National Conference on Behavioural Economics and Intelligent Decision Systems for Climate Change and Sustainable Development
Journal of Advanced Research in Embedded System
Volume 13, Issue 1&2 - 2026

Research Article

AI-Powered Disease Diagnostic Predictive Model using Neural Networks

Harmanpreet Singh[1], Anureet Kaur[2]

[1]Student, [2]Assistant Professor, Khalsa College, Amritsar, India
DOI: https://doi.org/10.24321/2395.3802.202605

INFO

Corresponding Author:
Harmanpreet Singh, Khalsa College, Amritsar, Assistant Professor, Khalsa College, Amritsar, India
E-mail Id:
harmanpreetsinghgill13@gmail.com
Orcid Id:
http://orcid.org/0000-0002-8698-1633
How to cite this article:
Singh H, Kaur A. AI-Powered Disease Diagnostic Predictive Model using Neural Networks. *J Adv Res Embed Sys* 2026; 13(1&2): 262-268.

Date of Submission: 2026-10-04
Date of Acceptance: 2026-10-28

ABSTRACT

The digital transformation of healthcare has seen an increasing reliance on Artificial Intelligence (AI) and Machine Learning (ML) technologies to support diagnostic and decision-making processes. In a world where access to healthcare services remains uneven—especially in remote or economically underdeveloped regions—technology is being leveraged to fill the accessibility gap. The rise of intelligent applications has not only enhanced the accuracy of diagnostics but has also enabled faster, scalable, and cost-effective solutions. This research introduces a lightweight neural network web application that identifies patterns within symptoms to suggest a probable disease. With a main focus on accessibility, affordability, and adaptability, the core of this system is a deep learning model trained on a dataset consisting of 5,000 symptom-disease mappings covering 41 unique diseases. The model with regularisation has achieved an outstanding 96.54% accuracy. The neural network with users, whether healthcare professionals or general individuals, can interact with the application to input symptoms and receive a disease prediction within seconds. This serves as an initial assessment tool, prompting users to seek professional advice if necessary. The system is designed to be lightweight using TensorFlow Lite, making it deployable even on low-end devices. It is hosted online to ensure ease of access and is free of cost, promoting inclusivity. The incorporation of a feedback mechanism—where users can correct wrong predictions—adds another layer of intelligence by laying the groundwork for reinforcement-based learning in future versions.

Keywords: Healthcare, Accessible, Light-weighted, Neural Network, AI

Introduction

Modern healthcare systems are often challenged by the growing demand for accessible and timely medical care. In countries with large populations or limited medical infrastructure, patients may face long waiting times for consultations or misdiagnosis due to overloaded healthcare providers. Additionally, the lack of access to medical professionals in rural or remote areas further exacerbates the problem. In this context, AI-based diagnostic assistants offer an innovative solution. However, many existing tools are either commercially restricted, too complex for laypeople to use, or lack adaptability. This research seeks to address these shortcomings through the development of an AI-driven medical assistant that accurately predicts diseases based on symptoms, freely accessible, lightweighted and fast, feedback-enabled, and scalable. The problem this research tackles can be summarized by the following core questions:

- Can AI be used to make accurate predictions about diseases based solely on symptom input?
- Can such a system be made accessible and intuitive enough for public use?
- Can feedback from users be used to improve and adapt the model over time?

Neural Networks

A neural network is a machine learning system designed to mimic the organisation of the human brain. It consists of interconnected layers of nodes known as neurones. The input layer receives raw data, and the output layer produces the final output. Computation occurs in the hidden layers, which transform the input into the output.[1] A neural network's identification of patterns in data by adjusting the weights of connections between neurones is the most important objective of using it. During training, the network receives a set of labelled samples and updates its weights based on the differences between predicted and actual output. After training, the network can be used to make predictions or decisions using new input data.[2]

