

DEPLOYMENT OF MEDIBOT IN MEDICAL FIELD

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

P. RESHMANTH (Reg. No. 38110389)

P. SUSHANTH CHOWDARY (Reg. No. 38110387)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAR NAGAR, RAJIV GANDHI SALAI, CHENNAI – 600 119

MARCH – 2022



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119
www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **P. RESHMANTH (Reg. No: 38110389)** and **P. SUSHANTH CHOWDARY (Reg. No: 38110387)** who carried out the project entitled "**DEPLOYMENT OF MEDIBOT IN MEDICAL FIELD**" under our supervision from **August 2021** to **March 2022**.

Internal Guide

Ms. R. Yogitha

Head of the Department

Dr. L. LAKSHMANAN M.E., Ph.D.,

Dr. S. VIGNESHWARI, M.E., Ph.D.,

Submitted for Viva-voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **P. RESHMANTH (Reg No:38110389)** and **P. SUSHANTH CHOWDARY (Reg No: 38110397)** hereby declare that the Project Report entitled “**DEPLOYMENT OF MEDIBOT IN MEDICAL FIELD**” done by me under the guidance of **Ms. R. YOGITHA**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of **Bachelor of Engineering (B.E)** degree in **Computer Science and Engineering (CSE)**

DATE: 04-03-2022

P. RESHMANTH

P. SUSHANTH CHOWDARY

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D., Dean**, School of Computing and **Dr. S. VIGNESWARI, M.E., Ph.D., and Dr. L. LAKSHMANAN, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. R. YOGITHA** for her valuable guidance, suggestions and constant encouragement that paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of **Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

During this pandemic situation, most the people's health care is in the need of medicine and doctors' suggestions to improve and protect their health. Also, have seen many such cases where many people have been infected by COVID. To reduce physical contact and help the people from the spread of diseases the proposed methodology is to implement the medibot in hospitals. A medical bot is a Chatbot that uses NLP (Natural Language Processing) by text format. The medibot is supported by AI and Deep Learning for Medical Diagnostics. The goal of the project is to create a medibot that overcomes the proposed methodology. Many people could not meet the doctors for simple problems such as cold and fever. To reduce these cases will implement the medibot. This medibot can communicate with the patients and understand the symptoms, it will also give them medicines.

TABLE OF CONTENT		
CHAPTER NO.	TITLE	PAGE.NO
	ABSTRACT	v
	LIST OF FIGURES	ix
1	INTRODUCTION	1
1.1	OUTLINE OF THE PROJECT	1
1.2	SOFTWARE REQUIREMENTS	2
1.2.1	PROGRAMMING LANGUAGE	2
1.2.2	PYTHON NLBRARIES& PACKAGES	2
1.2.3	ALGORITHM ARCHITECTURE	2
1.3	HARDWARE REQUIREMENTS	2
2	LITERATURE REVIEW	3
2.1	MACHINE LEARNING	4
2.1.1	SOME MACHINE LEARNING METHODS	4
	2.1.1.1 SUPERVISED MACHINE LEARNING ALGORITHM	4
	2.1.1.2 UN-SUPERVISED MACHINE LEARNING ALGORITHM	
	2.1.1.3 SUPERVISED MACHINE LEARNING ALGORITHM	5
	2.1.1.4 REINFORCEMENT MACHINE LEARNING ALGORITHM	5
2.2	DEEP LEARNING	5
	2.2.1 HOW DOES DEEP LEARNING WORKS	6
2.3	PYTHON	6
2.4	PACKAGE INSTALLER	7
2.5	TENSOR FLOW	7
2.6	KERAS	8
2.7	NLP	8
2.8	NLTK	9
2.9	PICKLE	9
2.10	NUMPY	10
3	METHODOLOGY	11
3.1	DATASETS	11
3.1.1	JSON	11

	3.1.2	INTENTS	12
3.2		REQUIRMENTS	12
	3.2.1	PROJECT SCOPE	12
	3.2.2	EXISTING SYSTEM	13
	3.2.3	PROPOSED SYSTEM	13
	3.2.4	FUNCTIONAL REQUIREMENTS	14
	3.2.5	NON-FUNCTIONAL REQUIREMENTS	14
3.3		DATA PROCESSING	14
3.4		MODELS	15
3.4.1		SEQUENTIAL MODEL	15
3.4.2		DENSER FLOW	16
3.4.3		RNN	17
3.4.4		LSTM	17
3.5		EXPERIMENTAL SETUP	20
	3.5.1	TRAIN DATASET	20
	3.5.2	VALIDATION DATASET	20
	3.5.3	TEST DATASET	21
3.6		PARAMETERS GIVEN TO MODEL FOR TRAINING	21
	3.6.1	BATCH SIZE	21
	3.6.2	EPOCHS	21
	3.6.3	VERBOSE	21
	3.6.4	VALIDATION SPLIT	22
4		RESULTS AND DISCUSSION	26
4.1		EVALUATION MATRIX AND RESULT	26
4.2		TRAINING AND TESTING DATA	27
4.3		BUILDING THE MODEL	28
	4.3.1	PREDICT THE RESPONSE	29
5		CONCLUSION AND FUTURE WORK	33

5.1	CONCLUSION	33
5.2	FUTURE ENHANCEMENT	33
	REFERENCE	34
	APPENDIX	36
A.	SOURCE CODE	37
B.	SCREENSHOTS	48
C.	PUBLICATION WITH PLAGIARISM REPORT	50

LIST OF FIGURES

FIGURENO.	FIGURENAME	PAGENO.
1	IMAGE OF CONVERSATIONAL LSTM	19
2.	PROCESS DIAGRAM BETWEEN PATIENTS AND MEDIBOT	22
3.	DESIGN FLOW DIAGRAM	23
4.	SYSTEM ARCHITECTURE	24
5.	PROCESSING STEPS FOR CHATBOT	25
6.	ACCURACY	27

CHAPTER 1

INTRODUCTION:

1.1 Outlet of The Project:

This is an automated chat robot design to answer users frequently asked questions, earlier natural language processing techniques were using to design this robot but its accuracy of giving correct answer was less and now due to Deep Learning algorithms accuracy of giving correct answer increase, so here using python deep learning project we are building CHATBOT application to answer users questions.

To implement this technique first we train deep learning models with the train data (all possible question's answers) and whenever users give any question then application will apply this test question on train model to predict exact answer for given question.

Earlier companies were hiring humans to answer user's queries but by using this application we can answer user's question without using any manpower.

Chabot can be described as software that can chat with people using artificial intelligence. Chabot's are generally used to respond quickly to users. Chabot's, a common name for automated conversational interfaces, present a new way for individuals to interact with computer systems. Traditionally, to get a question answered by a software program involves using a search engine or filling out a form. A Chabot allows a user to simply ask questions in the same manner that they would address a human. There are many well-known voice-based catboats currently available in the market: Google Assistant, Alexa and Siri. Chabot's are currently being adopted at a high rate on computer chat platforms.

To implement this project, we are using python deep learning neural networks and NLTK (natural Language Processing API) to process train and test text data.

1.2 Software Requirements

- Good Internet speed
- Javascript enabled browser

1.2.1 Programming Language

Python

1.2.2 Python Libraries & Packages

- Tensorflow
- Keras
- NLTK
- Pickle

1.2.3 Algorithm – Architecture

RNN - LSTM(Deep Neural Networks)

1.3 Hardware Requirements

RAM- minimum 4gb/Above

Processor- minimum intel i5 core/Above

Windows- 7/Above

CHAPTER 2

LITERATURE SURVEY:

In today's world, health is the major key in the development of each sector. Health also needs modern technology and their implementations in order to boost the development in this area. As know that it is the most important field in each country, need to provide sufficient technological development. A lot of researches have been done in this field in order to modernize the methods of health status. Old methods of diagnosing are still a major hindrance in the advancement of medical facilities. The best way to overcome this problem is using a medical chatbot with self-diagnosis using Artificial Intelligence.

