncertain patient data in real time.

2. AIM AND OBJECTIVES

2.1 Aim of Proposed Project

The aim of the proposed project **“KogniCare – AI-Integrated Patient Monitoring System using Bayesian Reasoning”** is to design and develop an intelligent healthcare monitoring system that utilizes **probabilistic reasoning** through **Bayesian Networks** to analyze patient health data in real time.

The project seeks to enhance the reliability and efficiency of patient monitoring by:

- Managing **uncertainty** in medical data using probabilistic models.
- Providing **real-time diagnosis** and **risk assessment** for critical conditions.
- Delivering **explainable AI-based recommendations** to assist healthcare professionals in decision-making.
- Minimizing **false alarms** and improving overall **patient safety** through confidence-aware alerts.

By integrating **AI, probabilistic inference, and modern web technologies**, KogniCare aims to create a robust, transparent, and scalable system that supports intelligent healthcare monitoring and proactive medical intervention.

2.2 Objective of Proposed Project

The main objective of the proposed project “KogniCare – AI-Integrated Patient Monitoring System” is to develop an intelligent healthcare monitoring system that applies Bayesian probabilistic reasoning to analyze and interpret patient vital signs in real time.

The system aims to assist healthcare professionals by:

- Managing uncertainty in medical data using Bayesian Networks.
- Performing real-time medical diagnosis and risk assessment.

- Providing explainable AI-based insights for better clinical decision-making.
- Minimizing false alarms through confidence-based alert mechanisms.
- Enhancing patient safety and treatment accuracy through continuous monitoring.

Ultimately, KogniCare strives to bridge the gap between artificial intelligence and healthcare by delivering a transparent, reliable, and data-driven diagnostic support system.

Project Highlights:

- **AI-Powered Healthcare Monitoring:**
  KogniCare integrates artificial intelligence to continuously monitor patient vitals such as heart rate, SpO_2, temperature, and respiratory rate for real-time health assessment.
- **Bayesian Network Implementation:**
  The system employs **Bayesian Networks** for probabilistic reasoning, enabling intelligent diagnosis even when medical data is incomplete or uncertain.
- **Uncertainty Management:**
  By quantifying uncertainty using entropy-based measures, the system provides confidence levels for each diagnosis, improving clinical reliability.
- **Explainable AI (XAI):**
  KogniCare ensures transparency by explaining its diagnostic reasoning, making AI decisions understandable to healthcare professionals.
- **Hybrid AI Integration:**
  Combines **Bayesian reasoning** with a **Large Language Model (Microsoft Phi-3.5 Mini)** for intelligent, conversational medical assistance.
- **Real-time Data Processing:**
  The system performs live inference and updates patient status every few seconds, offering immediate detection of abnormalities.
- **Interactive Web Dashboard:**
  Features a modern, user-friendly interface for visualizing vital signs, probability distributions, and alerts in real time.

- **Confidence-Aware Alerts:**
  Reduces false alarms by using probabilistic thresholds to classify alerts as normal, warning, or critical.
- **Automated Reporting:**
  Generates professional **PDF reports** summarizing patient data, diagnostic probabilities, and uncertainty levels for medical documentation.

3. IMPLEMENTATION

The **KogniCare – AI-Integrated Patient Monitoring System** is implemented using a modular architecture that combines **AI-based probabilistic reasoning**, **real-time data simulation**, and **web-based visualization**. The implementation demonstrates how Bayesian Networks can be used to manage uncertainty in medical diagnosis and assist healthcare professionals with explainable, data-driven insights.

3.1 System Architecture

The project follows a **modular architecture** divided into three primary layers:

1. **Frontend Layer (User Interface):**
   - Developed using **HTML5**, **CSS3**, and **JavaScript**.
   - Displays live vital signs, probability graphs, and alerts.
   - Integrated with **Chart.js** for real-time visualization of Bayesian inference results.
2. **Backend Layer (Core Processing):**
   - Built using **Python Flask** as the web framework.
   - Contains the **Bayesian Network**, **AI services**, and **data processing modules**.
   - Manages patient data, probabilistic computations, and API endpoints.
3. **AI Layer (Probabilistic Reasoning & NLP):**
   - Implements **Bayesian Networks** for probabilistic diagnosis.
   - Integrates **Microsoft Phi-3.5 Mini LLM** via OpenRouter API for explainable medical assistance.
   - Provides uncertainty quantification and reasoning explanations.

