Symptom Classifier

Balașcă Diana-Loredana, Burcă Alina, Maxim Maria-Valeria, Mihai Răzvan-Ionuț, Neculau Eduard-Ștefan, Pătrășcan Andrei-Alexandru

Abstract:

The integration of artificial intelligence[8] (AI) into the field of medical diagnostics represents a revolutionary advancement in healthcare. AI diagnosticians, powered by sophisticated algorithms and vast datasets, are transforming how diseases are detected and diagnosed, offering potential improvements in accuracy, efficiency, and accessibility. One of the primary advantages of AI in medical diagnosis is its ability to analyze and interpret large volumes of data rapidly. Traditional diagnostic processes, which rely heavily on human expertise, can be time-consuming and prone to errors. In contrast, AI systems can process complex medical data, including medical imaging, laboratory results, and patient histories, in a fraction of the time.

This project aims to develop a web application that provides support for individuals who want to better understand what health issues they might be facing, based on the symptoms they describe freely through a chat-style interface. The user writes, in their own words, how they feel (for example: “I have a cough and muscle aches”), and the application automatically analyzes the message using modern natural language processing technologies. The system identifies the key symptoms in the text, compares them with information from a medical database, and provides a list of possible diseases. The application offers clear and easy-to-understand explanations of the results. The application integrates several key components: a user-friendly chat interface, an intelligent text analysis system, a symptom classification engine and a module that suggests diagnoses and generates personalized recommendations. While the application does not replace a medical consultation, it can assist users in making a quicker and more informed decision.

Keywords: medical chatbot, symptom analysis, natural language processing, disease classification, machine-generated diagnosis

1. Introduction

1.1. Purpose

The purpose of this project is to help people who cannot immediately see a doctor but need medical advice. Many people have symptoms and don't know what to do or how serious these symptoms are. Our system will be a place where anyone can describe how they feel and get simple suggestions about what they might have and what they should do next, reducing worry and helping patients know whether they need to see a doctor urgently or not.

1.2. Product Scope

Our solution is a web application that automatically analyzes symptoms described by patients. The application will read people's descriptions of how they feel, identify important symptoms from the text, compare them with medical databases, and suggest possible health problems. Users will receive suggestions about what diseases they might have and what they should do - take simple medication, go to the pharmacy, consult a doctor, or go to the emergency room. The application has an easy-to-use interface for everyone. It's important to know that this application does not replace a doctor's visit, but only offers a first opinion and helps people decide if and how quickly they should see a medical specialist.

2. Applied Technologies

2.1. React (Frontend Framework)

React[9] is a JavaScript library for building user interfaces, particularly single-page applications. In the context of the symptom classifier project, React facilitates a dynamic and responsive interface that allows users to input symptoms and receive predictions quickly.

2.2. FastAPI (Backend Web Framework)

FastAPI[10] is a modern, high-performance web framework for building APIs with Python. It plays a crucial role in the project as the backbone of the backend RESTful API, which handles communication between the frontend and the natural language processing modules. FastAPI is chosen for its speed, asynchronous support, and automatic validation of request/response data, ensuring that user-submitted symptoms are processed securely and efficiently.

2.3. Teprolin (NLP Processor API)

Teprolin is a Romanian-language Natural Language Processing API that provides lemmatization and biomedical named entity recognition (BioNER). In this project, Teprolin is essential for preprocessing the user's symptom descriptions. It standardizes inputs by reducing words to their lemma form (e.g., "febra" and "febră" → "febra") and identifies medical-related terms in the text. This semantic normalization ensures that the symptom data is clean, consistent, and suitable for further analysis and classification.

2.4. Gemini (External Large Language Model API)

Gemini[11] is an external Large Language Model (LLM) API used in the project to enhance understanding and contextual interpretation of symptom descriptions. After initial processing by Teprolin, Gemini can help refine or expand the interpretation of symptoms, enabling the classifier to consider broader medical knowledge and provide more accurate or nuanced predictions.

