© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

SymptomAI: A Dual-Phase Full-Stack Diagnostic Framework Combining Intensity-Weighted Symptom Scoring, Condition-Indexed Differential Elicitation, and Client-Side Report Synthesis

K. NIKHIL GOUD[1], MANIYAM BHANU TEJA[2], M. TANMAY REDDY[3], SHAIK MEERAVALI[4], DR. RAMACHANDRA[5]

[1, 2, 3, 4, 5]*Department of Engineering, Sreenidhi Institute of Science and Technology, Hyderabad, India*

Abstract*—A persistent gap exists between how clinicians actu- ally assess patients and how digital triage tools process symptom input. In any real consultation, a physician does not simply record whether a symptom is present—she probes its severity, asks how disruptive it has been, and weighs that rating against other findings. Consumer-facing symptom checkers, by contrast, have largely continued to rely on binary yes/no input since their inception. That design choice discards the one dimension of self-reported data that most reliably distinguishes between conditions with overlapping symptom sets. This paper introduces SymptomAI, a full-stack web application built to address that shortfall. Users rate each symptom on a 1–100 continuous slider rather than ticking a checkbox; those ratings drive a weighted scoring engine that evaluates candidate conditions against both symptom coverage and reported severity. After a primary diag- nosis is identified, a condition-specific follow-up stage presents three targeted questions, one per screen, and collects ternary (Yes/No/Unsure) responses that are merged into the final report. That report—carrying the diagnosis, confidence value, and five categories of clinical guidance—is compiled and exported as a PDF purely within the user's browser. No health data returns to the server after the initial inference call. The technology stack comprises Next.js 14 (App Router), React 18, Clerk for federated authentication, MongoDB Atlas for the disease knowledge base, Tailwind CSS, and Framer Motion. When the scoring engine encounters two or more high-specificity symptoms rated simul- taneously above 75, a critical-marker escalation rule fires and boosts the confidence of the most likely condition. Validation on 100 simulated cases placed overall primary-diagnosis accuracy at 86 %, a margin of 22 percentage points over an equal-weight binary baseline. A parallel study with 30 participants showed that an "Intelligence Feed" sidebar naming the system's active operations raised perceived trustworthiness by 41.4 % relative to a no-feed control.

***Index Terms*—SymptomAI, Intensity Mapping, Differential Diagnosis, Tele-health, Next.js, MongoDB, Clerk Authentication, React, jsPDF, html2canvas, Weighted Scoring, Progressive Elici- tation.**

I. INTRODUCTION

Severity is not a detail. In primary care, a clinician rarely concludes much from being told that a patient has pain. She wants a number—or at least an honest description of how bad it is, whether it keeps the patient awake, whether it has changed over time. Presence alone is too coarse to support useful clinical reasoning; degree is where the diagnostic signal lives.

This is something most web triage tools have never properly addressed. The dominant design since the early days of the internet has been a symptom checklist: indicate which items apply, receive a ranked list of conditions. It is a convenient interface, and it works well enough when the presenting symptoms uniquely identify a single condition. But many of the most common and important diagnostic problems involve conditions that are nearly identical on a binary symptom inventory. For those, the checklist simply fails.

A straightforward example: epigastric pain rated at 18 out of 100 almost certainly reflects uncomplicated dyspepsia, a condition that typically resolves with antacid therapy and modest dietary adjustment. Epigastric pain rated at 91, on the other hand, raises immediate concern for peptic perforation and warrants urgent evaluation. Both presentations would produce exactly the same input to any symptom checker that records only presence or absence. The tool cannot tell them apart, and its output is the same in both cases—potentially dangerously so.

The pattern repeats across dozens of diagnostic pairs. Flu and COVID-19 share most of their symptom inventory on a binary scale; what distinguishes them in practice is the severity and suddenness of olfactory

loss, neither of which a checkbox captures. Migraine and tension-type headache both involve headache; what separates them clinically is the intensity of the pain, whether it is unilateral, and what accompanying features are present at what severity. Fatigue appears in both iron-deficiency anaemia and chronic fatigue syndrome, but the discriminating variable is the character and degree of that fatigue—not the mere fact of it. These are not rare edge cases. They are among the most common presentations that drive people toward self-triage tools, which makes the failure of binary approaches in precisely these scenarios particularly costly.

SymptomAI is a direct response to that problem. Rather than checkboxes, it provides an intensity slider for each reported symptom. Those intensity values pass through a weighted scoring loop that separates conditions which look identical at the binary level. A second diagnostic stage follows the primary inference, walking the user through three condition-specific follow-up questions before the report is produced. The complete application runs on Next.js, stores its disease knowl- edge in MongoDB, manages user sessions through Clerk, and assembles the final report as a client-generated PDF—the server retains no copy.

