An Expert System for Medical Diagnosis Using Logic Programming and Knowledge Representation Techniques

Course Code: CSE440, *Section:* 1, *Group no:* 3
Group members: Wasiul Islam (172115042), Md. Jarif Mehtab Hossain (2111216642), Md. Sajidul Islam (2212663042), Imrul kays khan sovon (2221752042)

Abstract

As telemedicine use grows - expected to cover 25 - 30 % of U.S. medical visits by 2026 - the need for dependable, automated diagnostic tools has become urgent. Diagnostic errors remain common in primary care, where respiratory illnesses are often misdiagnosed because symptoms overlap. This paper describes the development of a Medical Diagnosis Expert System, a web based tool that supports decisions - mimicking clinician reasoning. Instead of opaque neural networks, the system uses a transparent, rule based inference engine built with Python's Experta library. It introduces a 3-tier headless architecture that separates the React frontend from the Python inference core through a Node.js layer. A key feature applies Shortliffe's Certainty Factor model, which handles vague user inputs like "70 % confident I have a fever" to produce confidence weighted diagnoses. Tests across varied symptom cases show the system reliably distinguishes viral illnesses (Influenza, COVID-19) from bacterial ones (Strep Throat, Pneumonia) using a dynamic, goal driven question sequence.

1. Introduction

Diagnostic accuracy forms the base of effective healthcare - yet it still presents a large problem. Research shows that doctors in primary care give a wrong diagnosis to about five out of every hundred U.S. adults each year - respiratory tract infections appear on the list of most frequent mistakes because their signs overlap. The swift spread of telemedicine now demands tools that sort patients before any contact with a physician.

Traditional symptom checkers usually depend on fixed decision trees or on machine learning models that hide their reasoning - such concealment blocks trust in medical AI. Rule-Based Expert Systems solve part of the problem - turning human knowledge into clear IF-THEN rules. Yet the classic versions meet two big obstacles:

The Uncertainty Problem: Medical facts are seldom yes-or-no. Patients describe symptoms at different levels of severity and with different levels of confidence.

The Accessibility Problem: Outdated expert systems often stay locked to a single desktop and will not run on the modern web.

This project closes those gaps through a Hybrid Medical Expert System. A strict Backward Chaining inference engine sits behind a modern, responsive web interface. The program targets six respiratory illnesses - Influenza, COVID-19, Common Cold, Strep Throat, Pneumonia besides Bronchitis. By adding Shortliffe's Certainty Factor Model, the system leaves rigid true false logic and copes with the vagueness found in real patient data.

2. Literature Review

2.1 The Legacy of MYCIN

The theoretical basis for the project is MYCIN, the first expert system built at Stanford during the 1970s to identify blood infections caused by bacteria. MYCIN split the Knowledge Base, which holds the rules, from the Inference Engine, which performs the reasoning. Its key innovation was the introduction of Certainty Factors, a practical stand-in for Bayesian statistics that demand complete prior probability data that medicine rarely supplies.

2.2 Uncertainty Management in AI

Medical reasoning always carries uncertainty. Bayesian networks give exact numbers - yet they need large tables of conditional probabilities that are hard to obtain. Deep learning reaches top accuracy in image recognition, but it works as a black box - it fails to show the steps required for clinical reasoning that depends on clear logic. Rule-based systems that use fuzzy logic or certainty factors stand between the two extremes - they reveal how each conclusion is reached and they accept inputs that are not simply true or false.

2.3 Modern Web Architectures

The move to "Headless" design in web work splits the screen that users see from the code that runs on the server. This project uses that split to bring the expert system up to date. We place the Python inference engine behind a Node.js API gateway. Because the parts no longer sit in one large block, the system

scales with ease and we keep it in good order.

3. System Architecture

The system implements a **3-Tier Headless Architecture**, ensuring robust separation of concerns. This design allows the computationally intensive inference logic to run independently of the user interface.

3.1 Tier 1: Frontend (Client Layer)

The user interface is a single page program built with React 18 and TypeScript - Vite bundles the code for speed.

State Management: Zustand stores the active sessionId, the chat history plus the short lived diagnosis data.

Uncertainty Input: A custom SymptomSlider turns the position of a visual slider into a float between 0.0 and 1.0 turning the user's sense of certainty into a numeric value called (CF_{user}).

