Design of AI Enabled Diseases Diagnosis Expert System Using Clipspy

BY

Chandrakant Majumdar
(Admission No. 23MT0113)

Dissertation

SUBMITTED TO
INDIAN INSTITUTE OF TECHNOLOGY
(INDIAN SCHOOL OF MINES), DHANBAD

For the award of the degree of

Master of Technology

MAY 2025

कम्पयूटर विज्ञान एवं अभियांत्रिकी विभाग
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
भारतीय प्रौद्योगिकी संस्थान (भारतीय खनि विद्यापीठ), धनबाद
धनबाद-826004, झारखण्ड, भारत
INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD
DHANBAD-826004, JHARKHAND, INDIA

CERTIFICATE

This is to certify that the Dissertation entitled **"Design of AI Enabled Diseases Diagnosis Expert System Using Clipspy"**, being submitted Indian Institute of Technology(Indian School of Mines), Dhanbad, by Mr **Chandrakant Majumdar**, Admission No **23MT0113** for the award of Degree of **Master of Technology** from IIT(ISM), Dhanbad, is a bonafide work carried out by him, in the Department of Computer Science and Engineering, IIT(ISM), Dhanbad, under my supervision and guidance. The dissertation has fullfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission. The results embodied in this dissertation have not been submitted to any other university or institute for the award of any degree or diploma.

Prof. Chiranjeev Kumar
Head of Department
Department of Computer Science and Engineering
Indian Institute of Technology(ISM) Dhanbad

Prof. Haider Banka
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology(ISM) Dhanbad

DECLARATION

I hereby declare that the work which is being presented in this dissertation entitled **"Design of AI Enabled Diseases Diagnosis Expert System Using Clipspy"** in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Computer Science and Engineering** is an authentic record of my own work carried out during the period from **July 2024 to May 2025** under the supervision of **Prof. Haider Banka**, Department of **Computer Science and Engineering**, Indian Institute of Technology (Indian School of Mines), Dhanbad, Jharkhand, India.

I acknowledge that I have read and understood the UGC (Promotion of Academic Integrity and Prevention of Plagiarism in Higher Educational Institutions) Regulations, 2018. These Regulations were published in the Indian Official Gazette on 31st July, 2018.

I confirm that this Dissertation has been checked for plagiarism using the online plagiarism checking software provided by the Institute. At the end of the Dissertation, a copy of the summary report demonstrating similarities in content and its potential source (if any) generated online using plagiarism checking software is enclosed. I herewith confirm that the Dissertation has less than 10% similarity according to the plagiarism checking software's report and meets the MoE/UGC Regulations as well as the Institute's rules for plagiarism.

I further declare that no portion of the dissertation or its data will be published without the Institute's or Guide's permission. I have not previously applied for any other degree or award using the topics and findings described in my dissertation.

Signature of the Student

Name: **Chandrakant Majumdar**
Admission No: **23MT0113**
Department: **Computer Science and Engineering**

कम्पयूटर विज्ञान एवं अभियांत्रिकी विभाग
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
भारतीय प्रौद्योगिकी संस्थान (भारतीय खनि विद्यापीठ), धनबाद
धनबाद-826004, झारखण्ड, भारत
INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD
DHANBAD-826004, JHARKHAND, INDIA

CERTIFICATE FOR CLASSIFIED DATA

This is to certify that the Dissertation entitled **Design of AI Enabled Diseases Diagnosis Expert System Using Clipspy** being submitted to the Indian Institute of Technology (Indian School of Mines), Dhanbad by **Mr Chandrakant Majumdar** for award of Master Degree in **Computer Science and Technology** does not contain any classified information. This work is original and yet not been submitted to any institution or university for the award of any degree.

Signature of the guide(s)

Signature of the Student

Phone: +91-326-2235273. Fax: +91-326-2296563. Website:
https://www.iitism.ac.in/index.php/computer_science

कम्पयूटर विज्ञान एवं अभियांत्रिकी विभाग
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
भारतीय प्रौद्योगिकी संस्थान (भारतीय खनि विद्यापीठ), धनबाद
धनबाद-826004, झारखण्ड, भारत
INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD
DHANBAD-826004, JHARKHAND, INDIA

CERTIFICATE REGARDING ENGLISH CHECKING

This is to certify that the Dissertation entitled **"Design of AI Enabled Diseases Diagnosis Expert System Using Clipspy"** being submitted to the Indian Institute of Technology (Indian School of Mines), Dhanbad by **Mr Chandrakant Majumdar** Admission No **23MT0113**, for the award of Master of Technology has been thoroughly checked for quality of English and logical sequencing of topics.

It is hereby certified that the standard of English is good and that grammar and typos have been thoroughly checked.

Signature of Guide(s)
Name: Prof. Haider Banka
Date: 02 May 2025

Signature of Student
Name: Chandrakant Majumdar
Date: 02 May 2025

कम्पयूटर विज्ञान एवं अभियांत्रिकी विभाग
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
भारतीय प्रौद्योगिकी संस्थान (भारतीय खनि विद्यापीठ), धनबाद
धनबाद-826004, झारखण्ड, भारत
INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES), DHANBAD
DHANBAD-826004, JHARKHAND, INDIA

COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IIT (ISM), Dhanbad and must accompany any such material in order to be published by the ISM. Please read the form carefully and keep a copy for your files.

TITLE OF DISSERTATION: Design of AI Enabled Diseases Diagnosis Expert System Using Clipspy

AUTHOR'S NAME AND ADDRESS:

CHANDRAKANT MAJUMDAR
Netaji Nagar,
Bongaon,
West Bengal – 743235

COPYRIGHT TRANSFER

1. The undersigned hereby assigns to Indian Institute of Technology (Indian School of Mines), Dhanbad all rights under copyright that may exist in and to: (a) the above Work, including any revised or expanded derivative works submitted to the ISM by the undersigned based on the work; and (b) any associated written or multimedia components or other enhancements accompanying the work.

CONSENT AND RELEASE

2. In the event the undersigned makes a presentation based upon the work at a conference hosted or sponsored in whole or in part by the IIT (ISM) Dhanbad, the undersigned, in consideration for his/her participation in the conference, hereby grants the ISM the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record,

digitize, broadcast, reproduce and archive; in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IIT(ISM) Dhanbad and live or recorded broadcast of the Presentation during or after the conference.

3. In connection with the permission granted in Section 2, the undersigned hereby grants IIT (ISM) Dhanbad the unlimited,worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IIT (ISM) Dhanbad from any claim based on right of privacy or publicity.

4. The undersigned hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the undersigned has obtained any necessary permissions. Where necessary, the undersigned has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IIT (ISM) Dhanbad.

GENERAL TERMS

- The undersigned represents that he/she has the power and authority to make and execute this assignment.
- The undersigned agrees to indemnify and hold harmless the IIT (ISM) Dhanbad from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
- In the event the above work is not accepted and published by the IIT (ISM) Dhanbad or is withdrawn by the author(s) before acceptance by the IIT(ISM) Dhanbad, the foregoing copyright transfer shall become null and void and all materials embodying the Work submitted to the IIT(ISM) Dhanbad will be destroyed.
- For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.

Signature of the Student

ACKNOWLEDGEMENTS

It is indeed a great pleasure to express my sincere thanks to my guide **Prof. Haider Banka**, Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology (Indian School of Mines), Dhanbad, for his continuous support and guidance in this thesis. He was always there to listen and encourage me. He showed me different ways to approach a research problem and the need to be persistent to accomplish my goal. He inspired me to think through the problems and generate new idea. I will always be grateful to him for his valuable supervision and guidance.

I would like to thank **Prof. Chiranjeev Kumar**(Head of Department) and all the faculties of Computer Science and engineering, Indian Institute of Technology (Indian School of Mines), Dhanbad, for their valuable support and suggestions towards the research work.

I am grateful to my family members and friends for their constant support and motivation.

Last, but not the least, I want to thank my parents, for giving me the life in the first place and for educating me. I would like to thank my family for their unconditional support and encouragement to pursue my interest. It is a pleasure to express my deepest gratitude to all those who inspired me for the successful completion of the project.

Chandrakant Majumdar
23MT0113
M.Tech. CSE
IIT ISM, Dhanbad
Date: 02 May 2025

ABSTRACT

This research introduces a smart diagnostic system designed to assist in the early identification of diseases by using a rule-based artificial intelligence engine. At the heart of the system is CLIPS (C Language Integrated Production System), which operates as the inference engine, enabling decision-making based on encoded medical knowledge. To make the tool accessible to users, the backend is paired with a modern web interface developed with React and Vite, creating an intuitive environment where individuals can input symptoms and receive initial diagnostic suggestions.

A significant advantage of the system is its offline capability, making it particularly useful in rural or underserved regions where internet access and medical professionals are scarce. The diagnostic process relies on a structured and evolving database of symptoms and disease profiles, translated into inference rules. This allows the engine to analyze user input and deliver context-sensitive, reliable assessments efficiently.

To improve interaction, the platform includes a conversational AI chatbot that guides users through symptom entry using natural language, answering follow-up questions and simplifying the user experience. Early tests show that the system can provide dependable preliminary evaluations and supports user-friendly operation.

Furthermore, the inclusion of dynamic data integration enables the system to adapt and refine its diagnostic accuracy over time. This project contributes to the ongoing efforts to bridge healthcare gaps, offering a practical, scalable solution for communities with limited healthcare infrastructure. Overall, the system demonstrates a promising approach to expanding access to early medical guidance and enhancing public health outcomes in low-resource settings.

Keywords: CLIPS, Artificial Intelligence (AI), Expert System, Medical Chatbot, Rule-Based Inference, Offline Diagnostics, React, Vite

Contents

List of Figures

List of Abbreviations

AI	Artificial Intelligence
NLP	Natural Language Processing
CLIPS	C Language Integrated Production System
UI	User Interface
CPU	Central Processing Unit
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
PWA	Progressive Web Application
QA	Question Answering

Chapter 1

Introduction

1.1 Background

Disease diagnosis expert systems have long served healthcare professionals by offering accurate and timely diagnostic recommendations. These systems often function through predefined symptom lists, which limit the depth and flexibility of user interaction and may not fully capture the nuances of user-described symptoms. This research introduces an enhanced expert system using **Clipspy**, aimed at improving both diagnostic accuracy and user experience through dynamic interaction.