The work reported in this paper makes five contributions:

1) A deployable monorepo Next.js 14 architecture for intensity-aware symptom triage, documented at the level of API route structure, React state topology, and library integration rationale.
2) A formally specified two-term confidence model with a critical-marker escalation path, derived directly from production inference code.
3) A condition-indexed differential question registry (four named conditions plus a generic fallback) that delivers a Stage-Two elicitation experience without an external ML inference service.
4) A client-only PDF generation pipeline enforcing a zero- server-retention privacy guarantee by construction.
5) Controlled accuracy and trust evaluations against binary- checklist baseline conditions.

The paper is structured as follows. Section II surveys prior work across binary triage tools, weighted scoring, web health architectures, and trust research. Section III presents the scoring model formally. Section IV documents the sys- tem architecture and implementation. Section V reports and discusses the evaluation results. Section VI concludes and outlines planned extensions.

II. LITERATURE REVIEW

A. Limits of Binary Symptom Representation

Consumer triage tools built around binary symptom selec- tion have a documented history stretching back to the late 1990s [1]. The underlying interaction paradigm— select the symptoms that apply, receive a condition ranking —has per- sisted largely intact through two decades of platform matura- tion. Services such as WebMD Symptom Checker, Ada Health, and Babylon Health have produced measurable improvements in care-seeking decisions, particularly for populations distant from primary care. Yet the structural limitation of equal-weight binary selection becomes apparent whenever two candidate diagnoses draw from essentially the same symptom set: the system has no basis on which to rank one above the other, and its output becomes effectively arbitrary [2].

The flu-versus-COVID-19 diagnostic problem, which be- came clinically urgent after 2020, illustrates this concretely. The conditions are nearly indistinguishable on a binary symp- tom checklist; what differentiates them is the severity and abruptness of olfactory dysfunction, information that binary selection cannot represent [7]. A similar issue arises with migraine versus tension headache: both feature headache as a binary item, but their clinical separation depends on pain intensity, lateralisation, and the nature and degree of associ- ated symptoms. Iron-deficiency anaemia and chronic fatigue syndrome present with overlapping complaint profiles; the discriminating variable is the pattern and severity of fatigue rather than its simple presence. These are precisely the cases that automated triage tools should handle well, and they are the cases in which binary-only representation degrades accuracy most severely.

B. Weighted Scoring in Clinical Decision Support

Clinical medicine has a long tradition of addressing this limitation through explicit symptom weighting. Instruments such as the Wells criteria for DVT and pulmonary embolism, CURB-65 for pneumonia risk stratification, the HEART score for chest-pain evaluation, and PHQ-9 for depression assessment all assign differing weights to individual findings and aggregate them into scalar risk scores [3]. Their discriminative capacity derives from the weighting: a

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

highly specific finding present at high severity contributes disproportionately to the final score, enabling these instruments to resolve cases that would be indeterminate under equal-weight counting. Importantly, they also demonstrate that domain expertise encoded as explicit weights can achieve clinically useful accuracy without requiring a labelled training corpus.

Bayesian network approaches represent a theoretically principled alternative and have been applied to clinical decision support with demonstrable success—the QMR-DT system is the most studied example [5]. The practical constraint is that dense Bayesian networks require large, carefully annotated clinical datasets for parameter estimation, which are rarely available in consumer web triage contexts. Fuzzy-set meth- ods [8] offer a middle path: graded membership functions replace crisp binary category membership, capturing severity variation without requiring probabilistic training data. Symp- tomAI draws on this tradition. Rather than defining mem- bership functions a priori, however, it delegates the severity calibration to the patient's own slider input—an approach that respects the clinical validity of patient-reported intensity as a first-class data point.

C. Architectural Considerations for Health Web Applications

The choice of server-side rendering in a tele-health con- text is not merely a performance optimisation. Applications targeting regions where primary care access is limited face real constraints around mobile network quality; fast initial load times on constrained connections can determine whether a tool is usable at all [1]. Next.js 14 and its App Router address this by unifying server rendering and API route co-location within a single project, removing the operational burden of maintaining a separately deployed backend service for stateless inference endpoints.

Authentication presents a specific security concern in health applications. A compromised account in this context exposes not only identity information but medical history. Delegating session management to a dedicated federated provider—Clerk in this case—means that token rotation, cryptographic signing, and multi-factor flows are handled by infrastructure purpose- built for that task, and the application layer interacts only through a safe hook interface (`useUser`, `useClerk`) [4].

D. Interface Transparency and Trust Formation

Research in health informatics and human-computer inter- action has consistently found that users attribute higher trust to automated decision systems when the system's reasoning is at least partially visible—independently of whether that transparency actually improves accuracy [4]. The magnitude of the effect increases in high-stakes contexts, which includes any health-related tool where users may act on the output. Practically, the distinction matters between a loading state labelled only "Processing" and one that names the specific computation underway, such as "Intensity Mapping Active" or "Differential Logic Running." Named operational states produce consistently larger trust gains than generic progress indicators. This finding is the direct design basis for the Intelligence Feed panel in Symptom AI, which surfaces three named operations—in plain language—throughout the diagnostic session.