3.2 Core Implementation Modules

1. **bayesian_network.py – Bayesian Model:**
   - Implements nodes for **Heart Rate**, **SpO$_2$**, **Temperature**, and **Respiratory Rate**.
   - Defines relationships between these vitals and medical conditions such as **Heart Failure**, **Sepsis**, and **Respiratory Distress**.
   - Uses **Conditional Probability Tables (CPTs)** for probabilistic reasoning and dynamic inference.
2. **vitals_service.py – Vitals Simulation:**
   - Generates realistic, time-varying vital sign data.
   - Simulates real-time patient monitoring with 5–30 second updates.
   - Produces anomalies to trigger alerts and test diagnostic responses.
3. **uncertainty_service.py – Bayesian Analysis Engine:**
   - Performs **probabilistic inference** using Bayesian Networks.
   - Computes **entropy-based uncertainty** and **confidence scores**.
   - Provides the final diagnostic probabilities for visualization.
4. **ai_service.py – LLM Integration:**
   - Connects to **Microsoft Phi-3.5 Mini (via OpenRouter API)**.
   - Handles medical queries and explains Bayesian reasoning in natural language.
   - Offers context-aware responses combining AI reasoning with probabilistic insights.
5. **report_service.py – PDF Report Generator:**
   - Generates professional reports summarizing patient vitals, diagnosis probabilities, and uncertainty levels using **ReportLab**.

6. **app.py – Main Application Controller:**
    - Initializes the Flask app and registers all modules and routes.
    - Connects frontend and backend layers for smooth real-time operation.
    - Hosts API endpoints for vital data, Bayesian results, and AI chat services.

3.3 Frontend Implementation

- The user interface is built using **HTML, CSS, and JavaScript**, featuring:
    - **Real-time Charts** using **Chart.js** to display live vitals and probabilities.
    - **Bayesian Analysis Panel** showing probability distributions and confidence levels.
    - **AI Chat Interface** for user interaction with the medical assistant.
    - **Responsive Design** to ensure accessibility across desktop and mobile devices.

3.4 Execution Workflow

1. **Vitals Simulation:** The system generates continuous vital sign data.
2. **Bayesian Inference:** The data is passed into the Bayesian Network for probabilistic diagnosis.
3. **Uncertainty Estimation:** Confidence scores and entropy values are calculated.
4. **AI Integration:** The LLM interprets results and provides medical explanations.
5. **Visualization:** Results are displayed in real time on the web dashboard.
6. **Reporting:** PDF reports can be generated for patient documentation.

3.5 Output

- **Dashboard Output:**
    - Displays live health metrics, probabilistic diagnoses, and alert statuses.

- **AI Chat Output:**
    - Provides context-aware medical explanations with confidence levels.
- **PDF Report Output:**
    - Summarizes diagnostic probabilities, confidence scores, and trends.