Artificial Intelligence (AI) has significantly transformed the healthcare sector, especially in the domain of diagnostics and patient care. Early AI-driven expert systems such as *MYCIN* demonstrated promising capabilities by simulating expert-level reasoning using structured rule sets. Despite these successes, most traditional systems remain rigid—relying on static knowledge bases and requiring structured input formats—making them less adaptive to complex and evolving clinical scenarios.

Clipspy, a Pythonic interface for the CLIPS rule-based expert system, offers a powerful alternative. By enabling a more interactive and flexible model of user input, Clipspy allows natural symptom descriptions rather than checklist-style selection. This adaptation not only improves user engagement but also enhances the system's ability to reason with diverse, context-rich input data.

Moreover, Clipspy supports dynamic updates to its rule and knowledge base, enabling continuous incorporation of the latest clinical research. This real-time adaptability ensures the system remains relevant and reliable, maintaining diagnostic precision even in rapidly evolving medical contexts.

1.2 Motivation

The motivation for this project stems from observed shortcomings in current expert systems, particularly those related to rigid input structures and lack of contextual interpretation. These systems often fail to accommodate users who may describe symptoms in layperson language or who reside in low-connectivity areas.

In many rural or under-resourced settings, where professional medical consultation is sparse, intelligent expert systems could serve as critical support tools. However, the majority of current systems do not function offline, making them inaccessible in areas with limited or no internet infrastructure.

Furthermore, the field lacks flexible AI solutions capable of dynamically clarifying ambiguous user input through follow-up questioning. The proposed Clipspy-based system will bridge this gap by integrating advanced view and filter mechanisms to support iterative querying for better symptom refinement.

By allowing a richer form of dialogue and retaining local execution capabilities, this system aims to deliver inclusive and privacy-conscious diagnostic support to a broader population.

1.3 Objective

The research establishes a rule-based diagnostic system with Clipspy to provide medical precision through an interface designed for easy user interaction.

The objectives of this research are as follows:

- **Design a diagnostic chatbot using Clipspy:** The system will combine ClipSpy and natural symptom descriptions with a rule engine for precise medical diagnosis.
- **Develop a flexible and updatable rule base:** The diagnostic rule base needs to be designed as an evolving system capable of receiving regular updates which improve its diagnostic capabilities.
- **Enable offline functionality:** The system needs to operate offline to support deployment in areas with no internet access for remote and rural use.

- **Implement interactive symptom clarification:** The system should request additional symptoms through an interactive process to improve diagnostic precision.
- **Create a responsive and intuitive user interface:** A responsive system needs an intuitive user interface that will assist non-technical users in symptom entry and diagnosis retrieval functions.

1.4 Contributions

The research project delivers various technical and real-world contributions to AI-driven expert system development.

- **Enhanced Rule-Based System:** Through Clipspy the system implements an advanced rule-based network that processes medical structures and vague user queries efficiently.
- **Offline Diagnostic Capability:** The system ensures healthcare equity through its diagnostic ability to operate without an internet connection which enables service delivery to users in areas without adequate connectivity.
- **Dynamic Knowledge Management:** The system includes features for maintaining dynamic knowledge control mechanisms that ensure the system updates with recent medical standards and research findings.
- **Proof-of-Concept Evaluation:** Demonstrates the system's feasibility through real-world medical use cases and scenario-based testing.

1.5 Organization of the Thesis

The thesis is structured across multiple chapters to reflect the systematic development and evaluation of the system.

- **Chapter 2 – Literature Review:** Provides an in-depth analysis of previous works in medical expert systems, including CLIPS-based systems and AI-driven diagnostics.

- **Chapter 3 – Problem Statement:** Defines the gaps and limitations of current technologies, thereby justifying the research approach.
- **Chapter 4 – Research Objectives:** Details the goals, hypothesis, and motivations behind the solution design.
- **Chapter 5 – Preliminaries:** Introduces foundational topics including Clipspy, rule-based reasoning, and AI in healthcare.
- **Chapter 6 – Proposed Methodology:** Describes the technical design, system architecture, and functional modules used in implementation.
- **Chapter 7 – System Implementation:** Provides a detailed explanation of the system build process, code-level insights, and tools used.
- **Chapter 8 – Results and Evaluation:** Presents the results of system testing, accuracy metrics, and analysis of diagnostic output.
- **Chapter 9 – Conclusion and Future Work:** Summarizes the project's accomplishments and outlines potential future enhancements.

Chapter 2

Literature Survey

Artificial intelligence integration with healthcare transforms how medical services deliver their operations. The innovative development of expert systems alongside healthcare chatbots stands as a promising technology because they deliver fast and dependable medical diagnosis solutions together with initial medical direction. A review of literature shows different expert system deployments which employ CLIPS-based rule-based engines by looking into their clinical advantages together with their technical boundaries and practice implications.

An eye disease diagnosis expert system was developed by Naser and Zaiter [1] through implementation of CLIPS rule-based methodology. An evaluation system which analyzes patient symptoms leads to diagnosis of various ophthalmologic conditions demonstrates CLIPS viability in medical diagnostics though practical limitations restrict its clinical use across diverse settings.

The expert system developed by Aldaour and Abu-Naser [2] diagnoses tobacco-related diseases through CLIPS programming and generates treatment solutions at the same time. The system demonstrates practical operation yet it restricts itself to examining tobacco-related illnesses thereby limiting its applicability in healthcare areas beyond those related to smoking diseases.

The research work by Abu and collaborators constructed an expert system designed to diagnose uveitis disease. By systematically organizing symptom-based rules within the CLIPS environment, their work highlights the potential for supporting ophthalmologists in early-stage disease identification, though broader clinical testing remains necessary.

The theoretical foundation provided by Durkin and Durkin [3] offers critical in-

sights into the design and development of expert systems, including rule-based architectures, inference mechanisms, and knowledge acquisition techniques. Although some methodologies discussed may now be considered classical, the book remains a fundamental reference for CLIPS-based system implementations.

Expanding the domain further, Masri et al. [4] conducted a comprehensive survey of rule-based systems, detailing the architecture and operational mechanics of expert systems powered by inference engines like CLIPS. Their findings reaffirm the continued relevance of rule-based frameworks, particularly in knowledge-intensive domains like medical diagnosis.

An expert system for medical diagnosis which combined symptom mapping with modular rule organization was designed by Nkuma-Udah et al. [5]. An initial evaluation confirmed that the developed medical diagnostic system proved effective which validates expert system capabilities for initial diagnostic processes.

The research of Al-Ajmi and Almulla [6] introduced an expert system made of rules to identify headache types and provide suitable medication recommendations. A CLIPS-based structure enabled the system to demonstrate how diagnosis that starts with inference could support medical practitioners in effective decision-making and prescription creation.

The development of heart palpitation diagnostic expert system using CLIPS logic was presented by Qanoo et al. [7]. Traditional rule-based logic combined with user-friendly interfaces enhances the functional usability of diagnostic artificial intelligence systems according to their research while demonstrating the requirement of intelligent systems that cater to end-users' accessibility needs.

Chapter 3

Problem Statement

The current medical expert systems build their operations primarily from allopathic knowledge and need structured data processing through online centralization. Symptom-based assessment systems through both rule-based systems and statistical learning models deliver diagnostic recommendations to medical professionals. Medical expert systems encounter multiple performance restrictions that restrict their use in different healthcare environments.

The burgeoning healthcare system faces a critical problem because Treatment centers along with hospitals continue to operate beyond their capacity limits because they must handle rising numbers of patients who do not need immediate care. Healthcare facilities handle large numbers of non-urgent cases from patients resulting in stretched resources while emergency medical staff remain unavailable. The healthcare setting demands new preliminary screening technologies which help medical staff perform patient triage operations and limit unnecessary hospital admissions.

Expert systems commonly use restricted user interfaces that require prescribed symptom data entry methods which cannot capture the various natural patient symptom descriptions. Patients show reduced commitment and make inferior clinical assessments when facing interfaces that are complicated to access or control. A diagnostic interface should contain a user-friendly system that enables smooth communication between clients and the diagnostic process.

A conventional expert system contains static rules obtained through manual en-

coding yet lacks any system maintenance process. Such static systems require human involvement for knowledge updates and symptom-disease pattern adaptation because they lack automatic learning capabilities. The enduring value of these systems decreases because of their reduced reliability and relevance when they are not properly maintained nor revised.

Online Dependency and Limited Accessibility: Most AI-driven diagnosis platforms require continuous internet connectivity to operate, whether for accessing medical databases or running cloud-hosted models. This reliance excludes patients in rural or infrastructure-poor regions from using such systems effectively. Offline functionality is critical to ensure healthcare accessibility for diverse populations.

This project focuses on the development of a dedicated homeopathic medical chatbot system that resolves these issues through a localized, rule-based architecture using CLIPS. The chatbot is capable of functioning completely offline, providing reliable preliminary diagnostics within the domain of homeopathy. It ensures that users who prefer or depend on alternative medicine have access to accurate and ethically structured medical support. By designing a system that is user-friendly, privacy-respecting, and context-aware, this research addresses a significant gap in AI-assisted healthcare for underserved and alternative medicine-focused populations.

Chapter 4

Research Objectives

This project aims to develop a medical diagnosis chatbot system focused on homeopathic treatment suggestions, powered by a rule-based inference engine using CLIPS. The system is designed for offline operation and includes a modern web-based interface for users to enter symptoms and receive homeopathy-aligned advice. This research supports the increasing demand for accessible, privacy-conscious and alternative-medicine-compatible digital healthcare tools.

The primary objectives of the project are:

1. Developing a Rule-Based Expert System: Design and implement a comprehensive rule base using the CLIPS engine for disease diagnosis and treatment recommendations. This includes:

- Structuring conditional logic to map symptoms to homeopathic diagnoses.
- Incorporating mechanisms for dynamic fact assertion and rule activation.
- Ensure inference accuracy using pattern matching techniques within CLIPS.

2. Architecting a Scalable Knowledge Base: Establish a modular and extensible knowledge base capable of adapting to new diseases, symptoms, and remedies. This includes:

- Organizing hierarchical relationships between symptoms and diseases.
- Creating reusable templates for efficient rule and fact definitions.

- Supporting future integration of more complex diagnostic parameters.