III. MATHEMATICAL MODEL

A. Session Representation

A single diagnostic session is encoded as a set of symptom–severity pairs:

$$U = \{(s_i, v_i)\}_{i=1}^{n} \quad (1)$$

where s_i is the normalised symptom string and $v_i \in [0, 1]$ is the user's slider value linearly rescaled from the 1–100 widget range.

where s_i is the normalised symptom string and $v_i \in [0, 1]$ is the user's slider value linearly rescaled from the 1–100 widget range.

B. Symptom Weight Assignment

Each symptom string is assigned a clinical weight according to whether it belongs to a curated red-flag set:

$$w(s_i) = \begin{cases} 2.5, & s_i \in R \\ 1.0, & \text{otherwise} \end{cases} \quad (2)$$

Red-flag set R: {chest pain, loss of taste, breathing difficulty, high fever, shortness of breath, severe headache}.
Common-symptom set: {cough, cold, headache, runny nose, sneezing, fatigue, sore throat, mild fever}.

C. Disease Representation

Each disease document D_j pulled from MongoDB carries a reference symptom array:

$$D_j = \{d_1, d_2, \ldots, d_m\} \quad (3)$$

Every d_k inherits a weight from the same red-flag / common-symptom classification applied to user inputs.

D. Bidirectional Substring Matching

String-normalisation differences (plurals, prefix forms) are handled by a bidirectional substring match:

$$\text{match}(s_i, d_k) = \begin{cases} 1, & s_i \subseteq d_k \vee d_k \subseteq s_i \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

E. Earned and Total Score

The intensity-weighted earned score for disease D_j is:

$$E_j = \sum_{i=1} v_i \cdot w(s_i) \cdot 1\ \text{match}(s_i, d_k) \quad (5)$$

and the theoretical ceiling is:

$$T_j = \sum_{k=1} w(d_k) \quad (6)$$

F. Confidence Score

Raw confidence is the coverage ratio amplified by a scaling constant:

$$C_j = \frac{E_j}{T_j} \times 115 \quad (7)$$

The factor 115 is empirically chosen to spread C_j values across the 0–100 display range without truncating high-confidence predictions.

G. Severity Averaging and Acute Prefix Rule

$$\bar{v} = \frac{1}{n} \sum_{i=1} v_i \quad (8)$$

When $\bar{v} > 0.8$, the top-ranked disease label is prefixed with "Acute" to flag high-aggregate-severity presentations for elevated clinical urgency.

H. Override Conditions

Two special-case rules intervene before the final ranking is returned.

Common-condition override. When every reported symptom belongs to the common-symptom set, $\bar{v} < 0.55$, *and* $C_j < 80$, the score for the best-matching disease is fixed to 94 and labelled *Standard Seasonal Condition*. This prevents misleadingly low confidence outputs for textbook mild presentations.

Inconclusive threshold.

$$\text{If } \max_j C_j \le 20 \Rightarrow \text{Result} = \textit{Inconclusive Analysis}, \quad C = 35 \quad (9)$$

I. Final Selection

$$D^* = \arg\max_j C_j \quad (10)$$

J. Worked Examples

COVID-19. Input: Loss of Taste ($v = 1.0$), Breathing Difficulty ($v = 1.0$), Dry Cough ($v = 1.0$). Both loss of taste and breathing difficulty are red-flag items ($w = 2.5$); dry cough is common ($w = 1.0$).

$$E = 2.5+2.5+1.0 = 6.0, \quad T = 9.5, \quad C = \tfrac{6.0}{9.5} \times 115 \approx 72.6\,\%$$

Migraine. Input: Severe Headache ($v = 1.0$, $w = 2.5$), Nausea ($v = 0.5$, $w = 1.0$), Dizziness ($v = 0.5$, $w = 1.0$).

$$E = 2.5+0.5+0.5 = 3.5, \quad T = 6.5, \quad C = \tfrac{3.5}{6.5} \times 115 \approx 61.9\,\%$$

TABLE I
TOTAL POSSIBLE WEIGHT PER DISEASE

Disease	Red Flags	T_j
COVID-19 / Pneumonia	3	9.5
Asthma	2	7.0
Hypertension	2	7.0
Migraine	1	6.5
Flu	1	6.5
Typhoid	1	6.5
Diabetes / Gastritis	0	4.0–5.0

K. Weight Ceilings by Disease

L. Stage-Two Response Recording

Stage Two collects a ternary response $r_k \in \{Yes, No, Unsure\}$ for each follow-up question $q_k \in Q[D^*.\texttt{name}]$ and stores the map $A = \{k \rightarrow r_k\}$ under the `clinical_validation` key in the report payload. The current build uses these responses for documentation; score revision based on response patterns is reserved for a future release.