4. SAMPLE CODE

```
import numpy as np
from itertools import product
import math
from typing import Dict, List, Tuple, Any, Optional
import random


class BayesianNode:
    def __init__(self, name: str, states: List[str], parents: List[str] = None):
        self.name = name
        self.states = states
        self.parents = parents or []
        self.cpt = {}  # Conditional Probability Table
    def set_cpt(self, cpt: Dict):
        """Set the conditional probability table for this node"""
        self.cpt = cpt
    def get_probability(self, state: str, parent_states: Dict = None):
        """Get probability P(state | parent_states)"""
        if not self.parents:
            return self.cpt.get(state, 0.0)
        parent_key = tuple(parent_states.get(parent, None) for parent in self.parents)
        return self.cpt.get(parent_key, {}).get(state, 0.0)
```

```
class BayesianNetwork:
    def __init__(self):
        self.nodes = {}
        self.edges = []
    def add_node(self, node: BayesianNode):
        """Add a node to the network"""
        self.nodes[node.name] = node
    def add_edge(self, parent: str, child: str):
        """Add an edge between nodes"""
        if child in self.nodes and parent in self.nodes:
            if parent not in self.nodes[child].parents:
                self.nodes[child].parents.append(parent)
            self.edges.append((parent, child))
    def query(self, query_var: str, evidence: Dict = None) -> Dict[str, float]:
        """Perform exact inference using enumeration"""
        evidence = evidence or {}
        query_node = self.nodes[query_var]
        # Calculate P(query_var = state | evidence) for each state
        probabilities = {}
        for state in query_node.states:
            # Add query variable to evidence
            extended_evidence = evidence.copy()
            extended_evidence[query_var] = state
            # Calculate joint probability
            prob = self._enumerate_all(list(self.nodes.keys()), extended_evidence)
            probabilities[state] = prob
        # Normalize probabilities
        total = sum(probabilities.values())
        if total > 0:
            probabilities = {k: v/total for k, v in probabilities.items()}
        return probabilities
    def _enumerate_all(self, vars_list: List[str], evidence: Dict) -> float:
        """Enumerate all possible assignments to calculate joint probability"""
        if not vars_list:
            return 1.0
        var = vars_list[0]
        rest = vars_list[1:]
        if var in evidence:
            # Variable is assigned in evidence
            prob = self._get_conditional_prob(var, evidence[var], evidence)
            return prob * self._enumerate_all(rest, evidence)
        else:
            # Sum over all possible values
            total = 0.0
            for state in self.nodes[var].states:
                extended_evidence = evidence.copy()
```

```
                extended_evidence[var] = state
                prob = self._get_conditional_prob(var, state, extended_evidence)
                total += prob * self._enumerate_all(rest, extended_evidence)
            return total
    def _get_conditional_prob(self, var: str, state: str, evidence: Dict) -> float:
        """Get P(var = state | parents)"""
        node = self.nodes[var]
        if not node.parents:
            return node.cpt.get(state, 0.0)
        parent_states = {parent: evidence.get(parent) for parent in node.parents}
        return node.get_probability(state, parent_states)
class MedicalBayesianNetwork:
    def __init__(self):
        self.network = BayesianNetwork()
        self.setup_medical_network()
    def setup_medical_network(self):
        """Setup the medical diagnosis Bayesian network"""
        # Define nodes with their possible states
        nodes_config = {
            'heart_rate': ['low', 'normal', 'high'],
            'spo2': ['low', 'normal', 'high'],
            'temperature': ['low', 'normal', 'high'],