Neural networks have a number of applications in various domains, such as robotics, natural language processing, speech recognition, image recognition, and many more. They have an excellent performance in tasks involving complex and multidimensional data, such as visual or auditory information. It is challenging and impossible for traditional machine learning approaches to extract the features in data that a neural network can extract by recognising patterns in data. Common types of neural networks include feedforward neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs).[3] Feedforward networks are among the simplest types and are often used for classification tasks. RNNs, on the other hand, are well-suited for natural language processing tasks involving sequential input, such as text or speech. Moreover, CNNs are employed for tasks related to image recognition and classification.[4]

Flask

Flask is a lightweight and flexible web framework written in Python for facilitating the rapid development of web applications. It follows the micro-framework philosophy, meaning it provides essential tools for building web applications while leaving developers free to choose additional extensions as needed. Its simplicity, combined with extensive community support and a wide ecosystem of extensions, has made Flask a popular choice among developers for creating APIs and full-stack applications (Grinberg, 2018).[5] This research leverages Flask to build the backend of the Disease Diagnostic System.

Streamlit

Streamlit is an open-source Python library that enables developers and researchers to create interactive and data-driven web applications with minimal effort. Unlike traditional web frameworks, Streamlit is designed specifically for data science and machine learning workflows, allowing users to turn Python scripts into shareable web apps with just a few lines of code. Its simple syntax, widget integration, and real-time data visualisation make it a valuable tool for rapid prototyping and presenting analytical results. Streamlit emphasises ease of use and integration with popular Python libraries like NumPy, Pandas, and TensorFlow (Streamlit Inc., 2022).[6]

Related Work

Predicting diseases using neural networks in the healthcare field has been a subject of extensive research. A number of studies have explored the application of neural networks in disease prediction, demonstrating promising results. According to Ashfaq et al.[7], a convolutional neural network (CNN) was employed to predict the development of diabetic retinopathy. The model achieved high accuracy in diagnosing the disease based on retinal images. Another study done by Dong et al.[8] focuses on the importance of targeted interventions for high-risk patients to reduce hospital readmissions and healthcare costs. The researchers propose a deep learning framework that combines both human and machine-derived features in a sequential manner using a cost-sensitive LSTM model to predict the risk of readmission, achieving an area under the curve (AUC) of 0.77. The incorporation of sequential trajectories had the most significant impact on the prediction performance, contributing a 26% improvement, followed by the inclusion of expert features alongside machine-derived features, which added a 3% improvement. The study also presents heatmaps that demonstrate substantial cost savings when targeted interventions are provided to high-risk patients. These findings emphasise the potential of the proposed deep learning model in identifying patients at risk of readmission, allowing healthcare providers to allocate appropriate resources and interventions, thereby improving patient outcomes and reducing healthcare costs. The researchers in Ali et al.[9] developed a model that achieved a high level of success in predicting Stage 2/3 acute kidney injury (AKI) before its detection using conventional criteria, with a median lead time of 30 hours and an area under the receiver operating characteristic (AUROC) curve of 0.89. It accurately predicted 70% of subsequent renal replacement therapy (RRT) episodes, 58% of Stage 2/3 AKI episodes, and 41% of any AKI episodes. The ratio of false alerts to true alerts for any AKI episodes was approximately one-to-one, indicating a positive predictive value (PPV) of 47%. Notably, among the patients identified as at risk by the model, 79% received potentially nephrotoxic medication after being flagged by the model but before the development of AKI. These results demonstrate the effectiveness of the model in early detection of AKI and its potential to guide timely interventions and prevent further complications.

Design and Methodologies

The AI-Powered Disease Diagnostic System is built upon a robust and modular software design. The system's architecture integrates multiple layers—from the user interface to the deep learning model and feedback management system—each playing a crucial role in disease prediction. In this section, we describe the software design, outline the primary algorithms used, and present representative code snippets that exemplify the system's functionality.