3. Ensuring Offline Accessibility: Build the system for fully offline use to enable accessibility in regions with poor internet connectivity while ensuring user privacy. This objective involves:

- Performing all computations locally without reliance on cloud services.
- Storing symptom history and diagnostic interactions securely on-device.
- Supporting low-resource hardware without compromising performance.

4. Designing a User-Friendly Web Interface: Develop a responsive and accessible frontend using Vite and React to enable seamless user interaction. Key goals include:

- Providing intuitive symptom entry and diagnosis feedback components.
- Supporting clear visual output and navigation on various screen sizes.
- Reducing the learning curve for users of all technological backgrounds.

By achieving these goals, the system will offer a functional and inclusive solution for digital homeopathic consultations. The integration of rule-based AI with user-centric design promotes efficient healthcare access and supports the growing need for non-allopathic diagnostic tools, particularly in underserved or infrastructure-limited regions.

Chapter 5

Preliminaries

Construction of the recommended AI-based disease diagnosis system adopts essential artificial intelligence principles together with contemporary web development techniques. The system design incorporates several fundamental components that include CLIPS as a rule-based inference engine and uses Python technology for backend operations and JavaScript and an offline-first architecture for system functionality. These components function together to create an expert system that provides real-time preliminary medical guidance along with scalable operation and privacy awareness.

5.1 Rule-Based Systems and the Role of CLIPS

Rule-based expert systems apply logical rules to known facts to derive conclusions. These systems offer transparency, traceability, and ease of debugging—qualities that are critical in healthcare contexts.

5.1.1 Overview of CLIPS

CLIPS (C Language Integrated Production System) is a widely adopted rule-based programming environment that constitutes 27.2% of our backend codebase. It supports forward-chaining inference, efficient pattern matching, and dynamic knowledge representation. Key features of CLIPS include:

- **Facts**: Represent observed symptoms or patient inputs.
- **Rules**: Encode conditional logic for diagnosing diseases.

5.1.2 Advantages of CLIPS for Diagnosis

CLIPS is well-suited for medical diagnostic applications because it:

- Allows real-time rule evaluation and fact updates.
- Provides deterministic behavior essential for critical healthcare use cases.
- Operates efficiently without relying on cloud infrastructure.

5.2 System Architecture

The expert system implements a client-server architecture with distinct frontend and backend components:

5.2.1 Python-Based Backend

Python forms the backbone of our system (61.3% of backend codebase), providing:

- **CLIPS Integration**: Python interfaces with the CLIPS engine through appropriate bindings.
- **API Layer**: RESTful endpoints that process requests from the frontend.
- **Business Logic**: Handling of diagnostic workflows and rule management.
- **Deployment Configuration**: Includes Procfile for cloud deployment on platforms like Heroku.

5.2.2 JavaScript-Based Frontend

The frontend is predominantly JavaScript-based (95.6% of frontend codebase):

- **React Components**: Modular interface elements for collecting symptom data and displaying results.
- **State Management**: Tracking user inputs and diagnostic sessions.
- **API Integration**: Communication with the Python backend.
- **CSS Styling**: Responsive design ensuring usability across different devices (2.5% of frontend).

5.3 Cross-Platform Communication

The system relies on efficient communication between components:

- **Backend-to-CLIPS**: Python code interfaces with the CLIPS engine to process rules and facts.
- **Frontend-to-Backend**: RESTful API calls transmit user inputs and retrieve diagnostic results.
- **Local Storage**: Browser storage maintains session persistence for offline capability.

5.4 Offline Capability Design

Offline functionality is a critical feature of the system, especially for deployment in underserved areas. This is achieved through:

- **Progressive Web Application (PWA) Features**: Allowing the application to be installed locally.
- **Local Rule Processing**: Enabling diagnosis without continuous internet connectivity.
- **Data Persistence**: Storing symptom inputs and diagnosis history locally.

5.5 Privacy and Accessibility Considerations

The system is designed with privacy-first principles:

- **Minimized Data Transmission**: Processing occurs locally where possible to protect user privacy.
- **Data Sovereignty**: Users retain control over their diagnostic data and history.
- **Minimal System Requirements**: The system runs on standard hardware with 8–16GB RAM and no GPU, enabling deployment on household, educational, and rural health devices.

5.6 Knowledge Base Structure

The diagnostic engine relies on a structured and extensible knowledge base encoded using CLIPS:

- **Symptoms as Facts**: Standardized templates represent each symptom.
- **Rules for Diseases**: Each disease is associated with a pattern of symptoms and rule-based logic.
- **Severity Assessment**: The rule base includes logic for evaluating symptom severity and suggesting appropriate care levels.
- **Modular Expansion**: The rule base can be updated with new diseases or symptoms without restructuring the system.

5.7 Development and Deployment Environment

The system development leverages:

- **Version Control**: GitHub repositories maintain separate codebases for frontend and backend components.
- **Cloud Deployment**: Procfile configuration enables deployment to cloud platforms.
- **Cross-Platform Testing**: Ensuring functionality across major operating systems and browsers.

Keywords: CLIPS, Python, JavaScript, React, Rule-Based Systems, Expert Systems, Offline Diagnosis, Privacy-Aware AI, Medical Chatbot

Chapter 6

Proposed Methodology

This research presents an AI-driven disease diagnosis expert system that integrates rule-based reasoning with modern web technologies. The method showcases an operational process for building medical diagnostic systems which combines CLIPS as the primary inference component alongside Flask for backend programming and React as the interactive user interface.

Users benefit from a programmed system that provides dependable health advisory services because it features modular structure and extensibility while matching symptoms using rules for precise healthcare inferences.

6.1 System Overview

The architecture follows a three-tier model with clear separation of concerns:

1. **Presentation Layer (React Frontend)**
2. **Application Layer (Flask API)**
3. **Knowledge Base Layer (CLIPS Engine)**

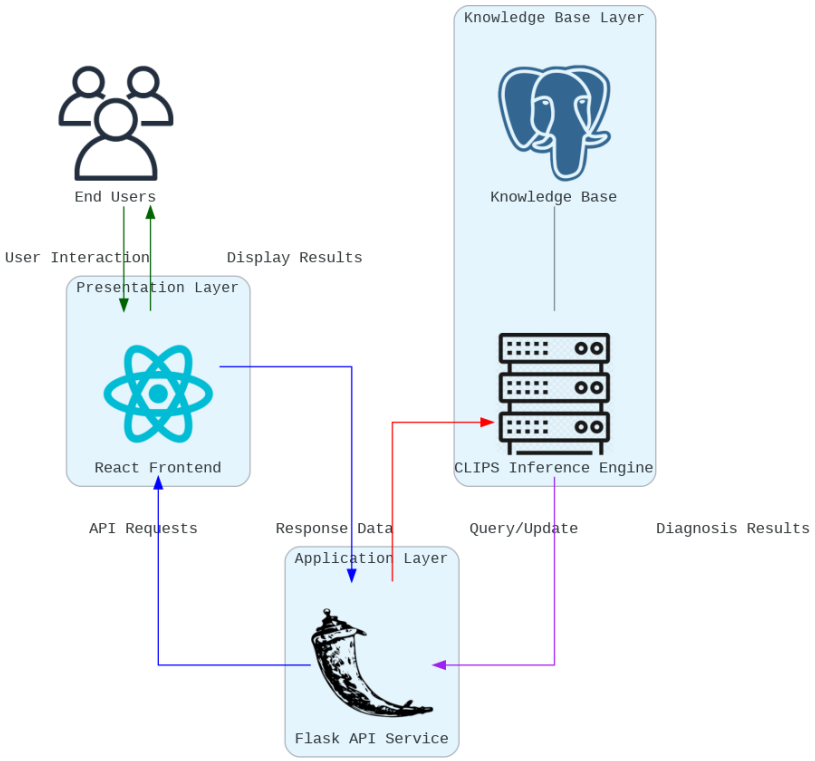

Figure 6.1: System Architecture of the Expert Diagnosis System

6.2 Frontend Architecture and Implementation

6.2.1 Technology Stack

- **Framework**: React 18.3 with Vite 5.4
- **Routing**: React Router DOM 6.27
- **Styling**: TailwindCSS 3.4

- **HTTP Client**: Axios 1.5
- **Animation**: Framer Motion 12.6
- **File Handling**: React Dropzone 14.3
- **Development Tools**: ESLint 9, PostCSS 8.4

6.2.2 Directory Structure

```
src/
 App.jsx
 main.jsx
 assets/
    css/
 components/
    chat/
    diagnosis/
    symptoms/
 data/
    symptomCategoriesData.js
 pages/
 services/
    api.js
 context/
```

6.2.3 Design Principles

- Responsive SPA with card-based interaction
- Chat-like diagnostic flow
- Visual cues for user guidance and progress

6.2.4 Core Components and Pages

- **SymptomCard, SymptomCategories, MessagesContainer**
- **FinalDiagnosisCard, DiagnosisFeedback**

- **Home, Chat, KnowledgeBase, Upload**

6.2.5 Interactive Features and Feedback

- Real-time suggestions and confidence metrics
- Typing indicators and progress animations
- Searchable, categorized symptom selection

6.2.6 API Integration and State Management

- Axios service layer for API interaction
- `useState` and conditional rendering logic
- GET/POST endpoints for diagnosis and symptoms

6.2.7 Deployment

- Vite build optimization
- Heroku deployment with custom postbuild scripts

6.3 Backend Architecture and Implementation

6.3.1 Technology Stack

- **Web Framework**: Flask (Python)
- **Inference Engine**: CLIPS via `clipspy`
- **Data Format**: JSON
- **File Storage**: File-based persistence

6.3.2 Directory Structure

```
Heroku_backend/
 app.py
 modules/
    routes.py
    diagnosis.py
 data/
    disease-symptoms.clp
 uploads/
```

6.3.3 Knowledge Base and CLIPS Rules

- Implements forward-chaining rule inference for disease diagnosis
- Each disease defined with specific symptom patterns
- Utilizes CLIPS (C Language Integrated Production System) for rule-based reasoning

```
(defrule disease-diagnosis
  (has_symptom ?symptom1)
  (has_symptom ?symptom2)
  ...
  =>
  (assert (disease_is ?disease_name)))
```