2.5. RESTful API Architecture

The system is designed using a RESTful architecture, which ensures that the interaction between frontend, backend, and third-party APIs remains stateless, modular, and easily testable. Each component communicates using standardized HTTP methods, promoting interoperability and scalability.

2.6. Concurrency and Asynchronous Processing

To maintain responsiveness and minimize latency, the backend leverages asynchronous programming supported by FastAPI. This is especially important when multiple API calls (e.g., to Teprolin and Gemini) need to be made in parallel. Asynchronous processing ensures that the server can handle multiple user requests concurrently, which is crucial in a real-world web environment.

3. System Architecture

3.1. System architecture diagram

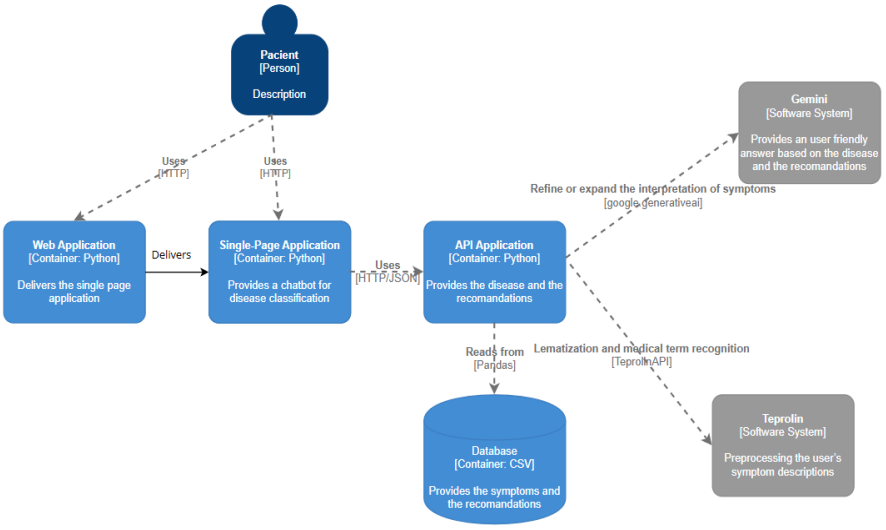

Fig. 1. *System architecture diagram*

3.2. Frontend

The user interface is designed to be simple, clean, and easy to use. Users can enter their symptoms in a chat-like input field. Once the message is sent, the system responds with detailed medical suggestions based on the input. The layout is minimal, with a calming visual theme that ensures readability and comfort for all types of users, including those with limited technical experience.