            'respiratory_rate': ['low', 'normal', 'high'],
    'heart_failure': ['absent', 'mild', 'severe'],
    'sepsis': ['absent', 'mild', 'severe'],
    'respiratory_distress': ['absent', 'mild', 'severe'],
    'patient_status': ['stable', 'at_risk', 'critical']
}
# Create nodes
for name, states in nodes_config.items():
    if name in ['heart_rate', 'spo2', 'temperature', 'respiratory_rate']:
        # Evidence nodes (no parents)
        node = BayesianNode(name, states)
    elif name in ['heart_failure', 'sepsis', 'respiratory_distress']:
        # Condition nodes (dependent on vital signs)
        parents = ['heart_rate', 'spo2', 'temperature', 'respiratory_rate']
        node = BayesianNode(name, states, parents)
    else:
        # Patient status (dependent on all conditions)
        parents = ['heart_failure', 'sepsis', 'respiratory_distress']
        node = BayesianNode(name, states, parents)
    self.network.add_node(node)
# Add edges
vital_signs = ['heart_rate', 'spo2', 'temperature', 'respiratory_rate']
conditions = ['heart_failure', 'sepsis', 'respiratory_distress']
for vital in vital_signs:
```

```
            for condition in conditions:
                self.network.add_edge(vital, condition)
        for condition in conditions:
            self.network.add_edge(condition, 'patient_status')

        # Set up CPTs (Conditional Probability Tables)
        self._setup_cpts()
    def _setup_cpts(self):
        """Setup conditional probability tables based on medical knowledge"""
        # Prior probabilities for vital signs (evidence nodes)
        self.network.nodes['heart_rate'].set_cpt({
            'low': 0.15,
            'normal': 0.70,
            'high': 0.15
        })
        self.network.nodes['spo2'].set_cpt({
            'low': 0.10,
            'normal': 0.85,
            'high': 0.05
        })
        self.network.nodes['temperature'].set_cpt({
            'low': 0.05,
            'normal': 0.85,
    'high': 0.10
})
self.network.nodes['respiratory_rate'].set_cpt({
    'low': 0.10,
    'normal': 0.80,
    'high': 0.10
})
# Heart Failure CPT (based on heart rate and SpO2 primarily)
heart_failure_cpt = {}
for hr, spo2, temp, rr in product(['low', 'normal', 'high'], repeat=4):
    key = (hr, spo2, temp, rr)
    # Calculate probabilities based on medical knowledge
    if hr == 'high' and spo2 == 'low':
        probs = {'absent': 0.2, 'mild': 0.5, 'severe': 0.3}
    elif hr == 'high' or spo2 == 'low':
        probs = {'absent': 0.4, 'mild': 0.4, 'severe': 0.2}
    elif hr == 'low' and spo2 == 'normal':
        probs = {'absent': 0.6, 'mild': 0.3, 'severe': 0.1}
    else:
        probs = {'absent': 0.8, 'mild': 0.15, 'severe': 0.05}
    heart_failure_cpt[key] = probs
self.network.nodes['heart_failure'].set_cpt(heart_failure_cpt)
```

```
# Sepsis CPT (based on temperature and heart rate primarily)
sepsis_cpt = {}
for hr, spo2, temp, rr in product(['low', 'normal', 'high'], repeat=4):
    key = (hr, spo2, temp, rr)
    if temp == 'high' and hr == 'high':
        probs = {'absent': 0.1, 'mild': 0.4, 'severe': 0.5}
    elif temp == 'high' or hr == 'high':
        probs = {'absent': 0.3, 'mild': 0.5, 'severe': 0.2}
    elif temp == 'low':
        probs = {'absent': 0.4, 'mild': 0.4, 'severe': 0.2}
    else:
        probs = {'absent': 0.85, 'mild': 0.12, 'severe': 0.03}
    sepsis_cpt[key] = probs
self.network.nodes['sepsis'].set_cpt(sepsis_cpt)
# Respiratory Distress CPT
respiratory_cpt = {}
for hr, spo2, temp, rr in product(['low', 'normal', 'high'], repeat=4):
    key = (hr, spo2, temp, rr)
    if spo2 == 'low' and rr == 'high':
        probs = {'absent': 0.1, 'mild': 0.3, 'severe': 0.6}
    elif spo2 == 'low' or rr == 'high':
        probs = {'absent': 0.3, 'mild': 0.5, 'severe': 0.2}
    elif rr == 'low':
        probs = {'absent': 0.6, 'mild': 0.3, 'severe': 0.1}
    else:
        probs = {'absent': 0.9, 'mild': 0.08, 'severe': 0.02}
    respiratory_cpt[key] = probs
self.network.nodes['respiratory_distress'].set_cpt(respiratory_cpt)
```