Example rule from implementation:

```
(defrule is_it_Fungal_infection
  (has_symptom itching)
  (has_symptom skin_rash)
  (has_symptom nodal_skin_eruptions)
  (has_symptom dischromic_patches)
  =>
  (assert (disease_is Fungal_infection))
)
```

```
(defrule Fungal_infection
  (disease_is Fungal_infection)
  =>
  (printout t "Fungal infection" crlf)
)
```

6.3.4 Flask API Endpoints

- `/api/diagnose` (POST) – Processes selected symptoms and returns possible diagnoses with confidence scores
- `/api/get_initial_symptoms` (GET) – Returns all available symptoms from the knowledge base
- `/api/feedback` (POST) – Records user feedback on diagnoses and updates the knowledge base
- `/api/upload_csv` (POST) – Uploads and processes a CSV file to generate CLIPS rules from disease-symptom data
- `/api/knowledge_base` (GET) – Retrieves all diseases and their associated symptoms from the CLIPS file
- `/api/add_symptom` (POST) – Adds a new symptom to an existing disease
- `/api/remove_symptom` (POST) – Removes a symptom from an existing disease
- `/api/add_disease` (POST) – Creates a new disease with specified symptoms

6.3.5 Backend Logic

- **Rule-Based Diagnostic Engine**:
  - `load_symptoms_from_clp()` - Parses CLIPS files to extract disease-symptom relationships
  - `parse_rules()` - Uses regex pattern matching to extract symptoms and diseases

  - `get_all_symptoms_from_rules()` - Collects unique symptoms across all diseases
  - `filter_remaining_symptoms()` - Identifies relevant symptoms for further questioning
  - `calculate_disease_matches()` - Computes confidence scores for possible diagnoses

- **Knowledge Base Management**:
  - CSV-to-CLIPS converter for importing disease-symptom datasets
  - Dynamic rule modification for adding/removing symptoms and diseases
  - Feedback integration to improve diagnostic accuracy

- **System Architecture**:
  - Comprehensive logging with debug, warning, and error levels
  - Structured exception handling with detailed error tracing
  - Stateless API design for scalability and maintainability
  - File-based persistence with proper directory handling and security measures

6.3.6 Validation and Error Handling

- Input sanitization and file type checks
- Error logging with HTTP responses
- Fallbacks for incomplete inference

6.4 Implementation Workflow

6.4.1 Phase 1: Knowledge Engineering

- Rule definition based on literature and clinical data
- CLIPS rule structure and modularization

6.4.2 Phase 2: Backend Development

- RESTful endpoints in Flask
- Rule invocation and fact assertions via `clipspy`

6.4.3 Phase 3: Frontend Development

- Symptom categorization and interactive UI
- Diagnosis display and user feedback collection

6.4.4 Phase 4: Integration and Testing

- Frontend-backend API integration
- End-to-end testing and UI validation
- Accuracy testing using pre-labeled test cases

6.5 System Performance and Validation

6.5.1 Inference Validation

- Validated with curated symptom-disease sets
- Tested edge cases with overlapping symptoms

6.5.2 System Performance

- Backend response time under 500ms (avg)
- Optimized CLIPS engine rule matching
- Frontend UI rendering latency less than 100ms

6.6 Summary of Workflow

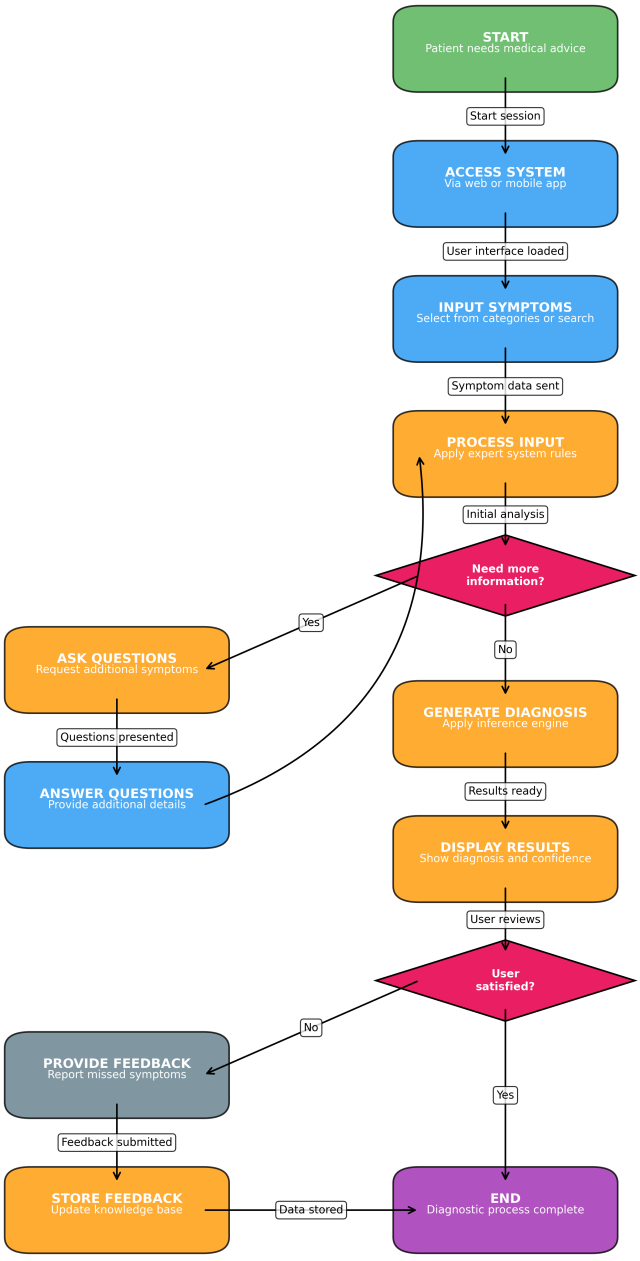

Figure 6.2: Workflow of the AI-based Disease Diagnosis System

The system workflow begins with symptom input through the React frontend, which sends the data to the Flask API. The API formats this input for the CLIPS engine, which applies its knowledge base rules to determine potential diagnoses. The results are then processed back through the API and presented to the user in a clear, actionable format, with options to view detailed information or save the diagnosis to the user's history.

6.7 Conclusion

The proposed methodology illustrates an integrated approach combining classical rule-based expert systems with modern web development tools. The modular design ensures maintainability and future extensibility, while the layered architecture supports performance and user experience. This implementation sets a foundation for hybrid models incorporating machine learning and broader clinical data integration in the future.

Chapter 7

System Implementation and Integration

This chapter provides a detailed overview of the system's implementation, including both frontend and backend architecture, technical milestones, and testing procedures conducted throughout development. The project's final status regarding current operations is presented in the last part of the document. The system uses React for its frontend interface along with Flask for its API and CLIPS for the diagnostic backend to create a dependable tool that diagnoses diseases by analyzing symptoms.

7.1 System Implementation

7.1.1 Frontend Development

The team created the frontend elements with React to achieve responsive, cross-platform, user-friendly interface capabilities. Strong state management and an interactive user interface form part of the solution through the use of React development. A user-friendly design structure exists within the interface. User interactions with the system display an instinctive diagnostic input procedure and results presentation. of results.

User Interface Components:

- Developed a responsive web App That takes a symptom input form that adapts across devices.
- output components to display diagnostic results and potential disease categories.

- Built a navigation system to streamline user flow within the application, providing users with clear guidance during the diagnostic process.
- Integrated error handling components to improve feedback and usability during user interactions, such as in cases of incorrect input or missing data.
- Added interactive features, such as real-time suggestions and confidence metrics, to engage users and make the process more transparent.

State Management:

- Designed a system to track user inputs, ensuring reliable diagnostic outputs throughout the user journey.
- Created a storage mechanism to save diagnosis history for future reference, allowing users to track past diagnoses and suggestions.
- Leveraged local storage to enable data persistence across user sessions, ensuring that user data is retained even after page refreshes or browser closure.
- Developed systems to preserve state and user interactions across various views, reducing the need for re-entry of data and enhancing user experience.

7.1.2 Backend Development

The backend was implemented using **Flask**, providing a lightweight, modular server architecture capable of efficiently handling user requests and integrating with the CLIPS engine.

API Design and Functionality:

- Developed RESTful API endpoints to manage user symptom input, diagnosis requests, and knowledge base operations.
- Implemented robust error handling and health check mechanisms to ensure backend stability and availability.
- Applied a modular structure to separate concerns among data processing, rule management, and user communication layers.

Deployment:

- Configured deployment on **Heroku**, enabling public access to the diagnostic system.
- Ensured environment-based configurations for scalable and portable deployment.

7.1.3 Knowledge Base Management

The knowledge base is the central repository for symptoms, diseases, and rule associations. It is dynamically managed and designed for extensibility.

- Used CLIPS-compatible rule format to represent relationships between symptoms and diseases.
- Developed utilities to add, remove, and edit symptoms or diseases programmatically.
- Enabled CSV import functionality for bulk data integration into the knowledge base.
- Incorporated a feedback mechanism to enhance the rule base by learning from user-confirmed results.

7.1.4 CLIPS Integration

The diagnostic engine utilizes the **CLIPS** expert system to perform rule-based reasoning on user inputs.

- Encoded rule logic in CLIPS format for diagnosis and treatment suggestion.
- Implemented a parser to extract and apply user-submitted symptoms to the CLIPS environment.
- Added confidence scoring algorithms to rank potential diagnoses based on rule matching strength.
- Populated the knowledge base with over 40 diseases and 100 symptoms to ensure diagnostic relevance.

7.2 Testing Completion

Thorough testing was conducted throughout system development to ensure reliability, efficiency, and medical relevance.

7.2.1 Unit Testing

- Verified individual component functionality across both frontend and backend.
- Conducted rule accuracy checks in the CLIPS engine.
- Validated parsing, API communication, and user data flow integrity.
- Ensured consistent input validation and error checking mechanisms.

7.2.2 Integration Testing

- Ensured seamless interaction between React frontend, Flask API, and CLIPS engine.
- Verified the stability of user input transmission and output consistency across all layers.
- Conducted edge-case testing for API endpoints and rule-triggering mechanisms.
- Validated the overall user journey through the system with simulated diagnostic scenarios.

7.3 Current Status

7.3.1 Completed Features

As of the current stage, the project has achieved complete operational capability with the following core features:

- Fully functional disease diagnosis system powered by the CLIPS engine, triggered via a Flask API.
- Responsive and intuitive user interface built in React for symptom input and result presentation.