Architecture

The system is divided into several interrelated modules. The Frontend Web Interface module, which is developed using Streamlit, is responsible for gathering user input and presenting predictions. It also provides a mechanism for feedback. The backend REST API module is implemented with Flask. The backend acts as a mediator between the frontend and the machine learning model. It processes requests, performs data transformations, and handles predictions. The deep learning model, which is the core of the system, is a neural network built with TensorFlow/Keras. This model predicts diseases based on input symptoms. In the Feedback and Database Layer, user feedback is recorded in a cloud-based database (MongoDB Atlas), which provides data for future model retraining. In the Data Preprocessing Pipeline, text input of symptoms is transformed into numerical feature vectors using TF-IDF vectorisation. This transformation is critical for effective model training and inference.

Dataset

The dataset is sourced from Kaggle, an online platform that hosts data science competitions and features an extensive repository of publicly available datasets. These datasets serve as a critical resource for researchers and practitioners, enabling them to train, test, and benchmark machine learning models.[10] The dataset for this research contains 5000 rows with a match of diseases and symptoms. Overall, the dataset contains 41 unique diseases. There are up to 17 symptom features with respect to diseases.

Preprocessing

The following steps were performed for data preprocessing:

- **Standardisation:** This step allows ensuring the String data type, stripping extra spaces, and lowercasing the values. The blank spaces between the symptoms were replaced by the underscore. For instance, symptoms like "Chest Pain" and "High fever" contain extra spaces and have inconsistent formatting. These were converted into "chest_pain" and "high_fever".
- **Handling Missing Values:** Drop the rows that contain missing values more than the threshold (90%). After dropping the rows, the dataset still has missing values. To resolve this, the missing values were replaced with the symptom -- "no_symptom".
- **Correcting Form:** Initially, multiple columns contain individual symptoms with respect to a disease. We merged all the columns of symptoms to form a list with respect to a disease.
- **List Transformation:** After converting symptoms to a list, we removed the symptoms that were named as "no_symptom".

At the end, the dataset only has two columns: the first contains diseases, and the second contains a list of symptoms.

Label Encoding and TF-IDF Vectorisation (Term Frequency-Inverse Document Frequency)

The disease columns are converted into numeric values using label encoding, which converts categorical variables into numerical formats by assigning unique numeric values to them [11]. The column with the symptoms list is converted into a numerical format using TF-IDF vectorisation, as shown in Eq. (3) [12]. 1. Term Frequency (TF): Measures how frequently a term appears in a document.

$$tf(t,d) = \frac{f_d(t)}{\max\limits_{\omega \in d} f_d(v} \qquad \text{Eq.(1)}$$

Inverse Document Frequency (IDF): Measure how important a rare term is across all documents.

$$idf(t,D) = \ln\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \qquad \text{Eq. (2)}$$

$$tfidf(t,d,D) = tf(t,d)\ .\ idf(t,D) \qquad \text{Eq. (3)}$$

Where, $f_d(t)$ = frequency of term t in document d

D = corpus of documents

Neural Network

The proposed system employs a deep learning-based multiclass classification model as shown in Figure 1, designed to predict diseases based on user-provided symptom descriptions. The implementation integrates data preprocessing, feature extraction, and a neural network model built using TensorFlow and Keras.

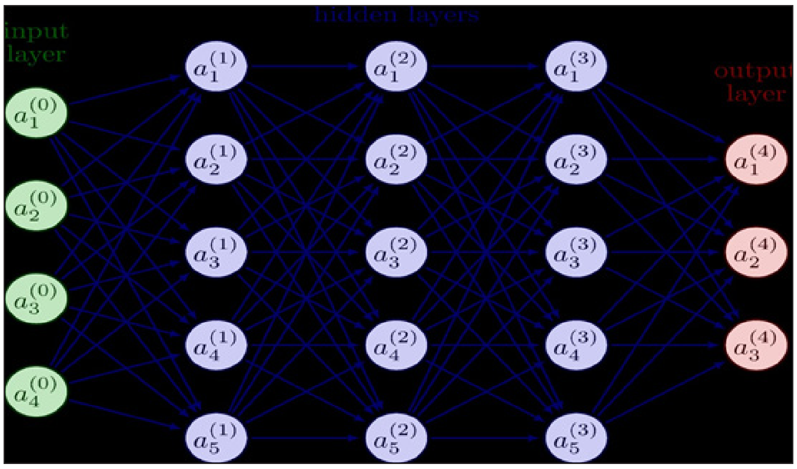


Figure 1.Architecture of Neural Network as indicated by[13]