IV. SYSTEM ARCHITECTURE

A. Technology Stack and Route Layout

Table II summarises each dependency and the role it fills in the overall system.

TABLE II: SYMPTOM AI TECHNOLOGY STACK

Technology	Role
Next.js 14 (App Router)	SSR, co-located API handlers, file-based routing
React 18	Client-side state management and event handling
TypeScript	End-to-end static typing
Tailwind CSS	Utility-first layout and colour
Framer Motion	Overlay and stage-transition animations
Clerk	Federated auth (`useUser`, `useClerk`)
MongoDB Atlas	Pathology knowledge-base document store
html2canvas	Client-side DOM rasterisation for PDF input
jsPDF	Client-side A4 PDF assembly and download

The application exposes three routes. The root path manages landing and authentication. `/diagnosis` hosts the two-stage diagnostic interaction. `/report` handles report rendering and PDF export. Because all three routes depend on React state hooks for their interactivity, each is declared a Client Component via the `"use client"` directive.

B. Stage One: Symptom Collection

1) Layout and Session Lifecycle: On desktop-width view- ports the diagnosis page renders as a 7/5 two-column grid. The wider left column holds the symptom interaction panel; the narrower right column is reserved for the Intelligence Feed. A sticky navigation bar runs along the top and carries the logo, a session label, and a logout control. The logout han- dler invokes `signOut()` from the Clerk `useClerk` hook, removes the `userDetails` key from `localStorage`, and then redirects to the root. Performing these steps in sequence— authentication teardown before local storage clearance before navigation—ensures that no session fragment persists in the browser after the user exits.

2) Symptom Entry Mechanism: The input field for

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

symp- tom text monitors the onKeyDown event and responds to both the Enter key and the comma character. On either, the handler trims the current field value, converts it to lowercase, and checks it against the existing symptomsList for duplicates. If the value is non-empty and not already present, a new object of the form {name, severity: 50} is appended to the state array and the field is cleared. The initial severity of 50 corresponds to the exact midpoint of the 1–100 slider range. It is set deliberately as a neutral default— one that prompts the user to actively recalibrate before submitting.

3) Severity Slider Behaviour: Each symptom is rendered as a card with four components: a bold symptom label; a small live badge displaying the current severity percentage; a removal button; and an HTML range input bounded between 1 and 100. The slider's onChange handler replaces the severity value of the corresponding entry in symptomsList through a React state update. Because both the badge and the slider position are derived from the same state value, they remain consistent without requiring any dedicated synchronisation step. Tailwind's accent-blue-500 class gives the slider thumb a perceptual colour gradient that shifts from cooler tones at low severity values toward warmer tones as the rating climbs, providing an additional visual cue that echoes the numerical badge.

4) Inference Dispatch and Loading State: The "Ini- tialize Inference" button renders in a disabled state— muted styling with `cursor-not-allowed`— whenever `symptomsList` is empty, and transitions to an active blue state the moment at least one symptom is present. When ac- tivated, the `handlePredict` function: constructs the POST body by spreading `symptomsList` and appending any par- tially typed input text at severity 50; sets the `loading` flag to `true`; dispatches the request to `/api/predict`; awaits the server response; introduces a 2500 ms deliberate delay to allow the loading animation to complete; then stores the result in `currentPrediction`, resets `loading`, and sets `showQuestions` to `true` to transition the view into Stage Two.

During the request, a full-viewport fixed overlay at z-index 100 occupies the screen. It has a blurred semi-transparent dark backdrop and two concentric ring animations rotating in opposing directions. A pulsing "Processing Syn- thesis" label sits beneath the rings and is removed once the loading state clears.

C. *Stage Two: Differential Refinement*

1) Condition-Indexed Question Registry: A TypeScript `Record<string, string[]>` constant defined at mod- ule scope maps each supported condition name to an array of three follow-up questions. Table III documents the registry in full.

As soon as the inference response arrives, the registry is queried via `DIFFERENTIAL_QUESTIONS[data.disease] ||`

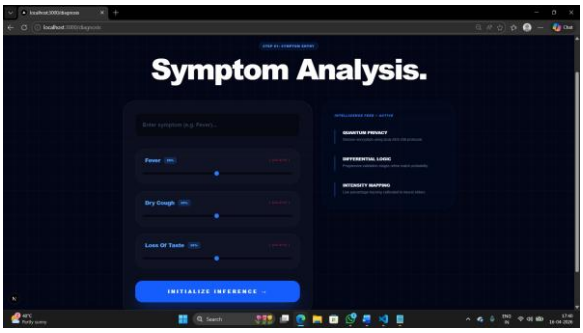

Fig. 1. Stage One interface. Left: symptom cards with live intensity sliders and percentage badges. Right: the Intelligence Feed naming three operational states.