The proposed method can use deep learning algorithms to increase the capabilities of computers and by enhancing it can understand what humans can do, which includes speech and text recognition. In this will use text-text in mediots to contact the patients and teach the chatbots to process natural language text It discusses about processing natural language using Recurrent Neural Network (RNN). The sequence to sequence long short-term memory cell neural network (LSTM) is used to train the model. In addition, it also talks about the challenges of implementing a Recurrent Neural Network based chatbot. Disease diagnosis system using several machine learning algorithms is proposed. A detailed comparison of four Machine Learning algorithms to predict disease based on symptoms provided, is also presented in this paper.

The purpose was to find the best Machine Learning algorithm to diagnose diseases early to help doctors and patients, as early prediction of disease can improve treatment efficiency. The chatbot will start questioning the patients regarding their symptoms and problems they are suffering and suggest an effective method to reduce the disease or give the medication according to the disease user suffering. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural

network. Recurrent Neural Network (RNN) are a type of Neural Network where the output from the previous step is fed as input to the current step. ... It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output [8]. In this can build algorithms to make the computer automatically analyze the given data and also make it understand human language. In this, mainly use text recognition using deep learning. Deep learning algorithms are used to enhance the capability of a chatbot. The main focus of the review is to implement medibot using both deep learning and neural networks.

2.1 Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

2.1.1 Some machine learning methods

Machine learning algorithms are often categorized as supervised or unsupervised.

2.1.1.1 Supervised machine learning algorithm

Can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system can provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

2.1.1.2 Unsupervised machine learning algorithm

Are used when the information used to train is neither classified nor labeled.

Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system does not figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

2.1.1.3 Semi-supervised machine learning algorithm

Fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method can considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it learn from it. Otherwise, acquiring unlabeled data generally does not require additional resources.

2.1.1.4 Reinforcement machine learning algorithm

Is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal. Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

2.2 Deep Learning

Deep learning is an AI function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of ML in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabelled. Also known as deep neural learning or deep neural network.

- Deep learning is an AI function that mimics the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions.
- Deep learning AI can learn without human supervision, drawing from data that is both unstructured and unlabelled.
- Deep learning, a form of machine learning, can be used to help detect fraud or money laundering, among other functions.

2.2.1 How Does Deep Learning Works

Deep learning has evolved together with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as big data, is drawn from sources like social media, internet search engines, e-commerce platforms, and online cinemas, among others. This enormous amount of data is readily accessible and can be shared through applications like cloud computing.

However, the data, which normally is unstructured, is so vast that it could take decades for humans to comprehend it and extract relevant information. Companies realize the incredible potential that can result from unravelling this wealth of information and are increasingly adapting to AI systems for automated support.

2.3 Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

2.4 Package Installer

PIP - pip is a command line application written in Python. PIP is a package management system used to install and manage software packages written in Python. It stands for "preferred installer program" or "Pip Installs Packages." PIP for Python is a utility to manage PyPI package installations from the command line.

Syntax: pip install tensorflow, keras, pickle, nltk

2.5 TensorFlow

It is an open-source artificial intelligence library, using data flow graphs to build models. It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation. TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the

model definition. TensorFlow has always provided a direct path to production. Whether it's on servers, edge devices, or the web, TensorFlow lets you train and deploy your model easily, no matter what language or platform you use. TensorFlow gives you the flexibility and control with features like the Keras Functional API and Model Subclassing API for creation of complex topologies. For easy prototyping and fast debugging, use eager execution.

2.6 Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras is the most used deep learning framework among top-5 winning teams on [Kaggle](#). Because Keras makes it easier to run new experiments, it empowers you to try more ideas than your competition, faster.

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was “designed for human beings, not machines,” and “follows best practices for reducing cognitive load. Keras is used by CERN, NASA, NIH, and many more scientific organizations around the world (and yes, Keras is used at the LHC). Keras has the low-level flexibility to implement arbitrary research ideas while offering optional high-level convenience features to speed up experimentation cycles.

2.7 NLP

Natural language processing strives to build machines that understand and respond to text or voice data—and respond with text or speech of their own—in much the same way humans do.

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There is a good chance you have interacted with NLP

in the form of voice-operated GPS systems, digital assistants, speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

NLP Use Cases - Natural language processing is the driving force behind machine intelligence in many modern real-world applications. Here are a few examples:

1. **Virtual assistants and chatbots**
2. **Spam detection**
3. **Text summarization**
4. **Social media sentiment analysis**
5. **Machine translation**

2.8 NLTK

NLTK is a leading platform for building Python programs to work with human language data. NLTK includes libraries for many of the NLP tasks, plus libraries for subtasks, such as sentence parsing, word segmentation, stemming and lemmatization (methods of trimming words down to their roots), and tokenization (for breaking phrases, sentences, paragraphs and passages into tokens that help the computer better understand the text). It also includes libraries for implementing capabilities such as semantic reasoning, the ability to reach logical conclusions based on facts extracted from text.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

The Python programming language provides a wide range of tools and libraries for attacking specific NLP tasks. Many of these are found in the Natural Language Toolkit, or NLTK, an open-source collection of libraries, programs, and education resources for building NLP programs.

2.9 Pickle

The Pickle module implements binary protocols for serializing and de-serializing a Python object structure. “*Pickling*” is the process whereby a Python object hierarchy is converted into a byte stream, and “*unpickling*” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”.

The data format used by pickle is Python-specific. This has the advantage that there are no restrictions imposed by external standards such as JSON or XDR (which can’t represent pointer sharing); however, it means that non-Python programs may not be able to reconstruct pickled Python objects.

By default, the pickle data format uses a relatively compact binary representation. If you need optimal size characteristics, you can efficiently compress pickled data. The module `pickletools` contains tools for analysing data stream generated by Pickle.

2.10 Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

NumPy fully supports an object-oriented approach, starting, once again, with *ndarray*. For example, *ndarray* is a class, possessing numerous methods and

attributes. Many of its methods are mirrored by functions in the outer-most NumPy namespace, allowing the programmer to code in whichever paradigm they prefer.

CHAPTER 3

METHODOLOGY:

3.1 DataSets:

A data set is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question. The data set lists values for each of the variables, such as the height and weight of an object, for each member of the data set. Each value is known as a datum. Data sets can also consist of a collection of documents or files

3.1.1 JSON:

JSON is a text format for storing and transporting data. JSON is self-describing and easy to understand.

- JSON stands for **JavaScript Object Notation**
- JSON is a lightweight data-interchange format.
- JSON is plain text written in JavaScript object notation.
- JSON is used to send data between computers.
- JSON is language independent.

Syntax – {“attribute1”: “Value1” , “attribute2” : “Value2” ,}

Example - {“firstName”: “Jason”,

“lastName”: “Jones”,

“address”: {

“city”: “San Francisco, CA”,

“zip”: 94121,

```
"address": "444 Columbus Ave"

},

}
```

Why JSON?

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

3.1.2 Intents:

Local Data Storage (No External Data Set required)

File Format: JSON

The dataset we will be using is 'intents.json'. This is a JSON file that contains the patterns we need to find and the responses we want to return to the user.

3.2 Requirements:

This chapter provides the details of the project's need-based survey, system requirements, Hardware Requirements, Software Requirements, and System Requirements.

3.2.1 Project Scope:

- This is an automated chat robot design to answer users frequently asked questions, earlier natural language processing techniques were using to design this robot but its accuracy of giving correct answer was less and now due to Deep Learning algorithms accuracy of giving correct answer increase, so here using

python deep learning project we are building CHATBOT application to answer users questions.

- To implement this technique first we train deep learning models with the train data (all possible questions' answers) and whenever users give any question then application will apply this test question on train model to predict exact answer for given question.

3.2.2 Existing System:

Before exploring new technology, one should examine prior work and learn from past ideas, both succeed and failed attempts. This section presents a selection of events from the last century, which introduced the ideas that formed the present definition of a Chabot. It is not an attempt to give an all-encompassing overview about the history of computing, instead the aim is to explain where the concept of chat bots and the interest of creating them originated.