Since the symptoms are in text format, they must be converted into numerical representations before being fed to the neural network. This is achieved using TF-IDF (Term Frequency–Inverse Document Frequency) vectorisation, which converts the text into weighted numerical feature vectors that capture the importance of each term within the dataset. Only the top 500 most informative features are retained to enhance computational efficiency and reduce model complexity. The transformed TF-IDF matrices are then converted into DataFrames for compatibility with TensorFlow.

And also, the target variable (disease) is categorical, and it is encoded into numerical form using LabelEncoder. This step allows the model to interpret disease categories as integer classes.

A compact feed-forward neural network (multilayer perceptron) is used for classification. The model has been optimised for faster training and reduced overfitting, which has an input layer that receives the 500-dimensional TF-IDF feature vectors, hidden layers that have two fully connected layers with ReLU activation to introduce non-linearity [14], and the output layer that contains a SoftMax layer with 41 neurones (equal to the number of distinct diseases) to produce class probabilities. The model is compiled using Adam Optimiser for efficient gradient-based optimisation. The loss function is computed using the SparseCategoricalCrossentropy as shown in Eq. (4), suitable for integer-encoded multiclass targets.[15]

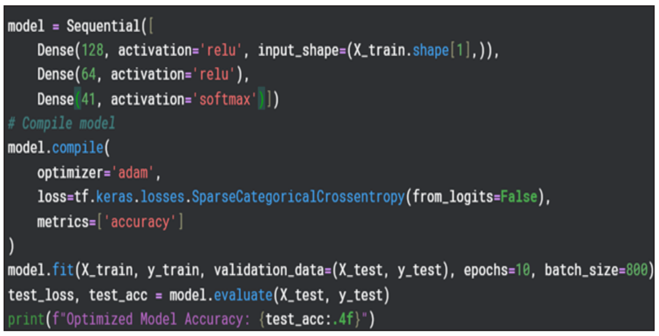
If there are c classes, the predicted probabilities are $y^{\wedge} = [p_0, p_1, \ldots .. p_{c-1}]$, then:

$$Loss = -\log(p_n) \qquad \text{Eq. (4)}$$

Where, p_n= Predicted probability of the correct class.

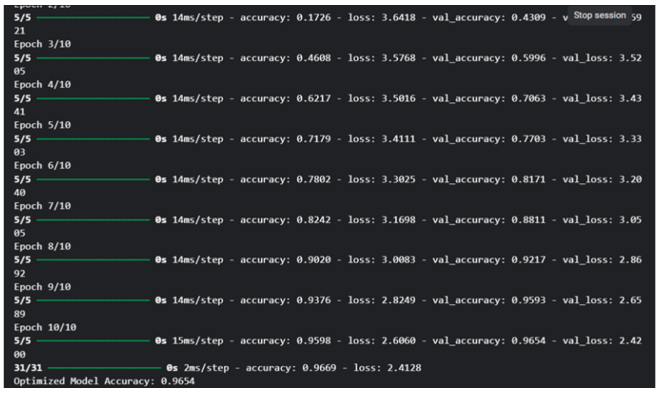
Model Evaluation

This model is evaluated on the unseen test set to measure its generalisation ability using the accuracy score, which measures the ratio of correct predictions to total predictions made. The obtained accuracy of the score indicates the model's ability to correctly classify diseases based on symptom inputs. Figures 2 and 3 show the code snippet of the training and testing of the disease diagnostic system:



```python
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(41, activation='softmax')])
# Compile model
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=800)
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Optimized Model Accuracy: {test_acc:.4f}")
```