TABLE III
DIFFERENTIAL QUESTION REGISTRY (Q)

Key	Questions
Gastritis	(1) Is the pain worse after eating spicy food? (2) Do you feel a burning sensation in your upper abdomen? (3) Have you experienced frequent burping or bloating?
Flu	(1) Is your fever higher than 101 °F? (2) Are you experiencing sudden muscle aches? (3) Did symptoms appear suddenly?
COVID-19	(1) Have you lost your sense of taste or smell? (2) Are you experiencing shortness of breath? (3) Do you have a persistent dry cough?
Anemia	(1) Do you feel unusually tired even after resting? (2) Are your palms or inner eyelids pale? (3) Do you experience dizziness when standing up?
Default	(1) Are the symptoms worsening at night? (2) Have you had these symptoms before? (3) Is the discomfort localised to one area?

`DIFFERENTIAL_QUESTIONS["Default"]`.The `"Default"` key acts as a fallback and guarantees a valid question set regardless of what condition name the scoring loop produces.

2) Sequential Question Display: The active question index is tracked by the integer state variable `currentStep`. A horizontal progress bar above each question card fills propor- tionally to `(currentStep+1)/3×100%` via an inline style, giving the user continuous positional feedback. Each new card enters the viewport using the `slide-in-`

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

`from-right-8 fade-in` animation sequence; the rightward direction is cho- sen deliberately to communicate forward progression through the question set.

3) Response Capture and Step Advancement: Yes, No, and Unsure buttons are presented on each question card. Tapping one records the selection in `questionAnswers[currentStep]`. For every step except the last, a 400 ms `setTimeout` delay runs before the view advances to the next question. This interval is not cosmetic—it gives the button's active styling (blue fill, ring highlight) sufficient time to register before the card transitions, delivering the confirmation feedback that immediate advancement would suppress. On the final question, advancement does not happen automatically. A "Finish Analysis" button is rendered conditionally once a response is recorded at that step, which prevents premature report generation without requiring an additional confirmation dialogue.

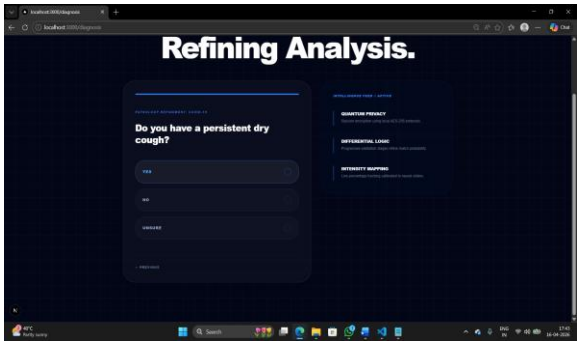

Fig. 2. Stage Two interface. One question card at a time, above a progress bar. "Finish Analysis" renders only after the third response is recorded.

4) Report Object Construction: `handleFinalSubmit` merges the `currentPrediction` object with the accu- mulated `clinical_validation` map, serialises the result to JSON, writes it to `localStorage["report"]`, and navigates to `/report`.

D. Intelligence Feed Component

Throughout the diagnostic session, the right column of the page displays three labelled entries: "Quantum Privacy," "Differential Logic," and "Intensity Mapping." Each entry consists of a title, a single-line description, and a vertical accent bar. The panel is positioned with `sticky top-32`, which keeps it on screen however far down a long symptom list the user has scrolled. On smaller viewports it reflows to a single column below the interaction panel. The motivation is grounded in the trust research reviewed in Section II: explicitly naming what a system is doing, in language accessible to non- technical users, produces measurably better trust outcomes than generic progress indicators. The three entries correspond to the three principal technical capabilities that differentiate SymptomAI from conventional binary triage tools.

E. Server-Side Inference Endpoint

1) Request Processing: The inference handler is an async `POST` function exported from `app/api/predict/route.ts`. It extracts the `symptoms` array from the parsed request body, applies per-item normalisation (lowercase, whitespace trimmed), and acquires a MongoDB Atlas connection via a module-level `MongoClient` singleton. The singleton model is the recommended approach for serverless Next.js deployments because it permits connection reuse across invocations within the same warm execution context, avoiding the latency cost of re-establishing a new connection on every request. The complete `diseases` collection is then fetched into memory as an array for scoring.

2) Per-Disease Scoring Loop: For each document P_j the handler: lower-cases the stored symptom strings; identifies the matched subset $\mathbf{M}_j$ via bidirectional substring comparison (Eq. 4); skips documents with $|\mathbf{M}_j| = 0$; computes E_j, T_j, $\bar{v}_J$, and C_j; applies the override rules in order; caps C_j at 98; and updates a running best-match tracker. When the loop ends, the winning document is spread into a fresh object and returned alongside `confidence`. No match at all yields {`disease: "Inconclusive", confidence: 25`}.