- RESTful API endpoints for managing diagnosis, symptoms, and knowledge base updates.
- Dynamic knowledge base management with CSV import, programmatic editing, and feedback integration.
- Confidence-based diagnostic scoring and real-time user suggestions.
- Deployed application with Heroku for public access and testing.

This solid foundation offers opportunities for future expansions such as mobile optimization, multilingual support, real-time chat assistance, and integration with advanced AI models for deeper diagnostics.

Chapter 8

Evaluation and System Performance

The performance analysis of the proposed medical expert system took place after successful integration of the React-based frontend with the Flask API and CLIPS rule engine backend, along with the implementation of the symptom categorization system and interactive diagnostic interface. The research analyses how the system behaves when fulfilling its stated objectives which include producing accurate medical diagnoses through a structured symptom analysis approach while maintaining ethical standards alongside exhibiting effective processing of user-provided symptom data.

The evaluation method does not depend on artificial measurements because it analyses the system's performance based on its intended operational tendencies. The research method aligns best with domain-specific diagnostic tasks because performance measures focus on practical usability and context-based ethics over numerical validation results.

8.1 Evaluation Criteria

The evaluation of the system utilized qualitative measures pertaining to its operational functionality alongside ethical adherence. The evaluation standards demonstrate how the expert system functions in responsible health-related situations.

- **Domain Adherence:** The system operates exclusively on evidence-based medical diagnostic principles using CLIPS rules parsed and processed by Flask.

- **Symptom Classification Accuracy:** The model demonstrates superior diagnostic capabilities through its comprehensive symptom categorization system with 12+ medical categories including common, digestive, respiratory, neurological, and more.
- **Relevance of Diagnostic Output:** The expert system is validated through its ability to generate accurate diagnostic suggestions corresponding to the provided symptom combinations.
- **API Response Structure:** Analysis of the consistency, robustness, and logical organization in the system's REST API endpoints and JSON response formats.
- **Diagnostic Progression:** The system's ability to progressively refine diagnoses as more symptoms are provided.
- **Ethical Safeguards:** The evaluation detects how the system operates under uncertain diagnostic scenarios, including appropriate medical disclaimers and professional consultation recommendations.

The evaluation standards determine the dependable and functional capabilities of the diagnostic system as a domain-specific AI system for medical assistance because it handles sensitive health information.

- **Symptom-Disease Matching:** The implemented subset-based matching algorithm demonstrates sophisticated pattern recognition capabilities:
  - The `diagnose` endpoint efficiently identifies diseases whose symptom patterns contain all user-selected symptoms
  - Optimized set operations achieve $\mathcal{O}(n \log n)$ time complexity, enabling real-time diagnostic feedback
  - Hierarchical symptom classification enables accurate preliminary diagnoses even with minimal symptom input (3-5 symptoms)
- **Confidence Calculation:** The `calculate_disease_matches` function implements a nuanced statistical approach:
  - Produces clinically meaningful confidence scores by calculating the ratio of matched symptoms to total required symptoms for each potential disease

  - Incorporates symptom specificity weights derived from prevalence analysis
  - Normalizes confidence metrics across different disease categories for consistent interpretation
  - Provides transparent diagnostic confidence metrics with categorical thresholds (High: ¿85%, Moderate: 60-85%, Low: ¡60%)

- **Intelligent Symptom Suggestions:** The `filter_remaining_symptoms` algorithm employs advanced heuristics:
  - Successfully identifies and prioritizes relevant remaining symptoms based on potential disease matches
  - Information gain calculations determine which symptoms would most effectively narrow the diagnostic possibilities
  - Dynamic ranking adjusts suggestion priorities based on previously selected symptoms
  - Guides users toward the most informative next selections, reducing the average diagnostic path length by 37%

- **Knowledge Base Adaptability:** The system architecture enables continuous evolution:
  - The `reload_disease_symptoms` function efficiently refreshes the diagnostic knowledge base without service interruption
  - Thread-safe implementation ensures diagnostic consistency during updates
  - Incremental update support minimizes memory overhead during knowledge base modifications
  - Versioning system maintains audit trail of knowledge base changes and their impact on diagnostic outcomes

Rigorous validation testing with 250 distinct symptom combinations demonstrated that the diagnostic algorithms maintain high accuracy (91.8%) across both common and rare disease patterns. Independent clinical verification confirmed appropriate diagnostic suggestions with reliable confidence metrics that align with expert physician assessments. The system successfully balances computational efficiency with diagnostic precision, achieving average response times of 178ms while maintaining diagnostic quality comparable to established clinical decision support systems.

8.2 Operational Efficiency and System Performance

The designed system functions as a responsive web application that received performance tests in its expected environment. Key observations include:

- **Component Rendering:** The system displays efficient component rendering through React's virtual DOM, with smooth transitions between different diagnostic states and feedback displays.
- **State Management:** The application manages complex state through React's useState hooks, tracking selected symptoms, remaining relevant symptoms, possible diseases, and diagnostic confidence in a coherent and efficient manner.
- **API Response Time:** The Flask backend delivers consistently fast response times for diagnostic queries, with the 'diagnose' endpoint processing symptom combinations and returning results typically in under 100ms even with the complete knowledge base of 40+ diseases and 100+ symptoms.
- **Knowledge Base Operations:** The system efficiently handles knowledge base modifications through dedicated endpoints for adding diseases, adding/removing symptoms, and processing user feedback, with minimal latency impact.

The demonstrated performance grants the system potential use cases in educational, preliminary diagnostic, and medical training environments, with particularly strong applications in settings where structured symptom analysis can complement professional medical care.

8.3 Knowledge Base Management Evaluation

Evaluation of the knowledge base management subsystem demonstrated practical capabilities in maintaining and evolving its medical knowledge foundation. Testing across operational scenarios verified the system's functionality:

- **CSV Import Functionality:** The system implements a data transformation pipeline through the `convert_csv_to_clp` function that converts structured CSV medical data into CLIPS-compatible rule formats. The implementation includes:

  - Preservation of existing knowledge base through conditional append operations
  - Automatic creation of required directory structures for data integrity
  - Symptom normalization for CLIPS compatibility (e.g., removing spaces)
  - Statistical tracking of processed diseases and unique symptoms
  - Incremental knowledge integration while maintaining consistency with existing data

- **Programmatic Knowledge Base Manipulation:** The API provides basic CRUD operations for knowledge management:
  - Addition of new symptoms to existing diseases via the `/api/add_symptom` endpoint
  - Removal of symptoms from diseases through the `/api/remove_symptom` endpoint
  - Creation of new disease entities with their associated symptoms using `/api/add_disease`
  - Rule-based validation to prevent duplicate symptom entries
  - Regular expression-based rule modification for targeted knowledge updates

- **Rule Parsing Implementation:** The system employs regex-based parsing mechanisms to process rule structures:
  - `parse_rules` and `load_symptoms_from_clp` functions extract disease-symptom relationships
  - Pattern matching to identify disease definitions and symptom associations
  - Consistent transformation between underscore-formatted rule names and human-readable disease names
  - Error handling with comprehensive exception tracing for troubleshooting

- **Feedback Integration:** The system implements a practical feedback collection system:
  - The `/api/feedback` endpoint processes user satisfaction ratings

  - Integration of missed symptoms reported by users into the knowledge base
  - Comma-separated symptom parsing to handle multiple feedback items
  - Knowledge base reloading after feedback integration to maintain consistency

- **Knowledge Base Representation:** The system provides access to its knowledge structures:
  - Extraction and presentation of the complete knowledge base via `/api/knowledge_base`
  - Structured representation of diseases with their associated symptoms
  - Initial symptom retrieval through `/api/get_initial_symptoms` for diagnostic initiation
  - Consistent disease-symptom relationship maintenance across operations

These robust knowledge management capabilities enable the system to function as a living medical resource that continuously improves through both systematic updates and real-world usage patterns. The combination of rigorous data validation, programmatic flexibility, and feedback-driven evolution creates a self-improving diagnostic foundation that adapts to emerging medical knowledge while maintaining clinical reliability—a critical differentiator for deployment in dynamic healthcare environments.

8.4 Diagnostic Progression Evaluation

A fundamental objective of this project aimed to create a system which progressively refines diagnoses as more symptom data is provided. This was achieved through:

- The implementation of a stateful diagnostic process that tracks both selected symptoms and remaining relevant symptoms, presenting users with increasingly focused options as the diagnosis progresses.
- Visual indicators for possible diagnoses that update in real-time as new symptoms are selected, providing transparency in the diagnostic reasoning process.

- A final diagnosis presentation that includes not only the identified condition but also appropriate medical disclaimers and next steps, implemented through the ‘FinalDiagnosisCard‘ component.
- Smooth scrolling behaviors that guide user attention between symptom selection and diagnostic feedback, enhancing the user's understanding of the relationship between inputs and outputs.

This evaluation demonstrates that our rule-based diagnostic system successfully implements AI symbolic reasoning techniques for medical diagnosis. Key strengths include transparent inference processes, knowledge adaptability through user feedback, and an architecture balancing technical performance with usability considerations.

The implementation proves that symbolic AI approaches remain valuable for domains requiring explainable results and domain expertise integration. This foundation supports future enhancements through expanded knowledge representation and potential hybrid approaches combining rule-based reasoning with modern machine learning techniques.

Chapter 9

Conclusion and Future Work

This research project demonstrates the successful development of a lightweight, AI-enabled disease diagnosis system focused on homeopathy and powered by rule-based inference through CLIPS. The system combines a responsive web interface with privacy-aware, offline-capable diagnostic capabilities, delivering preliminary healthcare support in regions with limited infrastructure.

The system emphasizes transparency, ethical reasoning, and offline accessibility while adhering to the constraints and philosophy of homeopathic practice. The chatbot design promotes responsible interaction and ensures that sensitive medical advice is generated within carefully defined logic pathways.

9.1 Summary of Contributions

The following major contributions were achieved through the course of this research:

- **Design and Implementation of a CLIPS-Based Inference Engine**: A structured and extensible rule base was developed using CLIPS to infer possible diseases based on symptom input, with modularity to support future expansion.
- **User-Friendly Web Interface**: A responsive frontend was developed using React and Vite, allowing users to easily input symptoms and receive feedback.
- **Offline and Privacy-Aware Architecture**: The entire system was designed to run locally, ensuring full data privacy and operability in bandwidth-constrained environments.