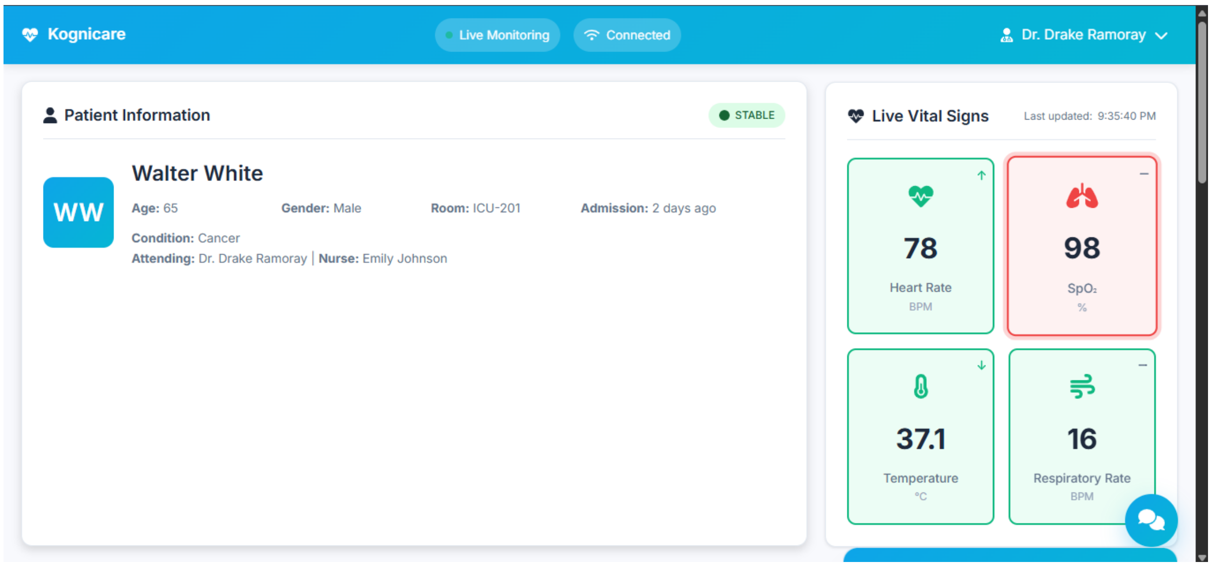
# Patient Status CPT

```
status_cpt = {}
for hf, sepsis, rd in product(['absent', 'mild', 'severe'], repeat=3):
    key = (hf, sepsis, rd)
    severe_count = sum(1 for condition in [hf, sepsis, rd] if condition == 'severe')
    mild_count = sum(1 for condition in [hf, sepsis, rd] if condition == 'mild')
    if severe_count >= 2:
        probs = {'stable': 0.05, 'at_risk': 0.15, 'critical': 0.8}
    elif severe_count == 1:
        probs = {'stable': 0.1, 'at_risk': 0.6, 'critical': 0.3}
    elif mild_count >= 2:
        probs = {'stable': 0.3, 'at_risk': 0.6, 'critical': 0.1}
    elif mild_count == 1:
        probs = {'stable': 0.7, 'at_risk': 0.25, 'critical': 0.05}
    else:
        probs = {'stable': 0.95, 'at_risk': 0.04, 'critical': 0.01}
    status_cpt[key] = probs
```

```
        self.network.nodes['patient_status'].set_cpt(status_cpt)
    def classify_vitals(self, vitals: Dict[str, float]) -> Dict[str, str]:
        """Convert numerical vital signs to categorical states"""
        classifications = {}
        # Heart Rate classification
        hr = vitals.get('heart_rate', 70)
        if hr < 60:
            classifications['heart_rate'] = 'low'
        elif hr > 100:
            classifications['heart_rate'] = 'high'
        else:
            classifications['heart_rate'] = 'normal'
        # SpO2 classification
        spo2 = vitals.get('spo2', 98)
        if spo2 < 95:
            classifications['spo2'] = 'low'
        elif spo2 > 99:
            classifications['spo2'] = 'high'
        else:
            classifications['spo2'] = 'normal'
        # Temperature classification (Celsius)
        temp = vitals.get('temperature', 37.0)
        if temp < 36.0:
            classifications['temperature'] = 'low'
        elif temp > 37.5:
            classifications['temperature'] = 'high'
        else:
            classifications['temperature'] = 'normal'
        # Respiratory Rate classification
        rr = vitals.get('respiratory_rate', 16)
        if rr < 12:
            classifications['respiratory_rate'] = 'low'
        elif rr > 20:
            classifications['respiratory_rate'] = 'high'
        else:
            classifications['respiratory_rate'] = 'normal'
        return classifications
    def update_with_vitals(self, vitals: Dict[str, float]) -> Dict[str, Any]:
        """Update the network with new vital signs and return inference results"""
        # Classify vitals into categorical states
        evidence = self.classify_vitals(vitals)
        # Perform inference for all conditions and patient status
        results = {}
        # Query each condition
        conditions = ['heart_failure', 'sepsis', 'respiratory_distress', 'patient_status']
        for condition in conditions:
            results[condition] = self.network.query(condition, evidence)
        # Add uncertainty analysis
        results['uncertainty_analysis'] = self._analyze_uncertainty(results)
        # Add evidence used
        results['evidence'] = evidence
        results['raw_vitals'] = vitals
        return results
    def _analyze_uncertainty(self, results: Dict) -> Dict[str, Any]:
        """Analyze uncertainty in the predictions"""
        uncertainty_metrics = {}
        for condition, probabilities in results.items():
            if isinstance(probabilities, dict):
                # Calculate entropy (uncertainty measure)
                entropy = -sum(p * math.log2(p) if p > 0 else 0
                               for p in probabilities.values())
                # Calculate confidence (max probability)
                max_prob = max(probabilities.values())
                # Calculate certainty (1 - entropy/max_entropy)
                max_entropy = math.log2(len(probabilities))
                certainty = 1 - (entropy / max_entropy) if max_entropy > 0 else 1
                uncertainty_metrics[condition] = {
                    'entropy': entropy,
            'confidence': max_prob,
            'certainty': certainty,