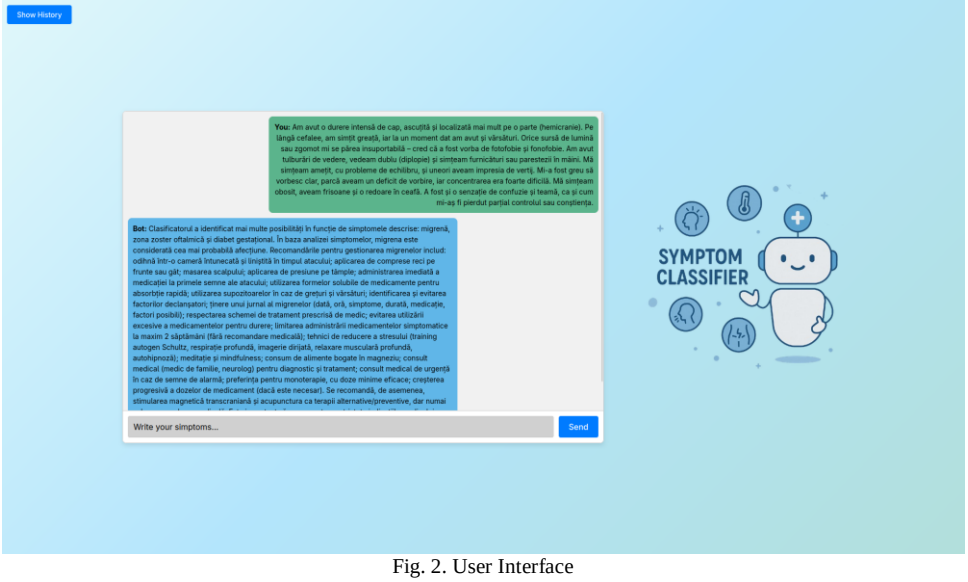

Fig. 2. User Interface

3.3. Backend

The backend component of the Symptom Classifier plays a central role in bridging the frontend interface with external processing services and machine learning modules. It receives symptom input from the frontend React application, forwards it to the **Teprolin NLP API** for biomedical tokenization and named entity recognition, then uses a local **classifier module** to determine the most likely disease(s). Finally, it enhances the user experience by fetching corresponding medical recommendations from a CSV file and using the **Gemini LLM API** to generate a user-friendly diagnostic message. All these operations are exposed via a RESTful interface built with FastAPI.

This component communicates directly with:

- The **frontend React application**, which sends user input and receives results
- The **Teprolin API**, for preprocessing and tokenization
- A **classifier module**, to predict the most probable disease
- The **Gemini LLM API** [12], for generating explanations and recommendations
- A **CSV file**, as a lightweight database for storing disease recommendations

3.4. The TEPROLIN Web Service

In this subsection the interaction between the application and the TEPROLIN Web Service is covered.

The TEPROLIN Web Service (WS)[4] was developed and is maintained in the ReTeRom project. The backend is the TEPROLIN text preprocessing platform that incorporates several NLP applications for which it provides a unified access interface as a Python 3 object.

TEPROLIN currently offers 15 text preprocessing operations for Romanian, 13 of which are described in (Ion, 2018). Our application uses the tokenization, lemmatization and the biomedical-named-entity-recognition operations.

Tokenization [5] is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning. The first step of the NLP process is gathering the data (a sentence) and breaking it into understandable parts (words).

Lemmatization [6] is the process of grouping together different inflected forms of the same word. It's used in computational linguistics, natural language processing (NLP) and chatbots. Lemmatization links similar meaning words as one word, making tools such as chatbots and search engine queries more effective and accurate.

The goal of lemmatization is to reduce a word to its root form, also called a *lemma*. For example, the verb *running* would be identified as *run*. Lemmatization studies the morphological, or structural, and contextual analysis of words.

Named entity recognition (NER) [7] — also called entity chunking or entity extraction—is a component of natural language processing (NLP) that identifies predefined categories of objects in a body of text.

These categories can include, but are not limited to, names of individuals, organizations, locations, expressions of times, quantities, medical codes, monetary values and percentages, among others. Essentially, NER is the process of taking a string of text (i.e., a sentence, paragraph or entire document), and identifying and classifying the entities that refer to each category.

The application makes use of a biomedical-NER to recognize and extract medical terms, a process that is essential for the process of classifying a list of symptoms into a diagnostic/disease.

4. Implementation Details

4.1. User Interaction and Data Submission

The user interface features a chat-style interaction between the user and a symptom classifier backend service. The user interface consists of a text input field where the user can input their symptoms in natural language and a a chat window that displays the ongoing conversation.

When the user submits a message, the input is sent to the backend via an HTTP POST request with a JSON body containing the user input. The response from the backend is parsed on the frontend and displayed in the chat window, below the user's message.