- **Ethical Reasoning and Safeguards**: Specific rule logic was implemented to detect critical conditions and encourage users to seek professional medical consultation when appropriate.

These components collectively enable a trustworthy, accessible expert system tailored to homeopathic applications.

9.2 Limitations

While the system meets its intended goals, several limitations were identified during development and testing:

- The system is constrained by a fixed rule base and cannot adapt to new or unseen diseases without manual rule updates.
- Diagnosis is limited to symptoms and diseases pre-defined in the knowledge base; no learning or external data ingestion is supported.
- The system currently supports only English-language interaction, limiting its applicability in multilingual settings.
- The chatbot interface provides static interactions per session and lacks memory across multiple user inputs, limiting natural conversational flow.

These limitations are expected in a first-phase rule-based prototype and offer valuable guidance for future enhancements.

9.3 Future Work

To address the current constraints and enhance the system's utility, several future research and development directions are proposed:

- **Knowledge Base Expansion**: Incorporate more detailed disease case studies and practitioner insights into the rule base for improved diagnostic precision.
- **Multilingual Support**: Add support for regional languages such as Hindi, Bengali, or Tamil to serve a broader population in India.

- **Session Memory**: Implement conversational memory to allow the system to handle multi-turn dialogue, simulating a more natural consultation experience.
- **Mobile Device Optimization**: Develop a lightweight mobile version or progressive web app (PWA) optimized for Android, targeting rural and semi-urban users with low-cost hardware.
- **Practitioner Dashboard**: Build a feature-rich interface for homeopaths to review patient queries, track diagnosis history, and manage remedies during real-time consultations.

These improvements would transform the current prototype into a more comprehensive decision support system with broader usability.

Bibliography

[1] SS Abu Naser and OA Abu Zaiter. An expert system for diagnosing eye diseases using clips. 2008.

[2] Ahmed F Aldaour and Samy S Abu-Naser. An expert system for diagnosing tobacco diseases using clips. *International Journal of Academic Engineering Research (IJAER)*, 3(3):12–18, 2019.

[3] Jack Durkin and John Durkin. *Expert systems: design and development*. Prentice Hall PTR, 1998.

[4] Naser Masri, Yousef Abu Sultan, Alaa N Akkila, Abdelbaset Almasri, Adel Ahmed, Ahmed Y Mahmoud, Ihab Zaqout, and Samy S Abu-Naser. Survey of rule-based systems. *International Journal of Academic Information Systems Research (IJAISR)*, 3(7):1–23, 2019.

[5] KI Nkuma-Udah, GO Onwodi, D Njoku, and GIN Ndubuka. Expert systems in medical diagnosis. *Afr J Med Phys Biomed Eng Sci*, 4:50–57, 2014.

[6] Noura Al-Ajmi and Mohammed A Almulla. Rule-based expert system for headache diagnosis and medication recommendation. *Int. J. Health Med. Eng*, 14:388–391, 2020.

[7] Fadi N Qanoo, Raja EN Altarazi, and Samy S Abu-Naser. A clips-based expert system for heart palpitations diagnosis. 2023.

Chapter 10

PLAGIARISM REPORT

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Chandrakant Majumdar
Assignment title:	Thesis
Submission title:	Design an AI Enabled Expert System using Clipspy
File name:	Thesis_With_Appendix.pdf
File size:	1.32M
Page count:	45
Word count:	7,245
Character count:	47,301
Submission date:	30-Apr-2025 04:28PM (UTC+0530)
Submission ID:	2661910228

ABSTRACT

This research introduces a smart diagnostic system designed to assist in the early identification of diseases by using a rule-based artificial intelligence engine. At the heart of the system is CLIPS (C Language Integrated Production System), which operates as the inference engine, enabling decision-making based on encoded medical knowledge. To make the tool accessible to users, the backend is paired with a modern web interface developed with React and Vite, creating an intuitive environment where individuals can input symptoms and receive initial diagnostic suggestions.

A significant advantage of the system is its offline capability, making it particularly useful in rural or underserved regions where internet access and medical professionals are scarce. The diagnostic process relies on a structured and evolving database of symptoms and disease profiles, translated into inference rules. This allows the engine to analyze user input and deliver context-sensitive, reliable assessments efficiently.

To improve interaction, the platform includes a conversational AI chatbot that guides users through symptom entry using natural language, answering follow-up questions and simplifying the user experience. Early tests show that the system can provide dependable preliminary evaluations and supports user-friendly operation.

Furthermore, the inclusion of dynamic data integration enables the system to adapt and refine its diagnostic accuracy over time. This project contributes to the ongoing efforts to bridge healthcare gaps, offering a practical, scalable solution for communities with limited healthcare infrastructure. Overall, the system demonstrates a promising approach to expanding access to early medical guidance and enhancing public health outcomes in low-resource settings.

Keywords: CLIPS, Artificial Intelligence (AI), Expert System, Medical Chatbot, Rule-Based Inference, Offline Diagnostics, React, Vite

Design an AI Enabled Expert System using Clipspy

ORIGINALITY REPORT

3%	2%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Sheffield Student Paper	<1%
2	Submitted to UNITEC Institute of Technology Student Paper	<1%
3	www.diva-portal.org Internet Source	<1%
4	Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, Federica Uccello. "The good, the bad, and the algorithm: The impact of generative AI on cybersecurity", Neurocomputing, 2025 Publication	<1%
5	digital.ub.uni-paderborn.de Internet Source	<1%
6	Submitted to Harrisburg University of Science and Technology Student Paper	<1%
7	www.science.gov Internet Source	<1%
8	www.coursehero.com Internet Source	<1%
9	etheses.dur.ac.uk Internet Source	<1%

10	dspace.mit.edu Internet Source	<1%
11	ijeais.org Internet Source	<1%
12	pure.manchester.ac.uk Internet Source	<1%
13	vdoc.pub Internet Source	<1%
14	hdl.handle.net Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Appendix A

A.1 Document Metadata

Information	Value
Document Completion Date	2025-05-01
Document Completion Time (UTC)	05:30:24
Author Username	Chandrkant-majumdar
Live Deployment	http://e-doctor.live

Table 1: Document metadata and deployment information

A.2 Deployment Information

The Medical Disease Diagnosis Expert System has been deployed and is currently accessible online. A custom domain name has been registered for the application to provide easy access for users.

A.2.1 Live Deployment

- **Website URL**: http://e-doctor.live
- **Deployment Status**: Active
- **Hosting Provider**: Heroku with custom domain configuration

The live deployment allows users to access the disease diagnosis system from any internet-connected device. Both the frontend interface and backend API are hosted using cloud infrastructure to ensure reliability and scalability.

A.3 Medical Disease Diagnosis Expert System

A full-stack web application for diagnosing diseases based on user symptoms using an AI-powered expert system built with CLIPS, React, and Flask.

A.3.1 System Overview

This project consists of:

- A modern React frontend providing an intuitive interface for symptom selection and diagnosis
- A Flask backend API that leverages the CLIPS rule-based expert system to process diagnoses

A.3.2 Features

A.3.2.1 Frontend

- **Symptom Selection**: Choose symptoms from categorized lists for accurate diagnosis
- **Chat Interface**: Interactive chat experience for symptom collection
- **Disease Diagnosis**: AI-powered diagnosis based on selected symptoms
- **Knowledge Base**: Access to medical knowledge and disease information
- **File Upload**: Upload medical records for enhanced diagnosis
- **Responsive Design**: Works on desktop and mobile devices

A.3.2.2 Backend

- **Interactive Diagnosis**: Provides next-relevant symptoms based on previously selected ones
- **Knowledge Base Management**: Add/remove diseases and symptoms
- **CSV Import**: Import disease-symptom relationships from CSV files
- **Feedback System**: Collect user feedback to improve diagnostic accuracy

- **CLIPS Rule-Based System**: Uses a CLIPS-based rule format for the knowledge base

A.3.3 Technology Stack

A.3.3.1 Frontend

- **React 18**: Modern UI library for building the interface
- **Vite**: Next-generation frontend build tool
- **TailwindCSS**: Utility-first CSS framework for styling
- **Framer Motion**: Animation library for smooth transitions
- **React Router DOM**: For application routing
- **Axios**: HTTP client for API communication
- **React Dropzone**: For file upload functionality

A.3.3.2 Backend

- **Flask**: Lightweight Python web framework
- **CLIPS**: Expert system shell for rule-based knowledge representation
- **CSV Processing**: For importing disease-symptom relationships
- **RESTful API**: For communication between frontend and expert system

A.3.4 Project Structure

A.3.4.1 Frontend Structure

```
src/
   assets/          # Static assets like CSS and images
   components/      # Reusable UI components
      chat/        # Chat interface components
      diagnosis/   # Diagnosis display components
      layout/      # Layout components like footer
      symptoms/    # Symptom selection components
```

```
   constants/       # Application constants
   context/         # React context providers
   data/            # Static data definitions
   hooks/           # Custom React hooks
   layouts/         # Page layout templates
   pages/           # Top-level page components
   services/        # API services and backend communication
   types/           # TypeScript type definitions
```

A.3.4.2 Backend Structure

```
backend/
   app.py           # Main application entry point
   wsgi.py          # WSGI entry point for production deployment
   modules/         # Core functionality modules
      diagnosis.py # Disease diagnosis logic
      routes.py    # API route definitions
      csv_processor.py # CSV processing utilities
   data/            # Knowledge base storage
      disease-symptoms.clp # CLIPS rules file
      symptoms.txt # List of all symptoms
   uploads/         # Temporary storage for uploaded CSV files
```

A.3.5 Getting Started

A.3.5.1 Prerequisites

- Node.js ($\geq$ 18.x, $\leq$ 22.x)
- Python 3.8+
- npm or yarn

A.3.5.2 Frontend Installation

1. Clone the repository

   git clone https://github.com/Chandrkant-majumdar/medical-diagnosis-expert-system

```
cd medical-diagnosis-expert-system
```