            'most_likely': max(probabilities.items(), key=lambda x: x[1])
        }
    return uncertainty_metrics
def get_recommendations(self, analysis_results: Dict) -> List[str]:
    """Generate clinical recommendations based on Bayesian analysis"""
    recommendations = []
    # Check patient status
    status_probs = analysis_results.get('patient_status', {})
    critical_prob = status_probs.get('critical', 0)
    at_risk_prob = status_probs.get('at_risk', 0)
    if critical_prob > 0.5:
        recommendations.append(" IMMEDIATE medical intervention required")
        recommendations.append("Consider ICU admission")
    elif at_risk_prob > 0.4:
        recommendations.append(" Increase monitoring frequency")
        recommendations.append("Prepare for potential deterioration")
    # Check specific conditions
    hf_probs = analysis_results.get('heart_failure', {})
    if hf_probs.get('severe', 0) > 0.3:
        recommendations.append(" Consider echocardiogram and BNP testing")
        recommendations.append("Monitor fluid balance closely")
    sepsis_probs = analysis_results.get('sepsis', {})
    if sepsis_probs.get('severe', 0) > 0.3:
        recommendations.append(" Initiate sepsis protocol immediately")
        recommendations.append("Obtain blood cultures and start antibiotics")
    rd_probs = analysis_results.get('respiratory_distress', {})
    if rd_probs.get('severe', 0) > 0.3:
        recommendations.append(" Consider arterial blood gas analysis")
        recommendations.append("Assess need for respiratory support")
    # Uncertainty-based recommendations
    uncertainty = analysis_results.get('uncertainty_analysis', {})
    high_uncertainty_conditions = [
        condition for condition, metrics in uncertainty.items()
        if isinstance(metrics, dict) and metrics.get('certainty', 1) < 0.6
    ]
    if high_uncertainty_conditions:
        recommendations.append(f" High uncertainty detected in: {',
'.join(high_uncertainty_conditions)}")
        recommendations.append("Consider additional diagnostic tests")
    if not recommendations:
        recommendations.append(" Continue current monitoring protocol")
        recommendations.append("Patient appears stable based on current data")
    return recommendations
```

```
# Example usage and testing
if __name__ == "__main__":
    # Create the medical Bayesian network
    medical_net = MedicalBayesianNetwork()
    # Example patient vitals
    test_vitals = {
        'heart_rate': 110,      # High
        'spo2': 92,             # Low
        'temperature': 38.5,    # High
        'respiratory_rate': 25  # High
    }
    print(" Medical Bayesian Network Analysis")
    print("=" * 50)
    # Analyze the patient
    results = medical_net.update_with_vitals(test_vitals)
    print(f"\n Patient Vitals:")
    for vital, value in test_vitals.items():
        classification = results['evidence'][vital]
        print(f"  • {vital.replace('_', ' ').title()}: {value} ({classification})")
    print(f"\n Bayesian Analysis Results:")
    conditions = ['heart_failure', 'sepsis', 'respiratory_distress', 'patient_status']
    for condition in conditions:
        probs = results[condition]
        uncertainty = results['uncertainty_analysis'][condition]
print(f"\n{condition.replace('_', ' ').title()}:")

        for state, prob in probs.items():

            print(f"  • {state.capitalize()}: {prob:.3f} ({prob*100:.1f}%)")

        print(f"  Confidence: {uncertainty['confidence']:.3f}")

        print(f"  Certainty: {uncertainty['certainty']:.3f}")

        print(f"  Most Likely: {uncertainty['most_likely'][0]}
({uncertainty['most_likely'][1]:.3f})")

    print(f"\n Clinical Recommendations:")

    recommendations = medical_net.get_recommendations(results)

    for i, rec in enumerate(recommendations, 1):

        print(f"  {i}. {rec}")

    print(f"\n Summary:")

    status_result = results['uncertainty_analysis']['patient_status']['most_likely']

    print(f"Patient Status: {status_result[0].upper()} (confidence: {status_result[1]:.3f})")
```