Draw backs of Existing System:

- No proper support
- Communication overhead
- High risk
- More Manpower

3.2.3 Propsed System:

Chatbot can be described as software that can chat with people using artificial intelligence. Chatbots are generally used to respond quickly to users. Chatbots, a common name for automated conversational interfaces, present a new way for individuals to interact with computer systems. Traditionally, to get a question answered by a software program involves using a search engine or filling out a form. A chatbot allows a user to simply ask questions in the same manner that they would address a human. There are many well-known voice-based chatbots currently available in the market: Google Assistant, Alexa and Siri. Chatbots are currently being adopted at a high rate on computer chat platforms. A chatbot can be used anywhere a human is interacting with a computer system. These are the areas where the fastest adoption is occurring.

Advantages: -

- Offers 24/7 Service
- Enhances User Experience
- Cost Efficient

3.2.4 Functional requirements:

In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behaviour, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional software requirements help you to capture the intended behaviour of the system. This behaviour may be expressed as functions, services or tasks or which system is required to perform.

3.2.5 non-Functional requirements:

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load? A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

3.3 Data Processing:

When working with text data, we need to perform various pre-processing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to pre-process the data.

Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.

Here we iterate through the patterns and tokenize the sentence using `nltk.word_tokenize()` function and append each word in the words list. We also create a list of classes for our tags.