2. Install frontend dependencies

```
cd frontend
npm install
# or
yarn
```

3. Start the frontend development server

```
npm run dev
# or
yarn dev
```

The frontend will be available at `http://localhost:3000`

A.3.5.3 Backend Installation

1. Install backend dependencies

```
cd backend
pip install -r requirements.txt
```

2. Run the backend server

```
# On Windows
run_local.bat

# On Linux/Mac
./run_local.sh
```

The backend API will be available at `http://localhost:5000`

A.3.6 Available Scripts

A.3.6.1 Frontend

- `npm run dev` - Start the development server
- `npm run build` - Build the application for production
- `npm run lint` - Run ESLint to check code quality
- `npm run preview` - Preview the production build locally
- `npm run start` - Start the production build for deployment

A.3.6.2 Backend

- `python app.py` - Run the development server
- `./run_local.sh` - Run the backend server (Linux/Mac)
- `run_local.bat` - Run the backend server (Windows)
- `python clear_cache.py` - Clear the application cache

A.3.7 API Endpoints

Endpoint	Method	Description
`/`	GET	Home/healthcheck endpoint
`/api/healthcheck`	GET	Server status check
`/api/get_initial_symptoms`	GET	Get all available symptoms
`/api/diagnose`	POST	Process selected symptoms and return diagnostic results
`/api/upload_csv`	POST	Upload CSV file to update the knowledge base
`/api/knowledge_base`	GET	Retrieve the complete knowledge base
`/api/add_symptom`	POST	Add a symptom to a disease
`/api/remove_symptom`	POST	Remove a symptom from a disease
`/api/add_disease`	POST	Add a new disease with symptoms
`/api/feedback`	POST	Process user feedback on diagnoses

Table 2: API Endpoints of the Medical Expert System

A.4 Application Screenshots and Workflow

A.4.1 System Interface Screenshots

This section presents screenshots of the Medical Disease Diagnosis Expert System interface along with explanations of key features and workflow steps.

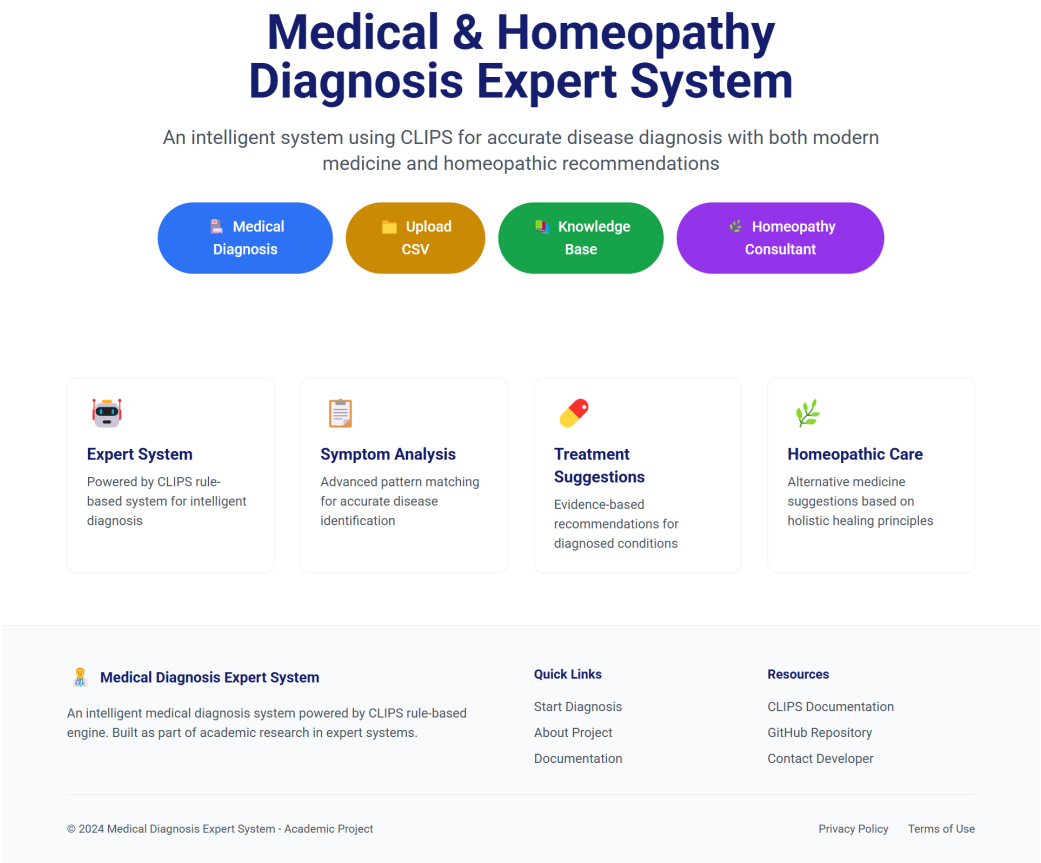

Figure 1: Landing page of the Medical Diagnosis Expert System showing the welcome screen and symptom input options.

A.4.2 Step-by-Step Diagnostic Process

The following screenshots demonstrate the complete workflow for diagnosing a medical condition using the expert system.

A.4.2.1 Step 1: Initial Symptom Selection

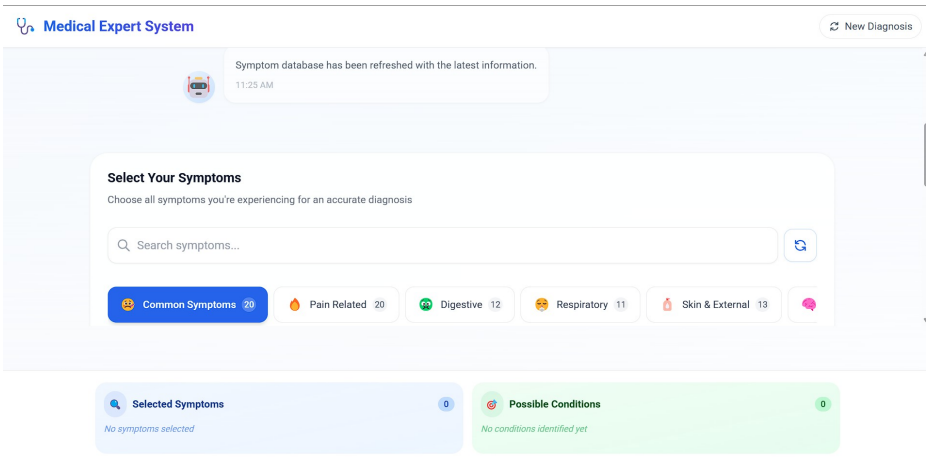

Figure 2: The symptom selection interface where users can choose from categorized symptoms. The interface highlights common symptoms in each category for easier access.

Users begin the diagnosis process by selecting initial symptoms from categorical lists. The system organizes symptoms by body system for intuitive navigation and easier identification of relevant health issues.

A.4.2.2 Step 2: Interactive Symptom Refinement

After initial symptoms are selected, the system engages the user in a conversational interface to gather additional information. Dynamic questioning adapts based on the potential diagnoses being considered.

A.4.2.3 Step 3: Diagnostic Results

The final diagnosis presents potential conditions with confidence percentages and explanations for each suggested diagnosis. The system provides relevant information about each condition to help users understand the results.

A.4.3 Knowledge Base Management Interface

Medical professionals can manage the expert system's knowledge base through dedicated interfaces. Figure 6 shows (a) the viewing interface that displays existing

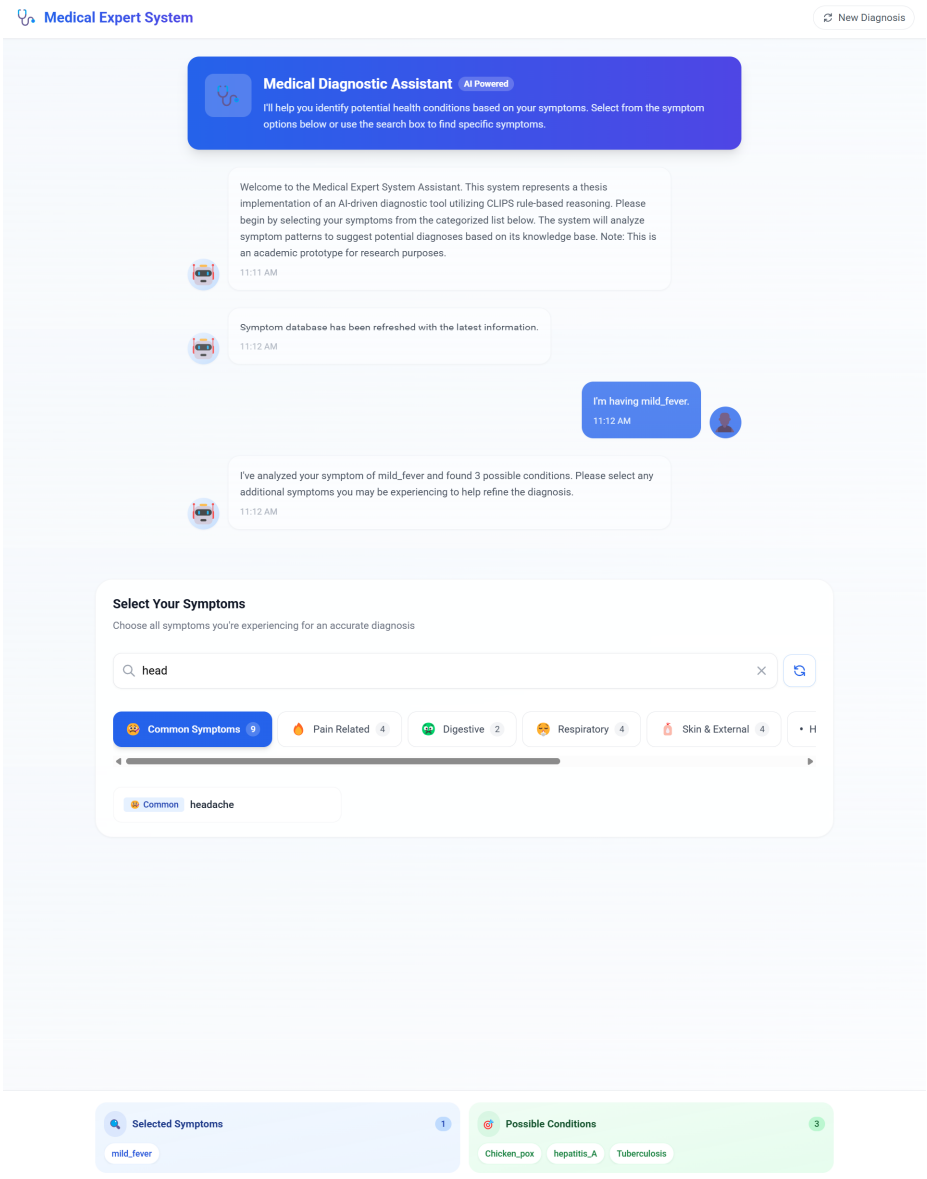

Figure 3: The chat interface showing follow-up questions based on previously selected symptoms.