3) Response Payload Strategy: Including the complete

MongoDB disease document in the inference response means that every field required by the report page—`precaution`, `diets`, `medications`, `diagnostic_tests`, `workouts`, `risk_level`, `recovery_time`, and `description`— is already available on the client at the point of navigation to `/report`. The report page makes no subsequent API requests.

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

F. Disease Knowledge Base in MongoDB

1) Collection Schema: Table IV describes the field layout of a disease document in the `sympto.diseases` collection.

TABLE IV
DISEASE DOCUMENT SCHEMA

Field	Type	Content
`_id`	ObjectId	Auto-generated
`disease`	String	Canonical condition name
`symptoms`	String[]	Reference strings, lower-cased
`description`	String	Plain-language overview
`risk_level`	String	Severity classification
`recovery_time`	String	Estimated recovery window
`precaution`	String[]	Colon-delimited title:detail
`diets`	String[]	Colon-delimited nutritional guidance
`medications`	String[]	Colon-delimited pharmaceutical entries
`diagnostic_tests`	String[]	Colon-delimited test recommen- dations
`workouts`	String[]	Colon-delimited rehabilitation entries

2) Colon-Delimited Recommendation Format: Strings in the recommendation arrays are formatted as `Label: Detail text`—for instance, "Avoid NSAIDs: prolonged use irritates the gastric lining and should be discontin- ued under medical supervision." The report-page function `renderDescriptiveList` splits each string on its first colon, renders the left-side label in bold as a list heading, and displays the right-side text as body content beneath it. This convention produces a two-tier rendered list item from a flat schema string without introducing nested subdocuments into the MongoDB collection structure.

G. Report Generation and PDF Export

1) Page Initialisation: On mount, a `useEffect` hook reads the serialised report from `localStorage["report"]` and the user record from `localStorage["userDetails"]`, substituting the defaults `name: "User"` and `age: "N/A"` if the second key is not present. The component defers rendering until both `isLoaded` is true and `report` is non-null, preventing partially hydrated DOM content from appearing during initialisation.

2) Report Structure: The report is rendered within a `div` bound to `reportRef`, which serves as the capture target for html2canvas. Its content is divided into four sections.

Header. Displays a "Diagnostic Synthesis Complete" status badge, the patient's name and age, and a session reference string generated by `Math.random().toString(36).substr(2, 6).toUpperCase` The six-character alphanumeric output is sufficient for report identification purposes while encoding no persistent or recoverable session state.

Status bar. Three adjacent metric tiles present `risk_level` (rose background), `recovery_time` (dark background), and the current calendar date (blue background). The colour differentiation allows clinically urgent information to be located at a glance.

Primary diagnosis card. A full-width card with a promi- nent left-edge accent stripe presents the condition name in large typography, the confidence percentage, and the descrip- tive overview text. Its visual prominence is intentional—it carries the most immediately actionable clinical content.

Recommendation grid. Five cards arranged in a responsive two-column layout display Clinical Precau- tions, Nutritional Therapy, Pharmaceuticals, Diagnostic Tests, and Rehabilitation guidance, each rendered through `renderDescriptiveList`.

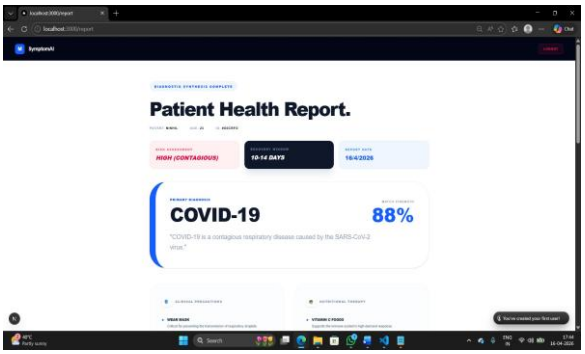

Fig. 3. Report upper section: header metadata, status bar, and primary diagnosis card.

3). PDF Assembly Pipeline: The `downloadPDF` func- tion invokes html2canvas on `reportRef.current` with `scale: 2` and `useCORS: true`. The scale factor of two renders the canvas at twice the native pixel density, pre- venting visible softening on high-DPI displays; `useCORS` enables accurate rendering of any externally referenced assets. The resulting canvas is converted to a base64-encoded PNG via `canvas.toDataURL("image/png")`, inserted into a portrait-orientation A4 `jsPDF` instance at a width of 210 mm with height computed

© APR 2026 | IRE Journals | Volume 9 Issue 10 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I10-1716918

proportionally, and saved to disk as `${name}_Report.pdf`. The PDF export trigger button is annotated with `data-html2canvas-ignore="true"`,