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        #add documents in the corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Now we will lemmatize each word and remove duplicate words from the list. Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.

```
# lemmatize, lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

# sort classes
classes = sorted(list(set(classes)))

# documents = combination between patterns and intents
print (len(documents), "documents")

# classes = intents
print (len(classes), "classes", classes)

# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)

pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

3.4 Models:

3.4.1 Sequential Model:

The core idea of Sequential model is simply arranging the Keras layers in a sequential order and so, it is called Sequential model. Most of the ANN also has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

A Sequential model is not appropriate when:

- ❖ Model has multiple inputs or multiple outputs.
- ❖ Any of the layers has multiple inputs or multiple outputs.
- ❖ If need to do layer sharing.
- ❖ If there is a need of non-linear topology (e.g., a residual connection, a multi-branch model).

In this project there are 3 layers, first layer 128 neurons, second layer 64 neurons and 3rd output layer contain number of neurons equal to number of intents to predict output intent with SoftMax.

We have used the following layers –

3.4.2 dense layer:

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

$$\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$$

where,

- **input** represent the input data.
- **kernel** represent the weight data.
- **dot** represent NumPy dot product of all input and its corresponding weights
- **bias** represent a biased value used in machine learning to optimize the model.
- **activation** represent the activation function.

Dropout - Dropout is a technique used to prevent a model from overfitting (**Overfitting** occurs when a model tries to predict a trend in data that is too noisy. This is caused due to an overly complex model with too many parameters. A model that is overfitted is inaccurate because the trend does not reflect the reality present in the data.). Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.

Activation Function - An activation function is a very important feature of an artificial neural network, they basically decide whether the neuron should be activated or not. In artificial neural networks, the activation function defines the output of that node given an input or set of inputs.

In the first layer we have used Relu function (ReLU is a non-linear activation function that is used in multi-layer neural networks or deep neural networks), and in the third layer we have used Softmax function (function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector.).

3.4.3 RNN:

Recurrent neural networks are the state-of-the-art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. Recurrent Neural Networks (RNN) are a class of Artificial Neural Networks that can process a sequence of inputs in deep learning and retain its state while processing the next sequence of inputs. Recurrent Neural Networks (RNN) are a type of Neural Network where the output from the previous step is fed as input to the current step.

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

Recurrent Neural Network (RNN) are a type of [Neural Network](#) where the output from previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.

Advantages:

- Possibility of processing input of any length.
- Model size not increasing with size of input.
- Computation considers historical information.

- Weights are shared across time.

3.4.4 LSTM:

LSTM stands for Long-Short Term Memory. LSTM is an advanced version of recurrent neural network (RNN) architecture that was designed to model chronological sequences and their long-range dependencies more precisely than conventional RNNs.

The major highlights include the interior design of a basic LSTM cell, the variations brought into the LSTM architecture and few applications of LSTMs that are highly in demand.

RNN is a network that works on the present input by taking into consideration the previous output (feedback) and storing in its memory for a short period of time (short-term memory). Nevertheless, there are drawbacks to RNNs.

First, it fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such “long-term dependencies”.

Second, there is no finer control over which part of the context needs to be carried forward and how much of the past needs to be ‘forgotten’.

Other issues with RNNs are exploding and vanishing gradients which occur during the training process of a network through backtracking. Thus, LSTM was brought into the picture. It has been designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered.

During the training process of a network, the main goal is to minimize loss (in terms of error or cost). We calculate the gradient i.e., loss with respect to a particular set of weights. Adjust the weights accordingly and repeat this process

until the optimal set of weights for which the loss is minimum. This is the **concept of backtracking**.

Sometimes, it so happens that the gradient is almost negligible. It must be noted that the gradient of a layer depends on certain components in the successive layers. If some of these components are small (i.e., less than 1), the resultant gradient will be even smaller. This is known as the **scaling effect**.

When this gradient is multiplied with the learning rate, which is itself a small value ranging between 0.1-0.001, it results in a smaller value. As a consequence, the alteration in weights is quite small, producing almost the same output as before. Similarly, if the gradients are quite large in value due to the large values of components, the weights get updated to a value beyond the optimal value. This is known as the **problem of exploding gradients**.

To avoid this scaling effect, the neural network unit was re-built in such a way that the scaling factor was fixed to one. The cell was then enriched by several gating units and termed as LSTM.

LSTMs comprises of three logistic sigmoid gates and one tanh layer. Gates have been introduced in order to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part is to be discarded. The output is usually in the range of 0-1 where '0' means 'reject all' and '1' means 'include all'.

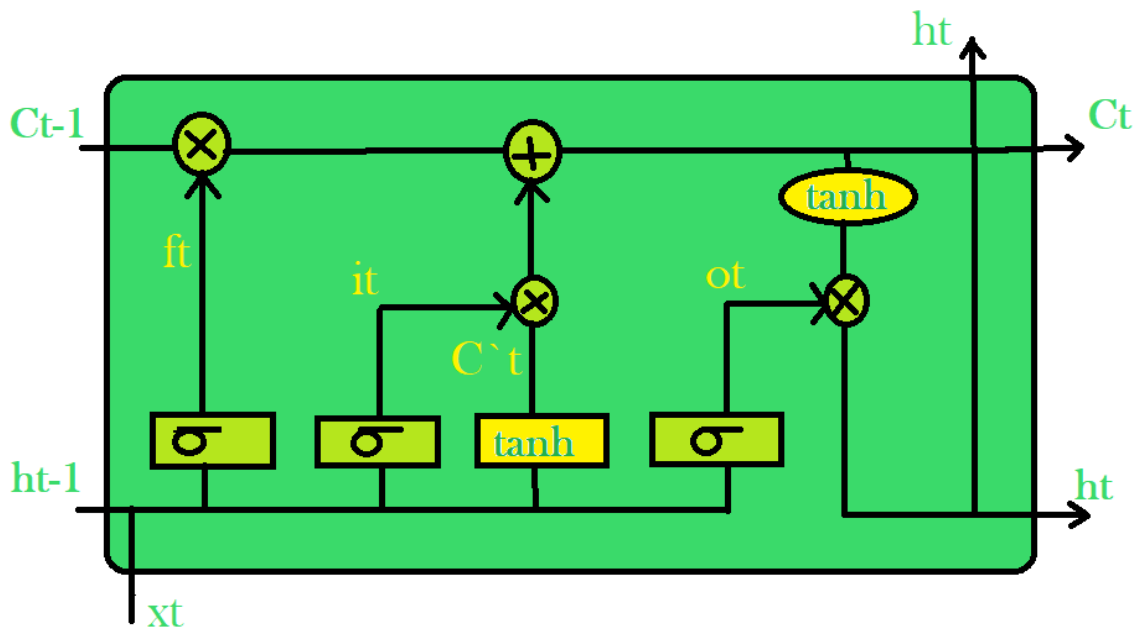


Fig-1

Image of Conventional LSTM

- ✓ Each LSTM cell has three inputs $ht-1$, $Ct-1$ and xt and two outputs ht and Ct . For a given time t , ht is the hidden state, Ct is the cell state or memory, xt is the current data point or input.
- ✓ The first sigmoid layer has two inputs $ht-1$ and xt where $ht-1$ is the hidden state of the previous cell.
- ✓ First sigmoid layer is also known as the forget gate as its output selects the amount of information of the previous cell to be included. The output is a number in $[0,1]$ which is multiplied (pointwise) with the previous cell state $Ct-1$.
- ✓ Second sigmoid layer is the input gate that decides what new information is to be added to the cell. It takes two inputs $ht-1$ and xt .
- ✓ The tanh layer creates a vector Ct' of the new candidate values. Together, these two layers determine the information to be stored in the cell state. Their point-wise multiplication ($it \times Ct'$) tells us the amount of information to be added to the cell state.
- ✓ The result is then added with the result of the forget gate multiplied with previous cell state ($ft \times Ct-1$) to produce the current cell state Ct .

- ✓ Next, the output of the cell is calculated using a sigmoid and a tanh layer. The sigmoid layer decides which part of the cell state will be present in the output whereas tanh layer shifts the output in the range of $[-1,1]$.
- ✓ The results of the two layers undergo point-wise multiplication to produce the output h_t of the cell.

3.5 Experimental Setup:

3.5.1 *Train Dataset:*

It is the sample of data used to fit the model. This step involves in extraction of features and train to fit a model and so on. The observations in the training set form the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed output variable and one or more observed input variables.

3.5.2 *Validation Dataset:*

The sample of data used to provide unbiased evaluation of a model to fit on the train data set. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

3.5.3 *Test dataset:*

The sample of data used to provide unbiased evaluation of a final model. It used to test if the model is making accurate predictions. The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

3.6 Parameters given to model for training:

3.6.1 Batch size:

Integer or None. Number of samples per gradient update. If unspecified, batch_size will default to 32. Do not specify the batch size if

the data is in the form of datasets, generators, or `Keras.utils.Sequence` instances.

3.6.2 Epochs:

Integer. Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided. Note that in conjunction with initial epoch, epochs is to be understood as "final epoch". The model is not trained for a number of iterations given by epochs, but merely until the epoch of index epochs is reached.

3.6.3 Verbose:

'auto', 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch. 'auto' defaults to 1 for most cases, but 2 when used with `ParameterServerStrategy`. Note that the progress bar is not particularly useful when logged to a file, so verbose=2 is recommended when not running interactively (e.g., in a production environment).

3.6.4 Validation Split:

Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.

20% of the train dataset is used as validation dataset for validating the model which is trained by 80% of the train dataset.

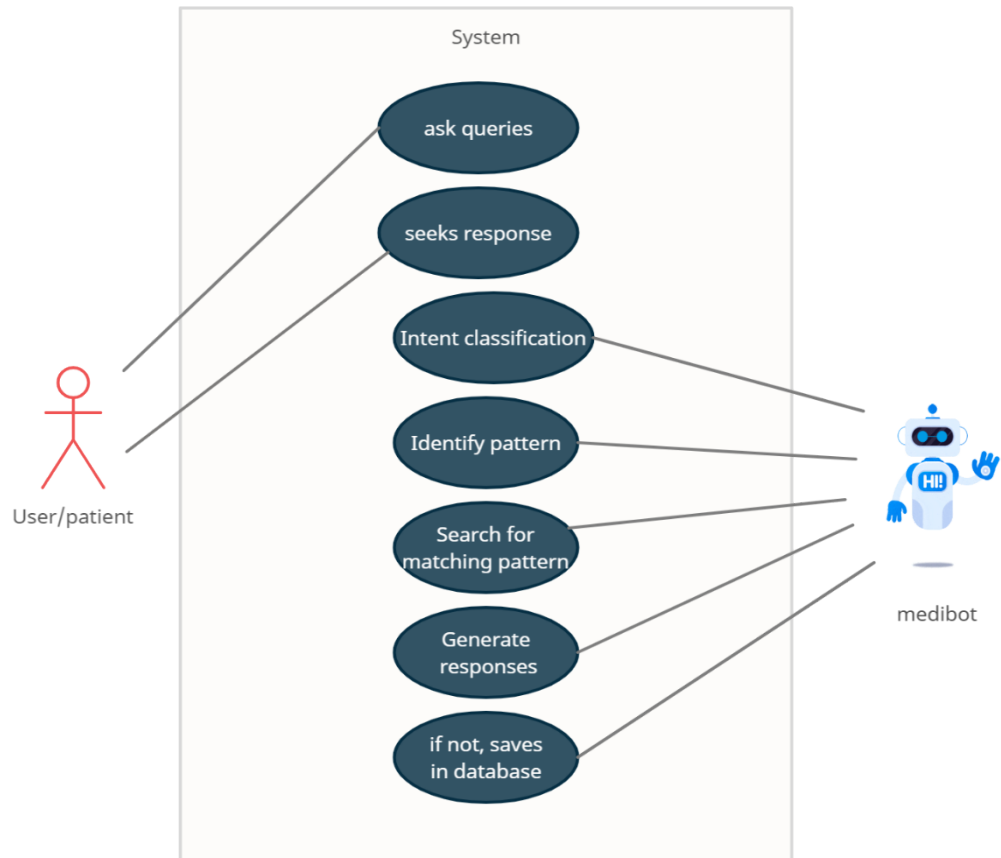


FIG-2

Process Diagram Between Patient and Medibot

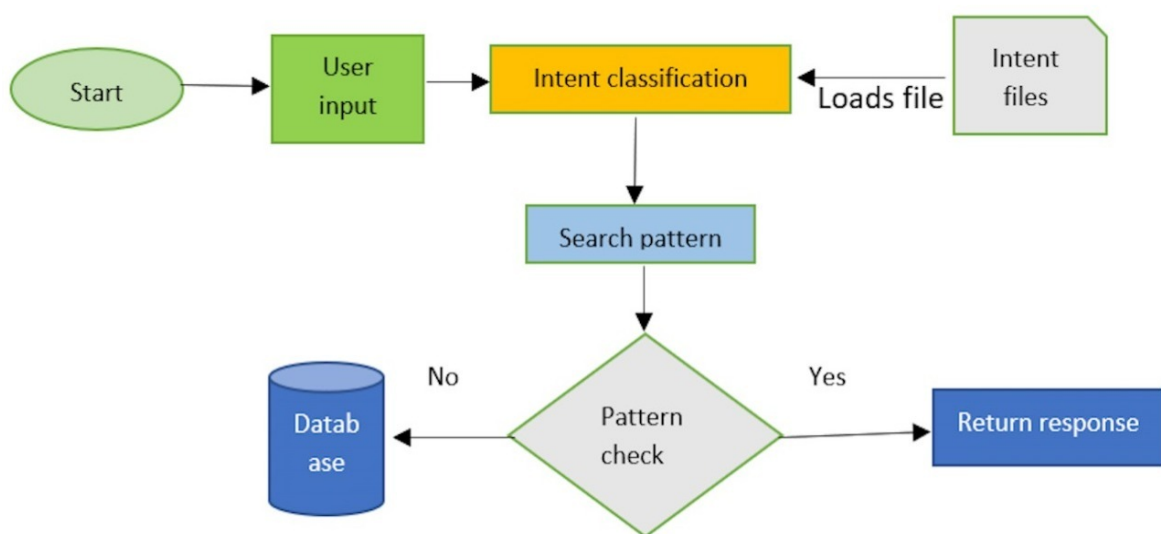


FIG-3

Design Flow Diagram

In this Design Flow Diagram first, it has to start and the user had to send input to intent classification. From intent files, data will be sent to intent classification. Then after it goes to search pattern where it searches for the correct pattern in pattern check if it is yes it goes to return response if not data base. In the next section going to explain the literature review in which know more about the medibot.

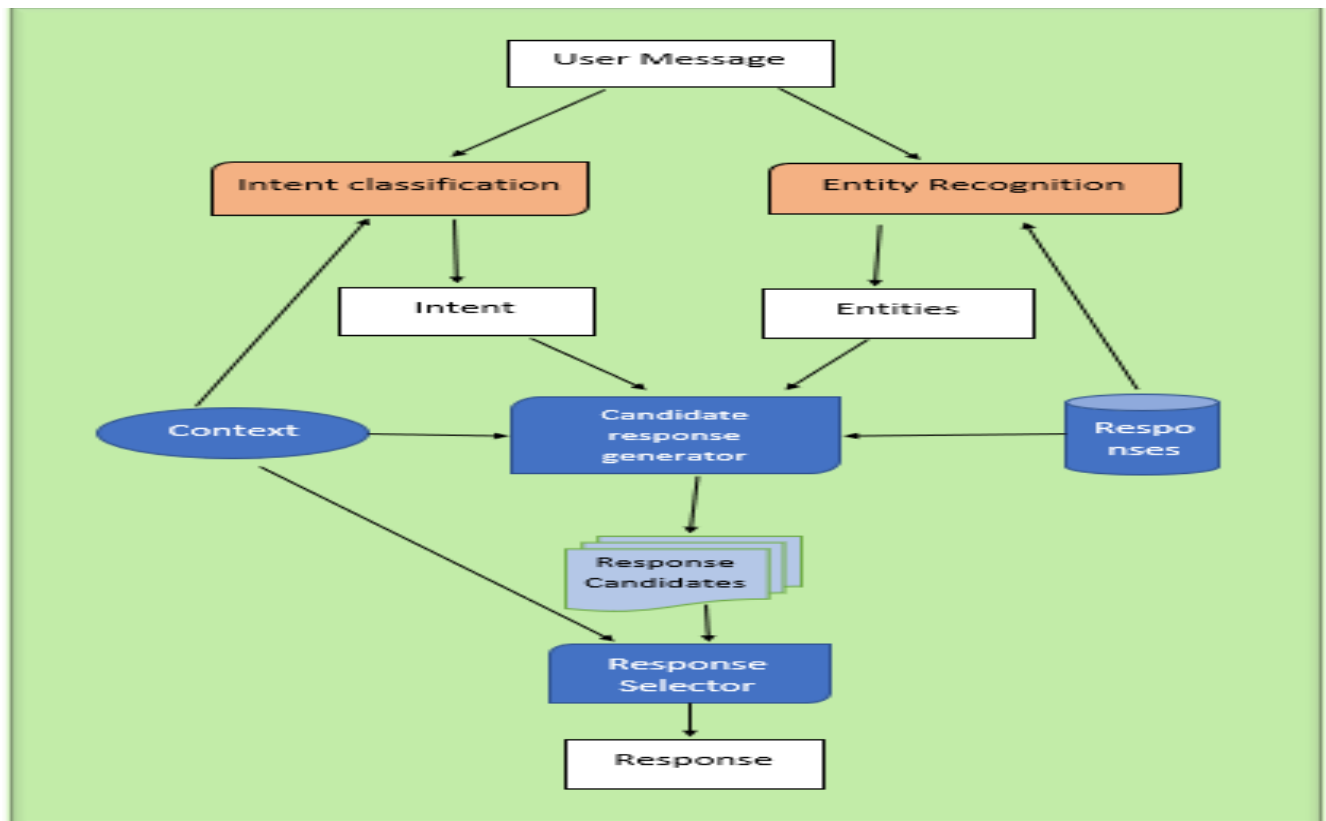


FIG-4

System Architecture

important for the development of a system. This medibot provides proper guidance to patients who have no awareness of their physical condition. Many people hesitate to ask doctors about symptoms with chatbots so they can solve their problems. The below fig 3 gives us working of the system.

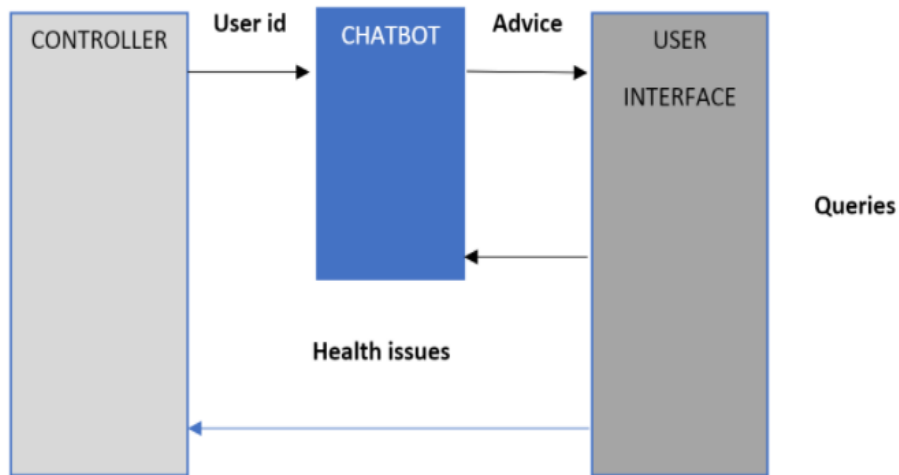


FIG-5

Processing steps for chatbot

According to the above diagram User, messages will be classified into Intent and Entities by Intent and Entities classification. After it will go to the response generator where it will send a response to candidates from context and responses. Response selector will select a response from the above responses shows in fig.4

CHAPTER 4

RESULT AND DISCUSSION

Model Evaluation is an integral part of the model development process. It helps to find best model that represents the data and how well the chosen model will work in the future. Evaluating model performance with the data used for training is not acceptable in data science because it can easily generate overoptimistic and overfitted models.

4.1 Evaluation Metrics and Result

Model evaluation metrics are required to quantify model performance. The choice of model evaluation metrics depends on a given machine learning task.

In this project evaluation metrics are:

- 1) Accuracy
- 2) Loss

Sample Code for Evaluating Model