4.2. Symptom Processing and Classification Flow

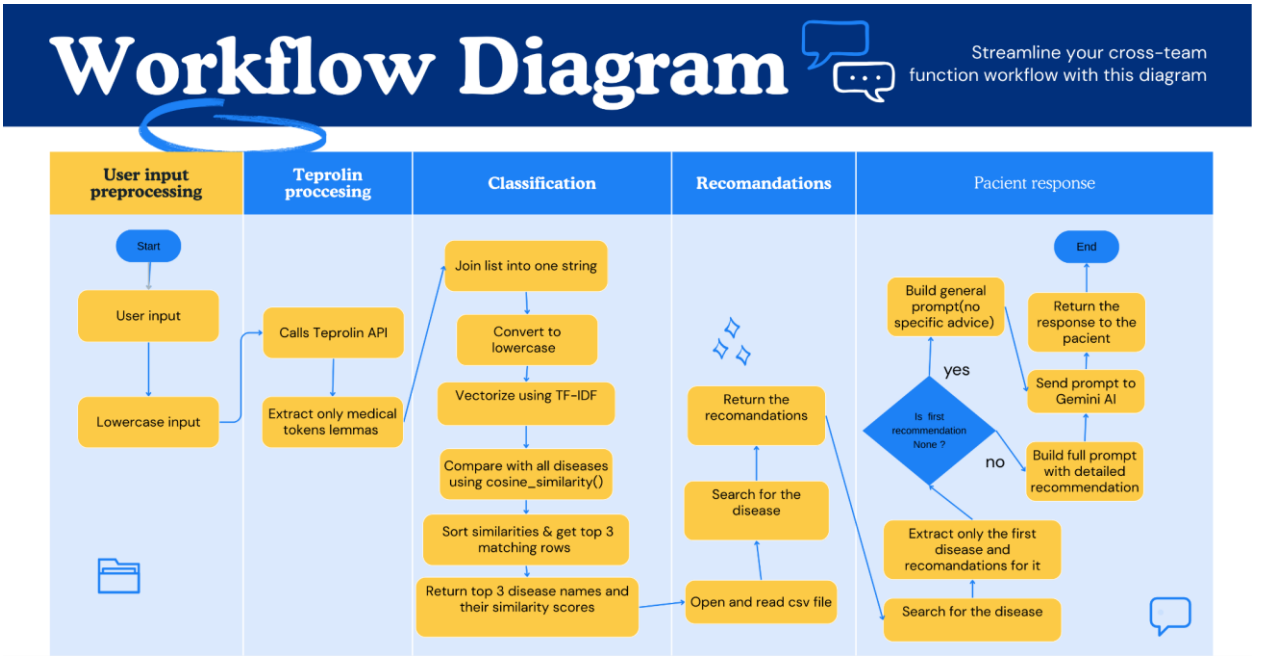

Fig. 3. Workflow Diagram

The backend of the Symptom Classifier web application serves as the core processing engine. It handles user input, coordinates with external services, and assembles personalized diagnostic responses. The processing pipeline is modular, allowing for clear separation of concerns and easy extensibility.

When a symptom description is submitted by the user, the backend begins processing it through the following key stages:

4.2.1. Preprocessing with NLP (Teprolin Integration)

The user input is first sent to an external Romanian language processing API that specializes in biomedical named entity recognition. The input text is tokenized and lemmatized, and only tokens marked as relevant medical entities are kept. This process ensures that only medically meaningful terms are passed on to the classification system.

4.2.2. Pre-classification

After extracting the relevant words from the user's message using TEPROLIN, the system proceeds to the semantic matching stage, where the identified symptoms are compared against a predefined medical database. This medical database is represented by a CSV file that contains, for each disease, an associated list of characteristic symptoms.

For each extracted symptom, the system performs a comparison with the entries in the CSV file, identifying the diseases that show the highest degree of similarity with the set of symptoms provided by the user. A matching score is computed for each disease, based on the number and relevance of shared symptoms.

Following this analysis, the top three conditions with the highest matching scores are selected and presented to the user in descending order (ranks 1, 2, and 3). This ranking provides a quick and intuitive overview of the most probable diagnoses suggested by the system.

In addition, for each of the identified diseases, the user receives a personalized recommendation automatically generated based on the symptom description and the characteristics of the condition. These recommendations include essential information regarding the next steps, thus offering a preliminary action guide in relation to the potential illness.

4.2.3. Disease Classification

After obtaining the cleaned list of medical terms, the system uses a classification component to determine the most probable disease(s) that match the described symptoms. **Cosine similarity** is a measure of **how similar two vectors are**, **regardless of their magnitude**, by comparing the angle between them. It focuses on the **orientation** (direction) of the vectors, not their magnitude — meaning it compares **what words are used**, not how long the input is.