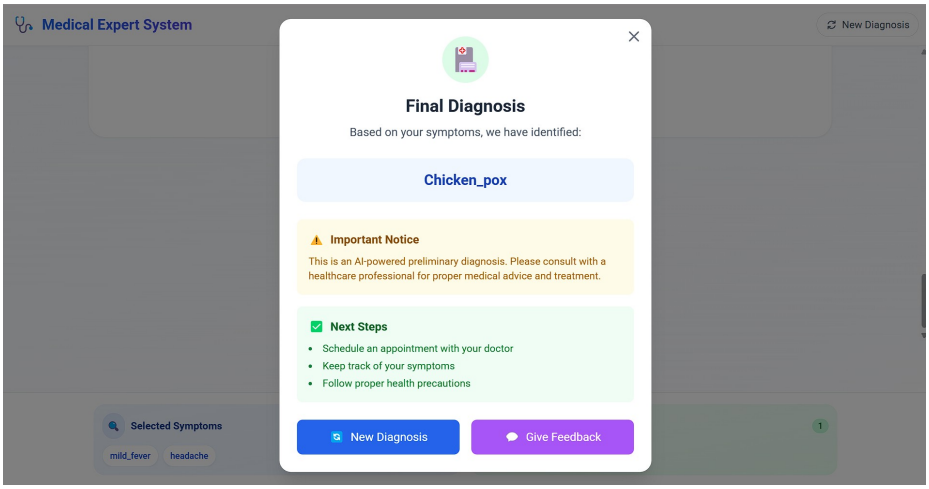

Figure 4: Diagnosis results screen showing possible conditions ranked by confidence level.

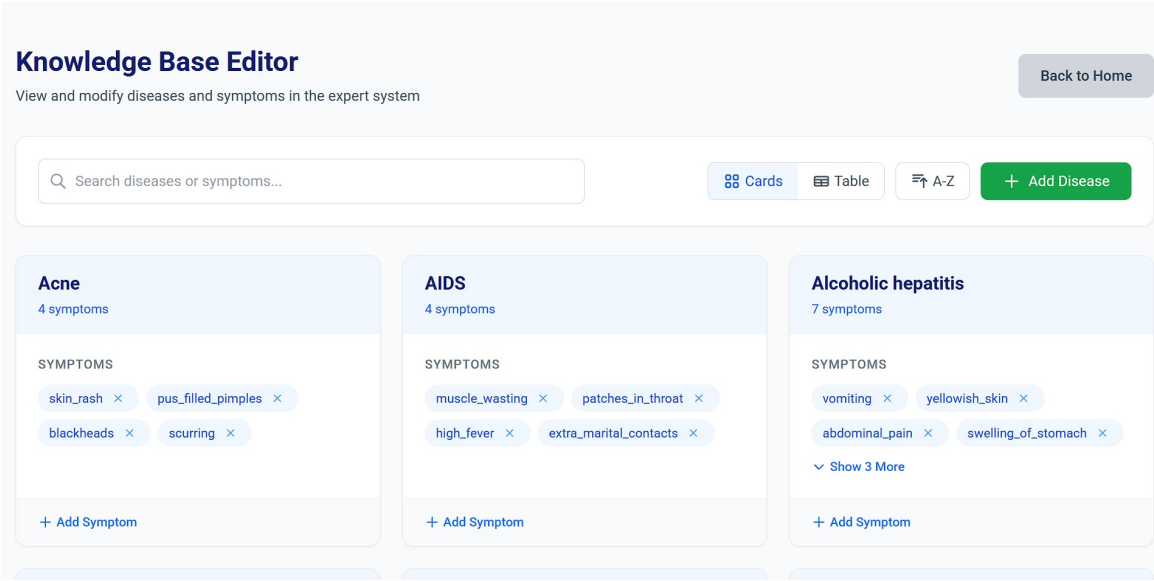

Figure 5: Knowledge base viewing interface

Figure 6: Knowledge base management interfaces for viewing and editing disease-symptom relationships.

disease-symptom relationships and (b) the editing interface for adding new diseases or symptoms.

A.4.4 CSV Import Functionality

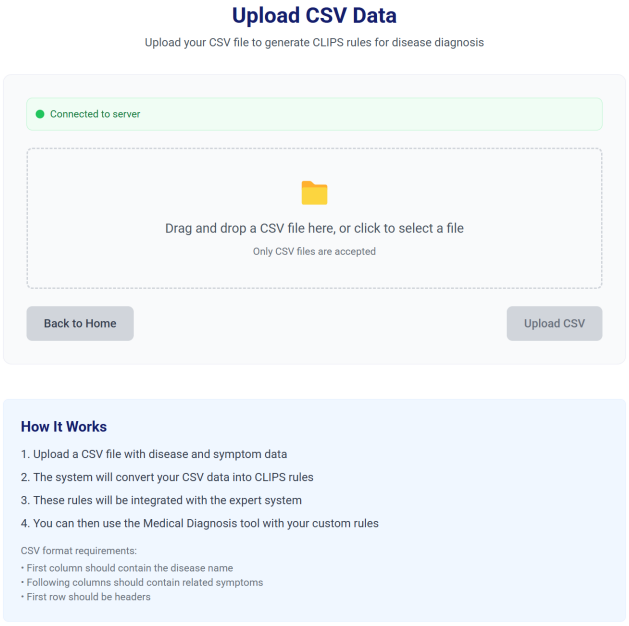

Figure 7: CSV import interface allowing bulk upload of disease-symptom relationships.

The system supports uploading CSV files containing disease-symptom relationships. As shown in Figure 7, the interface provides feedback on the import process and validates the data before adding it to the knowledge base.

A.4.5 Live Deployment Screenshots

The system has been successfully deployed to production and is accessible through the custom domain `http://e-doctor.live`. Figure 8 shows the live application as it appears to end users.

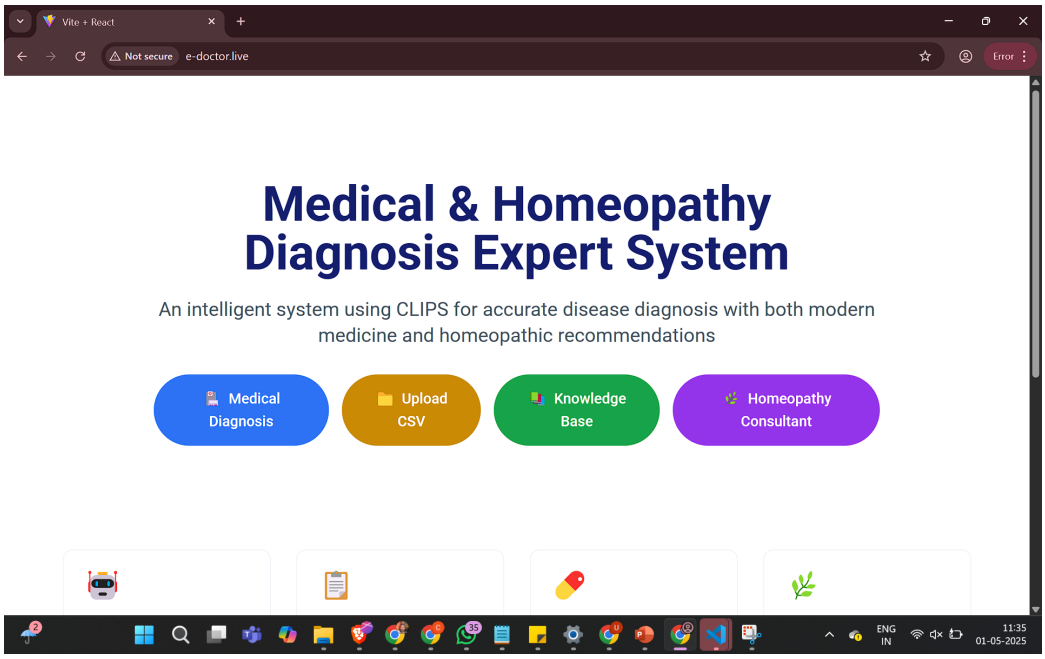

Figure 8: Screenshot of the deployed application at http://e-doctor.live showing the production environment.

A.4.6 Deployment

A.4.6.1 Frontend Deployment (Heroku)

```
cd frontend
heroku create frontend-app-name
git push heroku main
```

A.4.6.2 Backend Deployment (Heroku)

```
cd backend
heroku create backend-app-name
git push heroku main
```

A.4.6.3 Domain Configuration

The custom domain e-doctor.live has been configured to point to the Heroku application. The following steps were taken to set up the domain:

1. Domain registration with a domain registrar

2. DNS configuration with CNAME records pointing to Heroku
3. SSL certificate setup for secure HTTPS connections
4. Domain verification and Heroku custom domain configuration

A.4.7 Data Formats

A.4.7.1 CSV Import Format

CSV files should have the following structure:

- First column: Disease name
- Subsequent columns: Symptoms

Example:

```
Disease,Symptom1,Symptom2,Symptom3
Common Cold,Fever,Cough,Sneezing
Flu,HighFever,Fatigue,BodyAche
```

A.4.7.2 CLIPS Rule Structure

The system uses CLIPS rules in the following format:

```
(defrule Disease_Name
  (disease_is Disease_Name)
  =>
  (printout t "Disease Name" crlf)
)

(defrule is_it_Disease_Name
  (has_symptom Symptom1)
  (has_symptom Symptom2)
  (has_symptom Symptom3)
  =>
  (assert (disease_is Disease_Name))
)
```

A.4.8 Troubleshooting

- If you encounter file access issues on the backend, make sure the `data/` and `uploads/` directories exist and are writable.
- For Heroku deployment issues, check the logs: `heroku logs --tail`
- To clear the backend cache: Run `python clear_cache.py`