5. OUTPUT SCREENS


Kognicare
Live Monitoring
Connected
Dr. Drake Ramoray

Patient Information
STABLE

WW
Walter White
Age: 65 **Gender:** Male **Room:** ICU-201 **Admission:** 2 days ago
Condition: Cancer
Attending: Dr. Drake Ramoray | **Nurse:** Emily Johnson

Live Vital Signs
Last updated: 9:35:40 PM

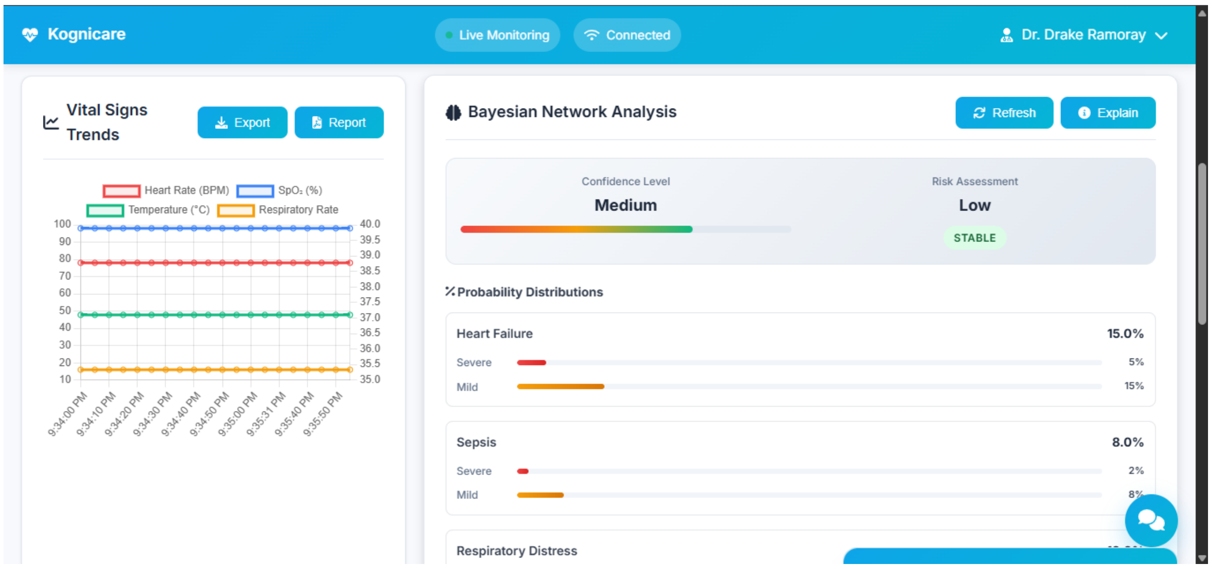
78
Heart Rate
BPM

98
SpO₂
%

37.1
Temperature
°C

16
Respiratory Rate
BPM

Figure 5.1 – Patient Information and Live Vital Signs


Kognicare
Live Monitoring
Connected
Dr. Drake Ramoray

Vital Signs Trends
Export
Report

Bayesian Network Analysis
Refresh
Explain

Confidence Level
Medium

Risk Assessment
Low
STABLE

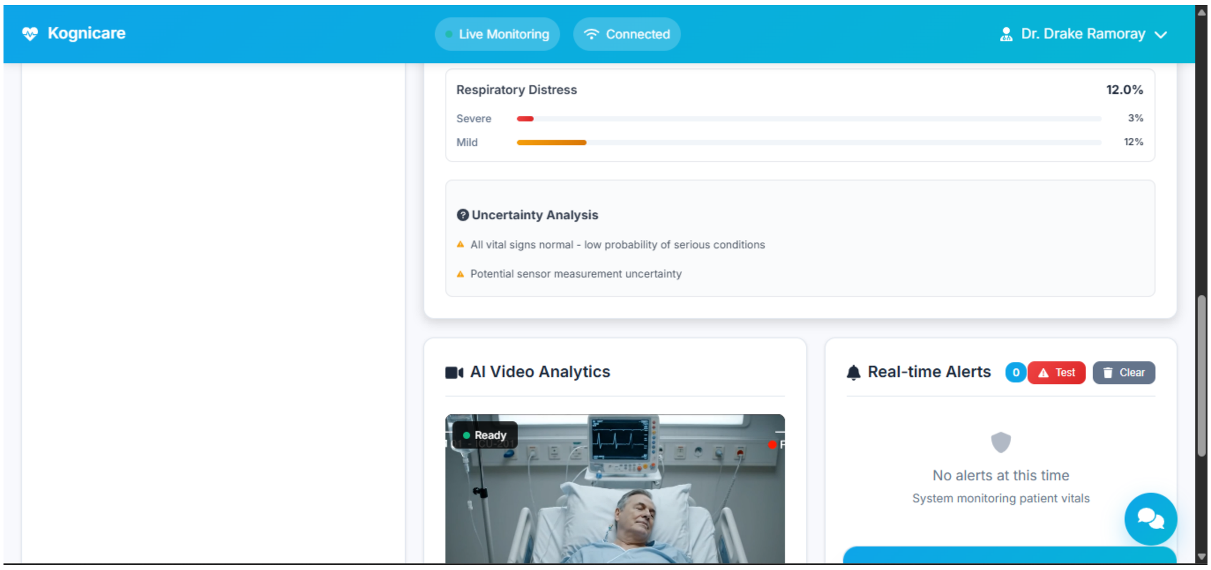
Probability Distributions

Heart Failure 15.0%
Severe 5%
Mild 15%

Sepsis 8.0%
Severe 2%
Mild 8%

Respiratory Distress

Figure 5.2 – Vital Signs Trends and Bayesian Network Analysis



Kognicare

Live Monitoring | Connected | Dr. Drake Ramoray

Respiratory Distress 12.0%

Severe 3%

Mild 12%

Uncertainty Analysis

All vital signs normal - low probability of serious conditions

Potential sensor measurement uncertainty

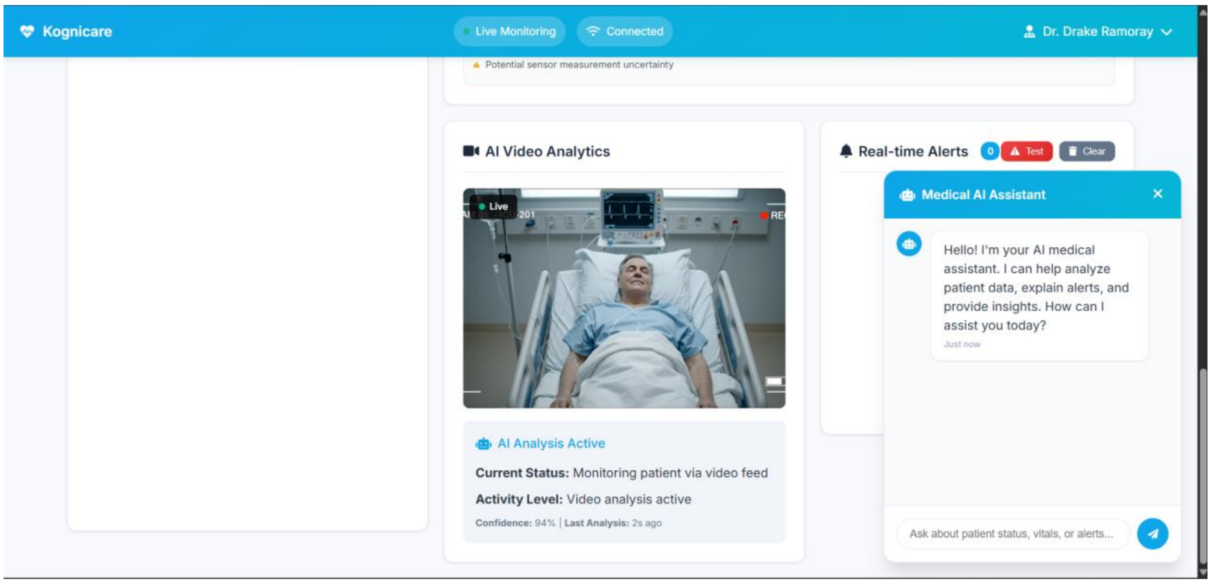
AI Video Analytics

Ready

Real-time Alerts 0 Test Clear

No alerts at this time

System monitoring patient vitals



Kognicare

Live Monitoring | Connected | Dr. Drake Ramoray

Potential sensor measurement uncertainty

AI Video Analytics

Live

AI Analysis Active

Current Status: Monitoring patient via video feed

Activity Level: Video analysis active

Confidence: 94% | **Last Analysis:** 2s ago

Real-time Alerts 0 Test Clear

Medical AI Assistant

Hello! I'm your AI medical assistant. I can help analyze patient data, explain alerts, and provide insights. How can I assist you today?

Just now

Ask about patient status, vitals, or alerts...

Figure 5.3 and 5.4 – AI Video Analytics and Real Time Alerts

6. CONCLUSION