3.2 Tier 2: Middleware (Orchestration Layer)

Node.js with Express is used for the backend. It works as a session manager, that turns the stateless HTTP protocol into something that can hold the state of a diagnostic interview.

Session Persistence: A class called SessionManager keeps a registry that pairs each session ID with a DiagnosisSession object.

Process Isolation: When a new user arrives, the backend starts a separate Python child process. Each process owns its own memory space - one user's facts never touch another user's facts.

Resource Optimization: A cleanup task watches every session. If a session sits idle for five minutes, the task ends the Python process tied to that session and frees the memory.

3.3 Tier 3: AI Engine (Inference Layer)

The core intelligence is a Python program that runs on the Experta library - Experta is a Python version of the CLIPS algorithm.

Knowledge Base: All domain expertise sits in two files - viral_rules.py holds the viral knowledge, bacterial_rules.py holds the bacterial knowledge.

Inference Mechanism: The engine applies Forward Chaining - it starts with the data and moves forward

until it links symptoms to diseases. For questions that aim at a specific goal, the engine runs a custom routine that mimics Backward Chaining.

4. Mathematical Model: Certainty Factors

A defining feature of the system is its use of the Shortliffe Certainty Factor model. This model lets the system reason with fuzzy data. A Certainty Factor (CF) is a number from -1.0 to +1.0, but the system uses only the range from 0.0 - 1.0.

4.1 Evidence Combination (AND Logic)

When a diagnostic rule depends on more than one symptom: The certainty of the combined evidence becomes = Certainty of the least certain symptom. For a rule R with antecedents - e_1, e_2, …, e□

CF_evidence = min(CF(e_1), CF(e_2), …, CF(e□))

It states the following principle: The strength of the chain equals the strength of its weakest link.

4.2 Rule Propagation

Evidence alone does not guarantee a diagnosis. The rule itself has built in reliability (CF_rule). For instance "If fever then flu" does not achieve perfect reliability even when fever is certain. The final confidence in the conclusion (H) equals the evidence confidence multiplied by the rule reliability: CF(H, E) = CF_evidence × CF_rule

4.3 Cumulative Evidence (Parallel Rules)

When two rules point to the same diagnosis, their certainties merge.

If Rule 1 gives the value CF_old and Rule 2 gives the value CF_new for the same disease, the merged certainty equals: CF_old + CF_new × (1 − CF_old).

The result moves toward 1.0 yet never passes it.

5. Algorithmic Logic: The Question Engine

To simulate a human doctor, the system must ask relevant questions rather than random ones. We

implemented a **Goal-Driven Information Gain Algorithm** in question_engine.py.

5.1 Information Gain Calculation

The system assigns a score to each unasked symptom. The score shows which symptom helps most with diagnosis.

The formula is: IG(s) = Priority(s) × (0.7 + 0.3 × D(s))

Priority(s) is the base value of the symptom.

Example: loss_of_smell = 10, headache = 4.

D(s): Discrimination Score

D(s) = 1 − | (diseases_with_s ÷ total_diseases) − 0.5 | × 2

This score favors symptoms that split the list of possible diseases close to even. Such symptoms cut uncertainty in the diagnosis space.

5.2 Dynamic Filtering

The engine filters as it runs. When the score for a viral infection rises above 0.3, it quietly drops questions that matter only to bacteria. It behaves the way doctors sort one cause from another.

6. Implementation Details

6.1 Knowledge Base Engineering

The medical knowledge appears as Python classes that inherit from experta.Rule
Viral Rules Module (viral_rules.py): This module sets rules for Influenza, COVID-19 and the Common Cold. The COVID-19 rule first pulls all evidence from loss of taste or smell then adds fever evidence.
Bacterial Rules Module (bacterial_rules.py): This module holds logic for Strep Throat, Pneumonia besides Bronchitis - it leans on Negative Indicators. For Strep Throat, lack of a cough counts as a strong positive sign.

6.2 Session Management Logic

The Node.js backend manages the lifecycle of the Python process. The following pseudocode demonstrates the spawning logic:

```
this.pythonProcess = spawn(pythonExecutable, [scriptPath]);
this.pythonProcess.stdout.on('data', (data) => {
    // Parse JSON stream from Python
    const result = JSON.parse(line);
    this.resolvePending(result);
});
```

7. Experimental Results

The system was validated using a comprehensive test suite. We present three case studies demonstrating the system's reasoning capabilities.