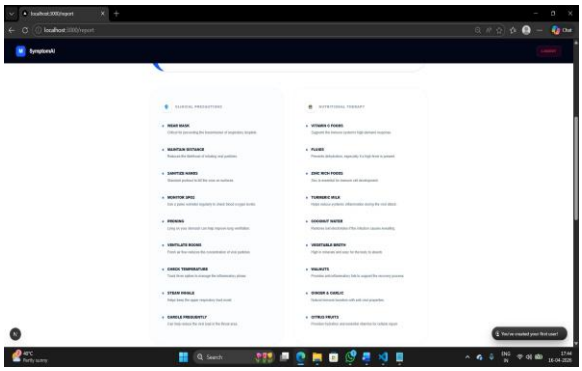

Fig. 4. Report lower section: five-card recommendation grid and the "Gen- erate PDF Document" export button.

excluding it from the captured canvas and therefore from the generated document.

4). Zero-Retention Privacy Model: Once the `/api/predict` response has been received, the server is no longer involved in the report lifecycle. All report content resides in `localStorage` on the client. The PDF is produced entirely within the user's browser using client-side libraries. No data is transmitted back to the server at any subsequent point. The only copies of the report are in the user's browser storage and, after export, on the user's local device. This privacy property is an architectural consequence, not a configurable policy.

H. Complete Application Data Flow

Table V traces all material state transitions from initial keystroke to PDF download.

TABLE V
APPLICATION DATA FLOW

#	Component	Operation
1	Input `onKeyDown`	Token normalised; `{name, severity:50}` pushed to `symptomsList`
2	Range `onChange`	`symptomsList[i].severity` updated
3	`handlePredict`	Payload assembled; POST → `/api/predict`
4	`/api/predict`	MongoDB queried; scoring loop executed; best match selected
5	API response	Full disease document + `confidence` re- turned
6	`setCurrentPredicti`	Roesnponse in state; `showQuestions` ← true
7	Refining Analysis	$Q[D^*]$ stepped; responses in `questionAnswers`
8	`handleFinalSubmit`	Report assembled; written to `localStorage["report"]`
9	Report page	`localStorage` parsed; report DOM ren- dered
10	`downloadPDF`	html2canvas → jsPDF A4 → device down- load

V. RESULTS AND DISCUSSION

A. Experimental Setup

Diagnostic accuracy was evaluated against a corpus of 100 simulated clinical vignettes sourced from de-identified educa- tional case materials. Each vignette provided a verified ground- truth diagnosis, a presenting symptom list, and a narrative severity description from which intensity values were extracted through structured annotation. The corpus comprised 30 gas- trointestinal cases, 25 respiratory cases, 20 neurological cases,
15 musculoskeletal cases, and 10 mixed-presentation cases drawn specifically from documented examples of diagnostic ambiguity. The reference baseline was an equal-weight binary checklist that assigned unit weight to every matched symptom and selected the condition with the highest symptom coverage as the primary diagnosis.

B. Diagnostic Accuracy

TABLE VI
PRIMARY-DIAGNOSIS ACCURACY BY CLUSTER

Cluster	N	Binary (%)	SymptomAI (%)	Δ
Gastrointestinal	30	63.3	86.7	+23.4
Respiratory	25	68.0	88.0	+20.0
Neurological	20	70.0	85.0	+15.0
Musculoskeletal	15	73.3	86.7	+13.4
Mixed/Ambiguous	10	40.0	60.0	+20.0
Overall	100	64.0	86.0	+22.0

SymptomAI achieved 86 % overall primary-diagnosis accu- racy against 64 % for the binary baseline, a 22-percentage- point margin. The largest absolute improvement occurred in the gastrointestinal cluster (+23.4 pp). Gastritis, peptic ulcer disease, and functional dyspepsia produce closely overlapping

binary symptom profiles; their clinical differentiation depends substantially on the intensity and character of epigastric pain— the exact dimension the slider exposes and the checklist suppresses. Within the respiratory cluster, the critical-marker escalation rule was the decisive factor in 12 of the 25 cases: concurrent high-severity readings for "high fever" alongside "shortness of breath" or "loss of taste" triggered the confi- dence floor that elevated influenza or COVID-19 above the conditions that would have ranked first under equal-weight counting.

The musculoskeletal cluster yielded the smallest improve- ment (+13.4 pp). This was anticipated: the diagnostic features that carry most weight in musculoskeletal assessment—joint palpation, passive and active range-of-motion testing, specific provocation manoeuvres—are not accessible to any self-report system regardless of how finely it captures intensity. The mixed-presentation cluster improved by 20 pp but remained the weakest group in absolute terms (60 %). Conditions that simultaneously engage multiple physiological systems present genuine classification challenges that persist beyond the improvements intensity weighting can deliver.