```
model.evaluate(np.array(train_x), np.array(train_y))
```

As a result of evaluation using model.evaluate() method,

- 1- Accuracy of the model is depicted as 1.0000
- 2- Loss of the model is depicted as 0.00073455

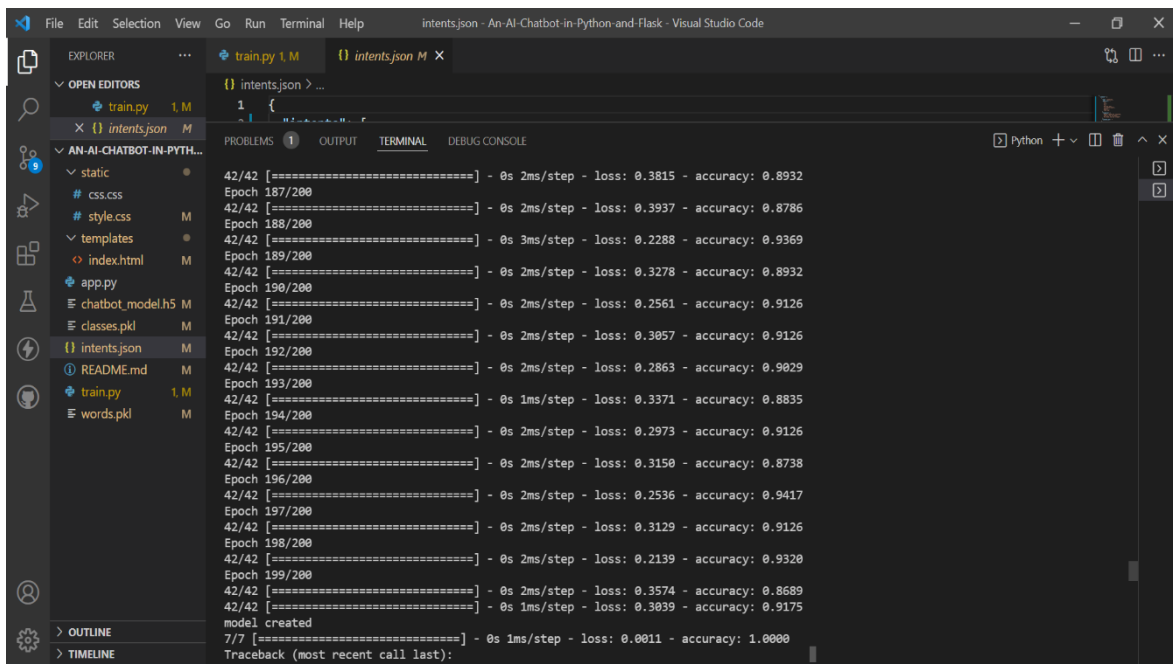


FIG-6

4.2. Training and Testing Data

We will create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. we will convert text into numbers.

create our training data

training = []

create an empty array for our output

output_empty = [0] * len(classes)

training set, bag of words for each sentence

for doc in documents:

initialize our bag of words

bag = []

list of tokenized words for the pattern

pattern_words = doc[0]

lemmatize each word - create base word, in attempt to represent related words

pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]

create our bag of words array with 1, if word match found in current pattern

for w in words:

```

bag.append(1) if w in pattern_words else bag.append(0)
# output is a '0' for each tag and '1' for current tag (for each pattern)
output_row = list(output_empty)
output_row[classes.index(doc[1])] = 1
training.append([bag, output_row])
# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0])
train_y = list(training[:,1])
print ("Training data created")

```

4.3. Building the Model

We have our training data ready, now we will build a deep neural network that has 3 layers. We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model. Let us save the model as 'chatbot_model.h5'.

```

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and
3rd output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
# Compile model. Stochastic gradient descent with Nesterov accelerated gradient
gives good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy'])

```

#fitting and saving the model

```
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5,
verbose=1)
model.save('chatbot_model.h5', hist)
print("model created")
```

4.3.1 Predict the response (Graphical User Interface)

To predict the sentences and get a response from the user to let us create a new file 'chatapp.py'.

We will load the trained model and then use a graphical user interface that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.

Again we import the necessary packages and load the 'words.pkl' and 'classes.pkl' pickle files which we have created when we trained our model:

```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np
from keras.models import load_model
model = load_model('chatbot_model.h5')
import json
import random
intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))
```

To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text pre-processing and then predict the class.

```
def clean_up_sentence(sentence):
# tokenize the pattern - split words into array
```

```

sentence_words = nltk.word_tokenize(sentence)
# stem each word - create short form for word
sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]
return sentence_words
# return bag of words array: 0 or 1 for each word in the bag that exists in the
sentence
def bow(sentence, words, show_details=True):
# tokenize the pattern
sentence_words = clean_up_sentence(sentence)
# bag of words - matrix of N words, vocabulary matrix
bag = [0]*len(words)
for s in sentence_words:
for i,w in enumerate(words):
if w == s:
# assign 1 if current word is in the vocabulary position
bag[i] = 1
if show_details:
print ("found in bag: %s" % w)
return(np.array(bag))
def predict_class(sentence, model):
# filter out predictions below a threshold
p = bow(sentence, words,show_details=False)
res = model.predict(np.array([p]))[0]
ERROR_THRESHOLD = 0.25
results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
# sort by strength of probability
results.sort(key=lambda x: x[1], reverse=True)
return_list = []
for r in results:
return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
return return_list

```

After predicting the class, we will get a random response from the list of intents.

```

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag']== tag):
            result = random.choice(i['responses'])
            break
    return result

def chatbot_response(text):
    ints = predict_class(text, model)
    res = getResponse(ints, intents)
    return res

```

Now we will develop a graphical user interface. Let's use Tkinter library which is shipped with tons of useful libraries for GUI. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI. Here is the full source code for the GUI.

#Creating GUI with tkinter

```

import tkinter
from tkinter import *
def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)
    if msg != "":
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))
        res = chatbot_response(msg)
        ChatLog.insert(END, "Bot: " + res + '\n\n')
        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)
base = Tk()
base.title("Hello")
base.geometry("400x500")

```

```

base.resizable(width=FALSE, height=FALSE)
#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)
ChatLog.config(state=DISABLED)
#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set
#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12",
height=5,
bd=0, bg="#32de97", activebackground="#3c9d9b",fg='ffffff',
command= send )
#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)
#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)
base.mainloop()

```

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION:

The Intent of this paper is to increase the awareness of health among the people. In current days, many people show their lazy behavior and don't consult a doctor during a time of illness so the implementation of a chatbot will help the people to diagnose the disease without consulting a doctor. The chatbot will act as a virtual doctor. The user will prescribe their symptoms of their illness and the chatbot will analyze the disease and suggest the necessary healthcare steps that need to be taken. In the datasets it includes information regarding diseases and health care steps.

5.2 FUTURE WORK:

Based on the system development and extensibility in future can also implement audio and face recognition to users benefits and also interact will doctors in case of a patient's emergency for treatment.

REFERENCES:

- [1] Srivastava, P., & Singh, N (2020, February). Automatized medical chatbot (medibot). In 2020 International Conference on Power Electronics & IoT Applications in Renewable Energy and its Control (PARC) (pp. 351-354). IEEE.
- [2] Kandpal, P., Jasnani, K., Raut, R., & Bhorge, S. (2020, July). Contextual Chatbot for healthcare purposes (using deep learning). In 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4) (pp. 625-634). IEEE.
- [3] Athota, L., Shukla, V. K., Pandey, N., & Rana, A. (2020, June). Chatbot for Healthcare System Using Artificial Intelligence. In 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO) (pp. 619-622). IEEE.
- [4] Karri, S. P. R., & Kumar, B. S. (2020, January). Deep learning techniques for implementation of chatbots. In 2020 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-5). IEEE.
- [5] Bharti, U., Bajaj, D., Batra, H., Lalit, S., Lalit, S., & Gangwani, A. (2020, June). Medbot: Conversational artificial intelligence powered chatbot for delivering tele-health after covid-19. In 2020 5th International Conference on Communication and Electronics Systems (ICCES) (pp. 870-875). IEEE.
- [6] Milla T Mutiwokuziva, Melody W Chanda, Prudence Kadebu, Addlight Mukwazvure, Tatenda T Gatora, "A Neural-network based Chat Bot" in ICCES,2017
- [7] Gentner, T., Neitzel, T., Schulze, J., & Buettner, R. (2020, July). A Systematic literature review of medical chatbot research from a behavior change perspective. In 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC) (pp. 735-740). IEEE.
- [8] [https://www.investopedia.com/terms/d/deeplearning.asp#:~:text=Deep%20learning%20is%20a%20subset,le arning%20or%20deep%20neural%20network](https://www.investopedia.com/terms/d/deeplearning.asp#:~:text=Deep%20learning%20is%20a%20subset,le%20arning%20or%20deep%20neural%20network).
- [9] Gupta, J., Singh, V., & Kumar, I. (2021, March). Florence-A Health Care Chatbot. In 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS) (Vol. 1, pp. 504-508). IEEE.

- [10] S. Divya, Indumathi, S. Ishwarya, M. Priyasankari, S. Kalpanadevi | A Self-Diagnosis Medical Chatbot Using Artificial Intelligence | Institute of Electrical and Electronics Engineers June 2019 Softić, A., Husić, J. B., Softić, A., & Baraković, S. (2021, March). Health Chatbot: Design, Implementation, Acceptance and Usage Motivation. In 2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH) (pp. 1-6). IEEE.
- [11] Badlani, S., Aditya, T., Dave, M., & Chaudhari, S. (2021, May). Multilingual Healthcare Chatbot Using Machine Learning. In 2021 2nd International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.
- [12] Madhu, D., Jain, C. N., Sebastain, E., Shaji, S., & Ajayakumar, A. (2017, March). A novel approach for medical assistance using trained chatbot. In 2017 international conference on inventive communication and computational technologies (ICICCT) (pp. 243-246). IEEE.
- [13] Sunny, A. D., Kulshreshtha, S., Singh, S., Srinabh, B. M., & Sarojadevi, H. (2018). Disease diagnosis system by exploring machine learning algorithms. *Int. J. Innov. Eng. Technol*, 10(2), 14-21.
- [14] Kandpal, P., Jasnani, K., Raut, R., & Bhorge, S. (2020, July). Contextual Chatbot for healthcare purposes (using deep learning). In 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4) (pp. 625-634). IEEE.
- [15] Sophia, J. J., Kumar, D. A., Arutselvan, M., & Ram, S. B. (2020). A survey on chatbot implementation in health care using NLTK. *Int. J. Comput. Sci. Mob. Comput*, 9.
- [16] Hwang, T. H., Lee, J., Hyun, S. M., & Lee, K. (2020, October). Implementation of interactive healthcare advisor model using chatbot and visualization. In 2020
- [17] International Conference on Information and Communication Technology Convergence (ICTC) (pp. 452-455). IEEE
- [18] Avila, C. V. S., Franco, W., Venceslau, A. D., Rolim, T. V., & MP, V. (2021). MediBot: An Ontology-Based Chatbot to Retrieve Drug Information and Compare its Prices.
- [19] Sivaraj, K., Jeyabalasuntharam, K., Ganeshan, H., Nagendran, K., Alosious, J., & Tharmaseelan, J. Medibot: End to end voice based AI medical chatbot with a smart watch.

- [20] Deherkar, K. (2018). MediBot: A predictor and analyzer for CVD. Asian Journal For Convergence In Technology (AJCT) ISSN-2350-1146.
- [21] KC, G. P., Ranjan, S., Ankit, T., & Kumar, V. (2019). A personalized medical assistant chatbot: Medibot. Int. J. Sci. Technol. Eng, 5(7).
- [22] Kazi, H., Chowdhry, B. S., & Memon, Z. (2012). MedChatBot: an UMLS based chatbot for medical students. International Journal of Computer Applications, 55(17).
- [23] Kazi, H., Chowdhry, B. S., & Memon, Z. (2012). MedChatBot: an UMLS based chatbot for medical students. International Journal of Computer Applications, 55(17).
- [24] Mathew, R. B., Varghese, S., Joy, S. E., & Alex, S. S. (2019, April). Chatbot for disease prediction and treatment recommendation using machine learning. In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 851-856). IEEE.
- [25] Rahman, M. M., Amin, R., Liton, M. N. K., & Hossain, N. (2019, December). Disha: An implementation of machine learning based Bangla healthcare Chatbot. In 2019 22nd International Conference on Computer and Information Technology (ICCIT) (pp. 1-6). IEEE.
- [26] Ayanouz, S., Abdelhakim, B. A., & Benhmed, M. (2020, March). A smart chatbot architecture based NLP and machine learning for health care assistance. In Proceedings of the 3rd International Conference on Networking, Information Systems & Security (pp. 1-6).
- [27] Oh, K. J., Lee, D., Ko, B., & Choi, H. J. (2017, May). A chatbot for psychiatric counseling in mental healthcare service based on emotional dialogue analysis and sentence generation. In 2017 18th IEEE International Conference on Mobile Data Management (MDM) (pp. 371-375). IEEE.
- [28] Badlani, S., Aditya, T., Dave, M., & Chaudhari, S. (2021, May). Multilingual Healthcare Chatbot Using Machine Learning. In 2021 2nd International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.

APPENDIX

A. SOURCE CODE:

1. App.py code:

```
# libraries

import random

import numpy as np

import pickle

import json

from flask import Flask, render_template, request

from flask_ngrok import run_with_ngrok

import nltk

from keras.models import load_model

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# chat initialization

model = load_model("chatbot_model.h5")

intents = json.loads(open("intents.json").read())

words = pickle.load(open("words.pkl", "rb"))

classes = pickle.load(open("classes.pkl", "rb"))

app = Flask(__name__)
```

```
run_with_ngrok(app)
```

```
@app.route("/")
```

```
def home():
```

```
    return render_template("index.html")
```

```
@app.route("/get", methods=["POST"])
```

```
def chatbot_response():
```

```
    msg = request.form["msg"]
```

```
    if msg.startswith('my name is'):
```

```
        name = msg[11:]
```

```
        ints = predict_class(msg, model)
```

```
        res1 = getResponse(ints, intents)
```

```
        res = res1.replace("{n}", name)
```

```
    elif msg.startswith('hi my name is'):
```

```
        name = msg[14:]
```

```
        ints = predict_class(msg, model)
```

```
        res1 = getResponse(ints, intents)
```

```
        res = res1.replace("{n}", name)
```

```
    else:
```

```
        ints = predict_class(msg, model)
```

```
        res = getResponse(ints, intents)
```

```
    return res
```

```
# chat functionalities
```

```

def clean_up_sentence(sentence):

    sentence_words = nltk.word_tokenize(sentence)

    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]

    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the
sentence

def bow(sentence, words, show_details=True):

    # tokenize the pattern

    sentence_words = clean_up_sentence(sentence)

    # bag of words - matrix of N words, vocabulary matrix

    bag = [0] * len(words)

    for s in sentence_words:

        for i, w in enumerate(words):

            if w == s:

                # assign 1 if current word is in the vocabulary position

                bag[i] = 1

                if show_details:

                    print("found in bag: %s" % w)

    return np.array(bag)

def predict_class(sentence, model):

    # filter out predictions below a threshold

    p = bow(sentence, words, show_details=False)

    res = model.predict(np.array([p]))[0]

```

```
ERROR_THRESHOLD = 0.25
```

```
results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
```

```
# sort by strength of probability
```

```
results.sort(key=lambda x: x[1], reverse=True)
```

```
return_list = []
```

```
for r in results:
```

```
    return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
```

```
return return_list
```

```
def getResponse(ints, intents_json):
```

```
    tag = ints[0]["intent"]
```

```
    list_of_intents = intents_json["intents"]
```

```
    for i in list_of_intents:
```

```
        if i["tag"] == tag:
```

```
            result = random.choice(i["responses"])
```

```
            break
```

```
    return result
```

```
if __name__ == "__main__":
```

```
    app.run()
```

2. train.py code:

```
# libraries

import random

from tensorflow.keras.optimizers import Adam, SGD

from keras.layers import Dense, Dropout

from keras.models import load_model

from keras.models import Sequential

from keras.utils.vis_utils import plot_model


import numpy as np

import pickle

import json

import nltk

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

# nltk.download("punkt")

# nltk.download("wordnet")


# init file

words = []
```

```

classes = []

documents = []

ignore_words = ["?", "!"]

data_file = open("intents.json").read()

intents = json.loads(data_file)


# words

for intent in intents["intents"]:

    for pattern in intent["patterns"]:

        # take each word and tokenize it

        w = nltk.word_tokenize(pattern)

        words.extend(w)

    # adding documents

    documents.append((w, intent["tag"]))


# adding classes to our class list

if intent["tag"] not in classes:

    classes.append(intent["tag"])


# lemmatizer

words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]

words = sorted(list(set(words)))

```

```

classes = sorted(list(set(classes)))

print(len(documents), "documents")

print(len(classes), "classes", classes)

print(len(words), "unique lemmatized words", words)


pickle.dump(words, open("words.pkl", "wb"))
pickle.dump(classes, open("classes.pkl", "wb"))


# training initializer

# initializing training data
training = []

output_empty = [0] * len(classes)

for doc in documents:

    # initializing bag of words

    bag = []

    # list of tokenized words for the pattern

    pattern_words = doc[0]

```

```

# lemmatize each word - create base word, in attempt to represent related
words

pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]

# create our bag of words array with 1, if word match found in current pattern

for w in words:

    bag.append(1) if w in pattern_words else bag.append(0)


# output is a '0' for each tag and '1' for current tag (for each pattern)

output_row = list(output_empty)

output_row[classes.index(doc[1])] = 1


training.append([bag, output_row])

# shuffle our features and turn into np.array

random.shuffle(training)

training = np.array(training)

# create train and test lists. X - patterns, Y - intents

train_x = list(training[:, 0])

train_y = list(training[:, 1])

print("Training data created")


# actual training

# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and
3rd output layer contains number of neurons

# equal to number of intents to predict output intent with softmax

```

```

model = Sequential()

model.add(Dense(128, input_shape=(len(train_x[0]),), activation="relu"))

model.add(Dropout(0.5))

model.add(Dense(64, activation="relu"))

model.add(Dropout(0.5))

model.add(Dense(len(train_y[0]), activation="softmax"))

model.summary()


# Compile model. Stochastic gradient descent with Nesterov accelerated gradient
gives good results for this model

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss="categorical_crossentropy", optimizer=sgd,
metrics=["accuracy"])


# for choosing an optimal number of training epochs to avoid underfitting or
overfitting use an early stopping callback to keras

# based on either accuracy or loss monitoring. If the loss is being monitored,
training comes to halt when there is an

# increment observed in loss values. Or, If accuracy is being monitored, training
comes to halt when there is decrement observed in accuracy values.