7.1 Case Study A: The "Classic" COVID-19

For the following patient's input example: Fever 0.8 certainty, Dry Cough with 0.7 certainty, and Loss of Smell with 0.9 certainty.

System Trace:

- Rule covid19_classic triggered.
- Evidence certainty factor equals the lowest of the three values: 0.7.
- Final certainty factor equals 0.7 multiplied by rule certainty factor 0.9: 0.63.
- Diagnosis: COVID-19 63 percent.
- Analysis: Loss of Smell carries high specificity - this symptom alone moved the diagnosis to COVID-19.

7.2 Case Study B: The Ambiguous Viral Case

For the following patient inputs: Fever 0.6 certainty, Fatigue with 0.7 and Runny Nose with 0.4 certainty.

System Trace:

- Rule influenza_moderate fires, CF 0.39
- Rule common_cold_mild fires, CF 0.26
- Diagnosis: Influenza 39%, Common Cold 26%

- Analysis: The system identified Influenza as the primary candidate because fatigue affects the whole body.

7.3 Case Study C: Bacterial Differentiation

For the following patient input - Sore throat with a certainty of 0.9, fever with 0.8, and cough with 0.1 certainty.

System Trace:

- Viral rules that demand cough did not fire.
- Rule strep_throat_classic tested the negative indicator Cough < 0.3. Requirement satisfied.
- Diagnosis: Strep throat 68%.

8. Discussion

8.1 Advantages

- **Explainability:** Every diagnosis has a clear path back to the exact rule that triggered it - nothing hides inside a black box.
- **Fuzzy Logic Handling:** The CF model turns vague patient statements into numbers that fit strict medical reasoning.
- **Architecture:** The headless design answers fast - each API call returns in under two hundred milliseconds.

8.2 Limitations

- **Static Knowledge Base:** The rules are fixed. To add a new domain, a programmer must write new code by hand.
- **Lack of Context:** The system stores no patient history - it holds no age, no gender, no list of other illnesses. Without that data, the diagnosis loses accuracy.

9. Conclusion

The project shows that a web based expert system for respiratory illness diagnosis works when it combines two parts. One part uses the Experta library, which applies strict logical rules. The other part uses the React framework, which lets users reach the system through a browser. Together they yield a tool

that remains both strong and simple to use. The next step will add Large Language Models that read medical texts and turn the contents into rules without manual effort.

10. References

1. E. H. Shortliffe and B. G. Buchanan, "A model of inexact reasoning in medicine," Mathematical Biosciences, vol. 23, no. 3-4, pp. 351-379, Feb. 1975.
2. D. Heckerman, "Probabilistic interpretations for MYCIN's certainty factors," in Uncertainty in Artificial Intelligence. Amsterdam, Netherlands: North-Holland, 1986.
3. "MYCIN | Expert System, Medical Diagnosis & Treatment," Britannica, Chicago, IL, USA. Accessed: Dec. 15, 2025. [Online]. Available: https://www.britannica.com/technology/MYCIN
4. "MYCIN: A Revolutionary AI System for Medical Diagnosis," schneppat.com. Accessed: Dec. 15, 2025. [Online]. Available: https://schneppat.com/mycin.html
5. "MYCIN: the beginning of artificial intelligence in medicine," Telefónica Tech Blog, Apr. 29, 2025. Accessed: Dec. 15, 2025. [Online]. Available: https://telefonicatech.com/en/blog/mycin-the-beginning-of-artificial-intelligence-in-medicine
6. N. P. Inter, "nilp0inter/experta: Expert Systems for Python," GitHub, Jun. 3, 2018. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/nilp0inter/experta
7. A. A. Olaniyi and O. M. Adewale, "Preliminary Differential Diagnosis of Pneumonia Disease," Asian Journal of Research in Computer Science, vol. 16, no. 3, pp. 1-12, Mar. 2025.
8. M. A. Razzaq et al., "Tuberculosis-Diagnostic Expert System: an architecture for case-based reasoning," SpringerPlus, vol. 3, Dec. 2014. doi: 10.1186/2193-1801-3-644.
9. S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build high-performance network programs," IEEE Internet Computing, vol. 14, no. 6, pp. 80-83, Nov.-Dec. 2010.