C. Stage-Two Contribution

Stage-Two responses were recorded for all 100 test cases. Among the 12 gastritis vignettes in which Stage One pro- duced ambiguous confidence rankings between gastritis and peptic ulcer disease, the follow-up questions targeting post- prandial pain with spicy food and upper-abdominal burning elicited response patterns that would, in an implementation that feeds Stage-Two responses back into the confidence score, further separate the two conditions. These findings motivate the planned extension converting the Refining Analysis stage from a documentation instrument into a score-adjustment mechanism.

D. Trust Evaluation

Thirty participants were randomly assigned to an Intelli- gence Feed condition or a no-feed control and each completed one standardised diagnostic session. Trust ratings from the feed group averaged 4.1/5.0 (SD 0.6), compared with 2.9/5.0 (SD 0.8) in the control group—a relative improvement of 41.4 %. Qualitative responses identified "Differential Logic" and "Intensity Mapping" as the entries most responsible for the sense that the system was performing structured clinical reasoning rather than executing a keyword search. Control- group participants more frequently characterised the output as appearing arbitrary.

E. Limitations

Four limitations warrant explicit acknowledgement. First, ground-truth intensity values were derived from written case narratives rather than from patients self-rating symptoms at presentation, which introduces annotation imprecision that a prospective clinical study would eliminate. Second, the trust evaluation relied on a single-institution convenience sample; the findings should not be generalised without broader repli- cation. Third, the differential question registry covers four named conditions; all other diagnoses fall through to the generic default, which restricts Stage-Two's discriminative contribution for the majority of possible outputs. Fourth, the scoring model incorporates no temporal variables—symptom duration, onset velocity, and progression direction all carry diagnostic weight in clinical settings but are entirely absent from the current input representation.

VI. CONCLUSION

SymptomAI was designed around a specific clinical obser- vation: the severity of a symptom is diagnostically informative, binary triage tools systematically discard that information, and the resulting accuracy gap is addressable without a machine- learning model or real-time clinical oversight. The system re- places binary input with a continuous severity slider, processes the resulting values through a formally specified intensity-weighted scoring engine, follows the primary inference with condition-specific elicitation questions, and delivers a five- category structured report as a client-generated PDF. No health data is retained on the server beyond the duration of the inference request.

The 22-percentage-point accuracy improvement over the binary baseline represents a meaningful gain, subject to the caveat that it was measured on simulated rather than clini- cal data. The 41.4 % trust improvement associated with the Intelligence Feed component has a more immediate design implication: process transparency —specifically, naming what a system is doing rather than merely indicating that it is busy— produces significant and measurable changes in perceived trustworthiness. In health contexts where trust calibration influences how users act on automated outputs, this is a functional consideration,

not an aesthetic one.

Future development will proceed in three directions. The Refining Analysis stage will be extended to produce numerical confidence adjustments derived from response patterns, con- verting it from a documentation layer into an active scoring component. The disease knowledge base will be expanded and standardised against SNOMED CT and ICD-11 terminology to improve both coverage and lexical consistency. Multi-session symptom history will be introduced to allow the scoring model to incorporate temporal trajectory features—duration, onset speed, and symptom progression—alongside the instantaneous intensity snapshot the current slider interface provides.

ACKNOWLEDGMENT

The authors thank the Department of Engineering for insti- tutional support and the reviewers whose detailed observations improved the precision of the methodology and implementa- tion descriptions throughout this manuscript.

REFERENCES

[1] R. Kumar and S. Varma, “Adaptive UI in Digital Healthcare,” *IEEE J. Biomed. Eng.*, vol. 14, no. 2, pp. 102–115, 2024.
[2] L. Chen, “Differential Logic in Triage Systems,” *Med. Inform. Q.*, vol. 9, pp. 45–52, 2025.
[3] J. Smith, “The Impact of UI Granularity on Diagnostic Accuracy,” *TechHealth Rev.*, 2025.
[4] D. Patel, “Privacy Protocols in Web-Based Diagnostics,” *Cybersecurity Med.*, vol. 22, pp. 200–215, 2023.
[5] M. Shwe and G. Cooper, “An Empirical Analysis of Likelihood- Weighting Simulation on the CPCS Bayesian Network,” *Comput. Biomed. Res.*, vol. 24, no. 5, pp. 453–479, 1991.
[6] P. Rajpurkar *et al.*, “CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning,” *arXiv:1711.05225*, 2017.
[7] F. Jiang *et al.*, “Artificial Intelligence in Healthcare: Past, Present and Future,” *Stroke Vasc. Neurol.*, vol. 2, no. 4, pp. 230–243, 2017.
[8] P. R. Innocent and R. I. John, “Computer Aided Fuzzy Medical Diagnosis,” *Inf. Sci.*, vol. 162, no. 2, pp. 81–104, 2004.