# from keras import callbacks

# earlystopping = callbacks.EarlyStopping(monitor ="loss", mode ="min", patience
= 5, restore_best_weights = True)

# callbacks =[earlystopping]

```

```
# fitting and saving the model
```

```
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5,  
verbose=1)
```

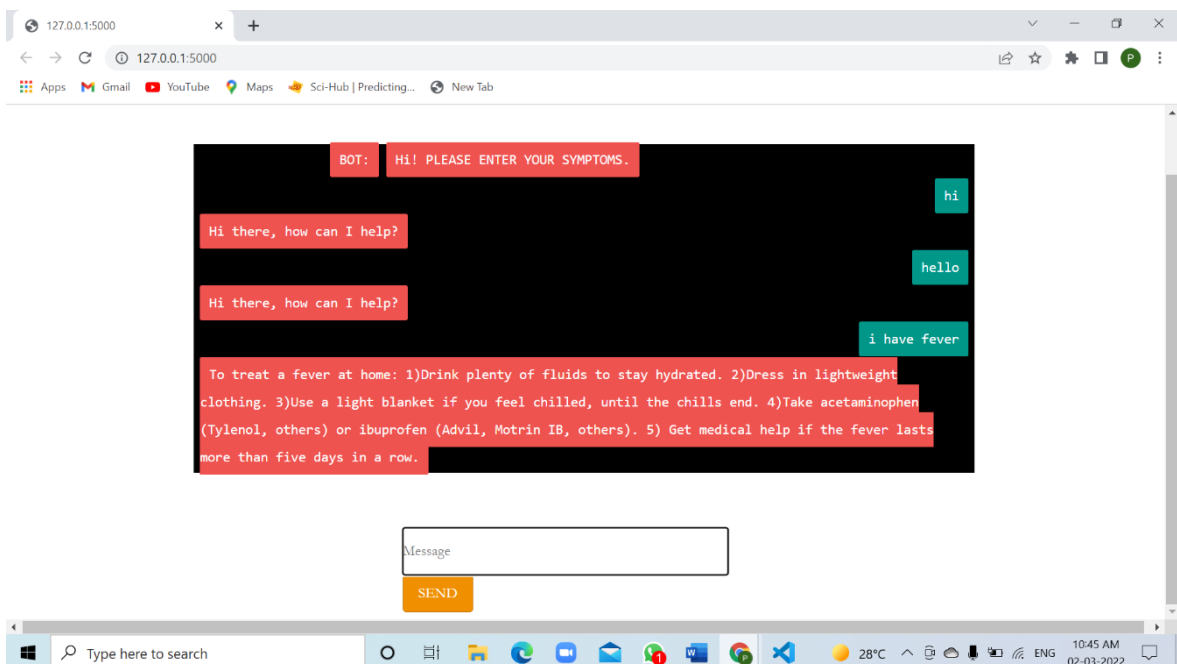
```
model.save("chatbot_model.h5", hist)
```

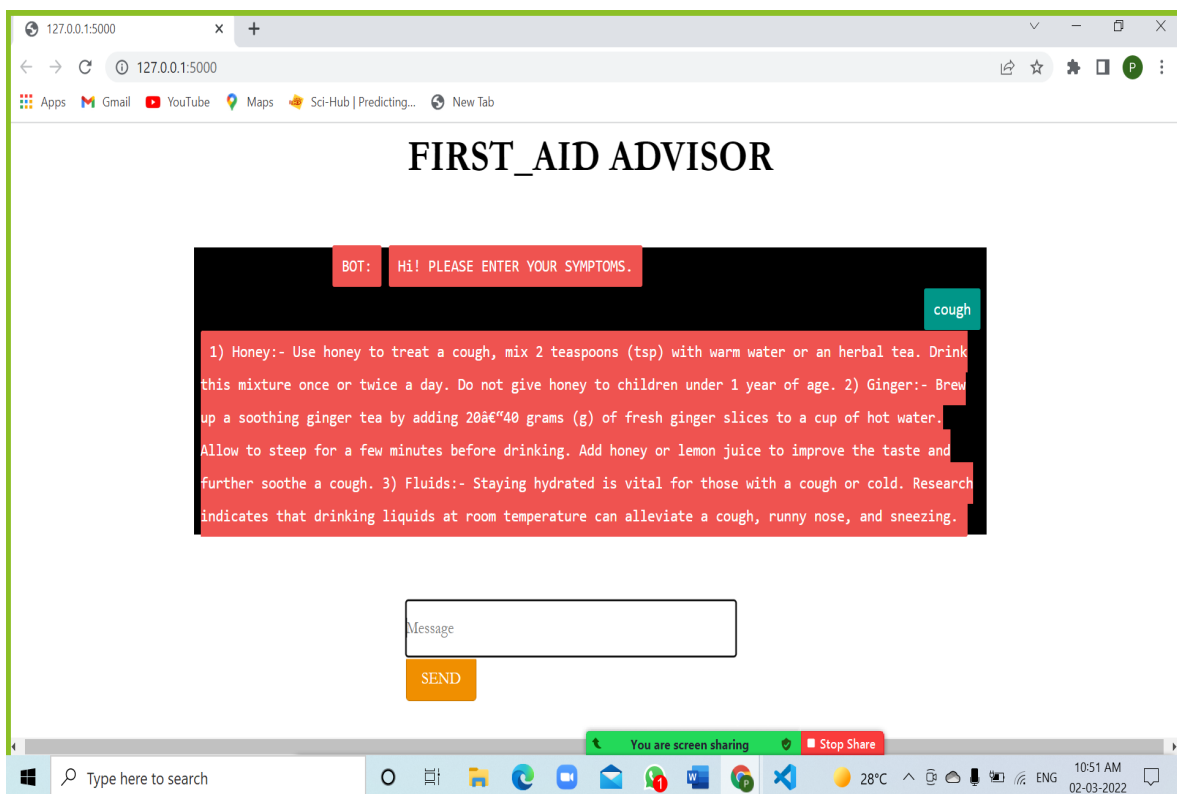
```
print("model created")
```

```
model.evaluate(np.array(train_x), np.array(train_y))
```

```
plot_model(model, to_file='model_plot.png', show_shapes=True,  
show_layer_names=True)
```

B. Screenshots





C. PUBLICATION WITH PLAGARISM REPORT

publication (1) (1).doc			
ORIGINALITY REPORT			
9%	5%	6%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
PRIMARY SOURCES			
1	Submitted to University of Hertfordshire Student Paper	2%	
2	Submitted to Monash University Student Paper	1%	
3	Submitted to Central Queensland University Student Paper	1%	
4	www.researchgate.net Internet Source	1%	
5	www.wgu.edu Internet Source	1%	
6	"Neural Information Processing", Springer Science and Business Media LLC, 2021 Publication	1%	
7	Salah Bouktif, Ayisha Manzoor. "Artificial Intelligence as a Gear to Preserve Effectiveness of Learning and Educational Systems in Pandemic Time", 2021 IEEE Global Engineering Education Conference (EDUCON), 2021 Publication	<1%	