The project **“KogniCare – AI-Integrated Patient Monitoring System”** successfully demonstrates how **Artificial Intelligence** and **Bayesian probabilistic reasoning** can be applied to healthcare monitoring for intelligent, real-time medical diagnosis under uncertainty.

Through the integration of **Bayesian Networks**, **uncertainty quantification**, and **LLM-based medical assistance**, the system effectively manages incomplete or noisy data and provides explainable, confidence-aware recommendations. This enhances the reliability of medical decision-making while minimizing false alarms and improving patient safety.

The project also emphasizes **transparency**, **scalability**, and **academic value**, serving as a practical demonstration of probabilistic reasoning and uncertainty management in AI-driven healthcare systems.

Overall, **KogniCare** lays a strong foundation for future advancements in **real-time patient monitoring**, **AI-assisted diagnosis**, and **intelligent healthcare technologies**.

7. BIBLIOGRAPHY

1. **Judea Pearl**, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, 1988.
2. **Koller, Daphne & Friedman, Nir**, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, 2009.
3. **Russell, Stuart & Norvig, Peter**, *Artificial Intelligence: A Modern Approach*, 4th Edition, Pearson Education, 2020.
4. **Heckerman, David**, "A Tutorial on Learning with Bayesian Networks," *Microsoft Research Technical Report MSR-TR-95-06*, 1995.
5. **Zhou, X., et al.**, "Bayesian Networks in Medical Applications," *Artificial Intelligence in Medicine*, Elsevier, 2014.
6. **World Health Organization (WHO)** – "Artificial Intelligence in Health: Ethics and Governance," WHO Guidance, 2021.
7. **OpenAI & Microsoft Research**, "Large Language Models for Healthcare Decision Support," *AI in Medicine Journal*, 2023.
8. **Flask Documentation**, https://flask.palletsprojects.com/
9. **Chart.js Documentation**, https://www.chartjs.org/
10. **ReportLab User Guide**, https://www.reportlab.com/docs/reportlab-userguide.pdf