Figure 2.This shows the making of neural network using the TensorFlow/Keras library of python[16]

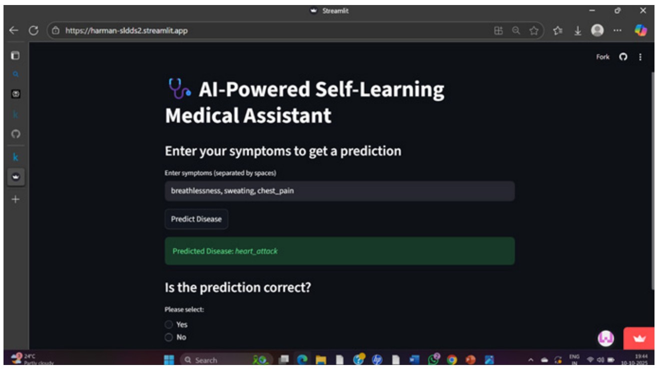


```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.1726 - loss: 3.6418 - val_accuracy: 0.4309 - v Stop session 59
21
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4608 - loss: 3.5768 - val_accuracy: 0.5996 - val_loss: 3.52
05
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6217 - loss: 3.5016 - val_accuracy: 0.7063 - val_loss: 3.43
41
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7179 - loss: 3.4111 - val_accuracy: 0.7703 - val_loss: 3.33
03
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7802 - loss: 3.3025 - val_accuracy: 0.8171 - val_loss: 3.20
40
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8242 - loss: 3.1698 - val_accuracy: 0.8811 - val_loss: 3.05
05
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9020 - loss: 3.0083 - val_accuracy: 0.9217 - val_loss: 2.86
92
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9376 - loss: 2.8249 - val_accuracy: 0.9593 - val_loss: 2.65
89
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9598 - loss: 2.6060 - val_accuracy: 0.9654 - val_loss: 2.42
00
31/31 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9669 - loss: 2.4128
Optimized Model Accuracy: 0.9654
```

Figure 3.This shows the training of neural network on 10 epochs, and final accuracy score of the model on the test dataset

Backend and Frontend

Availed "Streamlit" services to provide a user-friendly interface for users. Streamlit provides a textbox for the user to input symptoms, send an HTTP POST request to the /predict endpoint of the backend API, receive a response from the backend in JSON, and display the result (as shown in figure 4). In addition, it comes up with a textbox for user feedback if the output seems to be incorrect (as shown in figure 5). "Render" was used to deploy the model's backend, which performs core functionality.[17] We have used the Flask library to design the backend. The backend receives requests from the frontend, preprocesses symptoms, and invokes the model for prediction. Then it sends a prediction to the frontend and stores the user query and feedback in the MongoDB database (as shown in figure 7). Lastly, MongoDB Atlas stores user queries and user feedback for fine-tuning and rare disease learning.


AI-Powered Self-Learning Medical Assistant

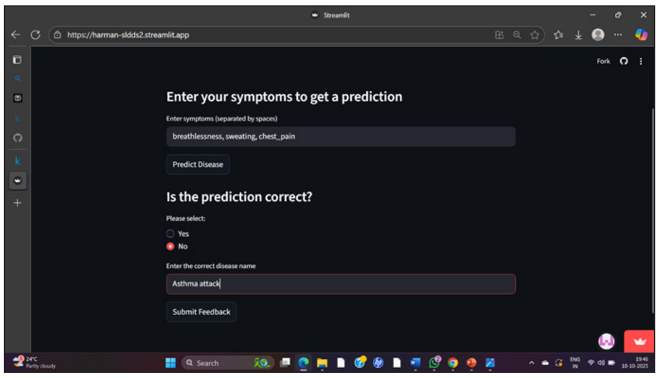
Enter your symptoms to get a prediction

Enter symptoms (separated by spaces): breathlessness, sweating, chest_pain

Predicted Disease: heart_attack

Is the prediction correct?