Formula:

$$S_C(x, y) = \frac{x \cdot y}{|| x || \times || y ||} \quad (1)$$

- $x \cdot y \rightarrow$ product (dot) of the vectors 'x' and 'y'.
- **||x||** and **||y||** → length (magnitude) of the two vectors 'x' and 'y'.
- **||x||** × **||y||** → regular product of the two vectors 'x' and 'y'.

In our application , each disease has a text description of symptoms, the user provides their symptoms, we use the TEPROLIN API to select only the medical terms from the user's input and all texts are converted into vectors using **TF-IDF**.Cosine similarity is used to **compare the user's symptoms to each disease description**, and the most similar diseases are selected.

Table 1. The meaning of the Cosine Similarity factor

Cosine Similarity	Meaning
1.0	Identical direction (most similar)
0	Orthogonal (no similarity)
-1.0	Opposite direction (least similar)

Cosine similarity[13] is a value that ranges from -1 to 1, where 1 indicates the vectors are identical and perfectly aligned (with no angle between them), 0 indicates the vectors are orthogonal (90 degrees to each other) with no match and -1 indicates completely opposite vectors (with a 180 degree angle between them).

4.2.4. Medical Recommendation Lookup

Once the disease(s) are identified, the backend loads medical advice and suggestions from a local CSV file that stores disease-to-recommendation mappings. For each matched disease, the system searches for any available guidance. If no recommendation is found, a default fallback message is prepared.

4.2.5. Generating a Human-Friendly Response (Gemini Integration)

To improve user understanding, the backend composes a natural-language explanation using a large language model. The prompt is carefully crafted to ensure:

- A formal, coherent tone suitable for academic or health-related contexts
- Clear communication of matched diseases
- Focused recommendations only for the most probable disease
- No inclusion of non-medical advice

The language model generates a personalized summary based on the classification and available recommendations, returning a concise and informative message tailored to the patient.

4.2.6. Returned Output

The backend returns a structured JSON response to the frontend, which includes:

- The generated response text, intended for direct display to the user
- The similarity factor, which may be used for additional UI logic (e.g., displaying confidence indicators)

References

[1] Universitatea „Alexandru Ioan Cuza” din Iași. (n.d.) “Resources – Tehnici de Prelucrare a Limbajului Natural.” https://edu.info.uaic.ro/tehnici-prelucrare-limbaj-natural/resources.html

[2] Universitatea „Alexandru Ioan Cuza” din Iași. (n.d.) “News – Tehnici de Prelucrare a Limbajului Natural.” https://edu.info.uaic.ro/tehnici-prelucrare-limbaj-natural/news.html

[3] Scikit-learn developers. (2019) “Scikit-learn Documentation v0.21.” https://scikit-learn.org/0.21/documentation.html

[4] Ion, Radu. (n.d.) “The TEPROLIN Web Service.” Institutul de Cercetări pentru Inteligență Artificială, București. Contact: radu@racai.ro

[5] IXOPAY. (2022) “What is NLP? Natural Language Processing & Tokenization.” https://www.ixopay.com/blog/what-is-nlp-natural-language-processing-tokenization

[6] TechTarget. (n.d.) “Lemmatization – Definition & Explanation.” https://www.techtarget.com/searchenterpriseai/definition/lemmatization

[7] IBM. (n.d.) “Named Entity Recognition – Overview.” https://www.ibm.com/think/topics/named-entity-recognition

[8] Prefer Publishing. (2023) “AI and Natural Language Processing.” *Kindle Series*, Vol. 38. https://preferpub.org/index.php/kindle/article/view/Book38

[9] React Developers. (2023) “Creating a React App – React Documentation.” https://react.dev/learn/creating-a-react-app

[10] FastAPI Contributors. (n.d.) “FastAPI Documentation.” https://devdocs.io/fastapi/

[11] Google. (2023) “Google AI – Official Website.” https://ai.google.dev/

[12] Google. (2024) “Gemini API – Documentation.” https://ai.google.dev/gemini-api/docs

[13] Built In. (2023) “What is Cosine Similarity?” https://builtin.com/machine-learning/cosine-similarity