Figure 4.This shows the web interface where user enter the symptoms and gets the predicted disease as output


Enter your symptoms to get a prediction

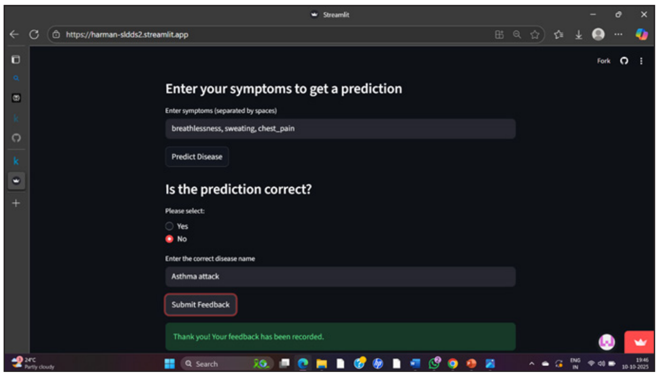
Enter symptoms (separated by spaces): breathlessness, sweating, chest_pain

Is the prediction correct?

Please select: Yes [✓ No]

Enter the correct disease name: Asthma attack

Figure 5.If the user has a doubt that the disease predicted by the model is not correct, the user can give the feedback as shown in this figure


Enter your symptoms to get a prediction

Enter symptoms (separated by spaces): breathlessness, sweating, chest_pain

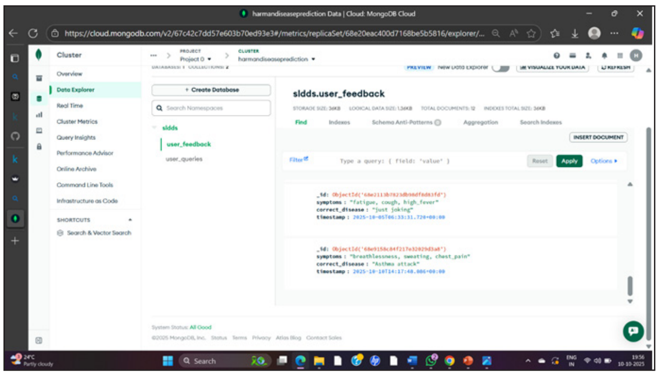
Is the prediction correct?

Please select: Yes [✓ No]

Enter the correct disease name: Asthma attack

Thank you! Your feedback has been recorded.

Figure 6.This illustrate that the system has successfully submitted the feedback into the MongoDB database


sldds.user_feedback

```
_id: ObjectId('68e2113b7823db98dfad83fd')
symptoms : "fatigue, cough, high_fever"
correct_disease : "just joking"
timestamp : 2025-10-05T06:33:31.720+00:00

_id: ObjectId('68e9158c84f217e32029d3a8')
symptoms : "breathlessness, sweating, chest_pain"
correct_disease : "Asthma attack"
timestamp : 2025-10-10T14:17:48.086+00:00
```

Figure 7.User feedback and User queries get stored in MongoDB Atlas in a JSON format. After validation of the dataset that is stored in the cloud database, the original dataset can be augmented with this dataset. Then the model can be retrained on entire dataset which would help the model to improve generalization and accuracy, and also learn new and rare symptoms-disease mappings

Future Work

A promising direction for future work is the integration of reinforcement learning (RL) to enable automated retraining of the model based on user feedback. The current feedback mechanism captures corrections that can be reviewed manually; however, an RL framework would allow the model to adjust its weights dynamically, thereby continuously learning from its mistakes and improving accuracy. Future research could investigate optimal strategies for integrating reinforcement signals with supervised learning, ensuring stable and consistent performance improvements. Advanced Natural Language Processing (NLP)[19]: To address the limitations in handling ambiguous or incomplete inputs, future iterations of the model should incorporate more advanced NLP techniques. Enhancing the preprocessing pipeline with state-of-the-art NLP models—such as transformers or BERT-based encoders[20]—could improve the extraction of relevant features from free-form text. This enhancement would better handle nuances in symptom descriptions, leading to more accurate and robust predictions. The current system is designed solely for disease prediction, but its potential can be expanded significantly by integrating diagnostic recommendations with precautionary advice. Future work should focus on developing a module that not only identifies likely diseases but also provides practical recommendations on preventative measures and lifestyle adjustments. This might include guidelines such as when to seek immediate medical attention, suggestions for home care, or alerts about potential complications based on the disease diagnosed. Continued expansion and diversification of the dataset will be key to addressing the challenge of performance on unseen data. Incorporating more varied datasets, including data from different demographics and geographies, could help the model to become better at generalisation. Additionally, leveraging data augmentation techniques and regularisation methods in the model training process may further reduce overfitting and improve robustness. While the current interface is designed for simplicity and ease of use, further refinements in the UI/UX can enhance accessibility and user engagement. Future developments could involve creating a more dynamic interface that supports multilingual inputs, voice commands, and more interactive feedback mechanisms. Additionally, incorporating detailed visualisations of the prediction process and confidence levels can help build user trust and understanding.

Discussion and Conclusion

The development of the AI-powered self-learning medical assistant represents a significant stride in applying artificial intelligence to healthcare diagnostics. This section reflects on the journey of this research by summarising key findings and achievements and then discussing the inherent limitations while outlining opportunities for further research and development. High prediction accuracy and efficiency are two of the most impressive outcomes of this research. The high prediction accuracy achieved by the deep learning model is particularly impressive. The system is trained on a dataset containing 5,000 records of symptom-disease mappings covering 41 distinct diseases, and the model reached approximately 96.54% accuracy on the test set. This result has a clear meaning: that the neural network, even with a relatively compact architecture, can capture meaningful patterns and correlations between the symptoms and diseases. In addition to accuracy, efficiency was a major design goal. Initially, when the model was deployed using the full TensorFlow/Keras format, it encountered performance issues on Render due to the limited 512 MB RAM provided by the hosting environment. Converting the model into the TensorFlow Lite format proved transformative. With optimisations such as float16 quantisation, the model not only fit within the memory constraints but also showed significant improvements in inference speed. The research benefits greatly from a modular software architecture. Each component, from the Streamlit-based frontend to the Flask backend deployed on Render, and from the data preprocessing pipeline to the deep learning inference engine, integrates seamlessly. This integration ensures that user input flows efficiently from the web interface, through the REST API, to the prediction model, and back to the user in real time. The real-time feedback mechanism is another notable achievement. Users can confirm or correct the model's prediction immediately after obtaining a result. This feedback is stored in MongoDB Atlas and lays the foundation for future automated retraining strategies. Although the current version of the system does not perform automatic model updates, the design anticipates future integration of reinforcement learning techniques to harness this user feedback dynamically. A crucial goal of this research was to build an accessible and cost-effective diagnostic tool. The use of Render for backend deployment has enabled the research to benefit from cloud scalability without incurring high costs. Render's environment, though resource-constrained (512 MB RAM), proved sufficient once the model was converted to TensorFlow Lite. Moreover, leveraging free-tier services such as MongoDB Atlas and the open-source frameworks (TensorFlow, Flask, and Streamlit) has resulted in a solution that remains both low-cost and scalable. This makes the research particularly attractive for adoption in under-resourced settings where traditional diagnostic infrastructure might be limited. Despite the impressive performance on the training and test sets, the model shows some limitations when facing unseen data. In scenarios where the input symptom text differs substantially from the training examples, the model's prediction accuracy drops. This clearly suggests that the model may require further tuning and a more diverse set of training samples to generalise effectively in real-world applications. Another problem is with handling ambiguous or incomplete inputs. The current system expects symptom inputs to be provided in a straightforward, whitespace-separated format. However, in a practical case, users may enter ambiguous, incomplete, or even conflicting symptom information. Such cases can confuse the model, leading to incorrect predictions. In future versions, enhancing the preprocessing pipeline and integrating advanced natural language processing (NLP) techniques will help the model to better interpret and standardise more complex or unstructured inputs. Another challenge the system faces is memory and computational constraints. The current 512 MB RAM constraint might limit the future enhancements or the integration of more complex models.

References

1. M. Uzair and N. Jamil, "Effects of hidden layers on the efficiency of neural networks," in 2020 IEEE 23rd International Multitopic Conference (INMIC), 2020: IEEE, pp. 1-6.
2. J. Gawlikowski et al., "A survey of uncertainty in deep neural networks," Artificial Intelligence Review, vol. 56, no. Suppl 1, pp. 1513-1589, 2023. https://doi.org/10.48550/arXiv.2107.03342
3. Chollet, F. (2017). Deep Learning with Python. Manning Publications.
4. J. Wen et al., "Convolutional neural networks for classification of Alzheimer's disease: Overview and reproducible evaluation," Medical Image Analysis, vol. 63, p. 101694, 2020. https://doi.org/10.1016/j.media.2020.101694
5. Grinberg, M. (2018). Flask web development: Developing web applications with Python (2nd ed.). O'Reilly Media.
6. Streamlit Inc. (2022). Streamlit documentation. Streamlit. https://docs.streamlit.io (accessed on 10 October, 2025)
7. A. Ashfaq, A. Sant'Anna, M. Lingman, and S. Nowaczyk, "Readmission prediction using deep learning on electronic health records," Journal of Biomedical Informatics, vol. 97, p. 103256, 2019. https://doi.org/10.1016/j.jbi.2019.103256
8. J. Dong et al., "Machine learning model for early prediction of acute kidney injury (AKI) in pediatric critical care," Critical Care, vol. 25, no. 1, p. 288, 2021. https://doi.org/10.1186/s13054-021-03724-0
9. L. Ali, A. Rahman, A. Khan, M. Zhou, A. Javeed, and J. A. Khan, "An automated diagnostic system for heart disease prediction based on ${\chi^{2}} $ statistical model and optimally configured deep neural network," IEEE Access, vol. 7, pp. 34938-34945, 2019.
10. Kaggle. (2023). Kaggle datasets. Kaggle. https://www.kaggle.com/datasets (accessed on 10 October, 2025)
11. scikit-learn developers. (n.d.). sklearn.preprocessing.LabelEncoder [Software documentation]. scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (accessed on 10 October, 2025)
12. G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Information Processing & Management, vol. 24, no. 5, pp. 513–523, 1988, doi: 10.1016/0306-4573(88)90021-0.
13. Lee, F. (n.d.). What is a neural network? IBM. https://www.ibm.com/think/topics/neural-networks (accessed on 10 October, 2025)
14. TensorFlow. (n.d.). tf.keras.activations.relu [Software documentation]. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu (accessed on 10 October, 2025)
15. TensorFlow. (n.d.). tf.keras.losses.SparseCategoricalCrossentropy [Software documentation]. TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy
16. TensorFlow. (n.d.). TensorFlow Core [Software documentation]. TensorFlow. https://www.tensorflow.org/ (accessed on 10 October, 2025)
17. Render. (n.d.). Web services [Documentation]. Render. https://render.com/docs/web-services (accessed on 10 October, 2025)
18. MongoDB, Inc. (n.d.). What is MongoDB Atlas? MongoDB Docs. https://www.mongodb.com/docs/atlas/ (accessed on 10 October, 2025)
19. Jurafsky, D., & Martin, J. H. (2023). Speech and language processing (3rd ed. draft). Stanford University. https://web.stanford.edu/~jurafsky/slp3/
20. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT) (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.48550/arXiv.1810.04805