MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

Project Report

on

Desktop Voice Assistant

Submitted By:

Rubin Jain

0901AD211044

Chandan Jat

0901AD211009

Faculty Mentor:

Dr. Sunil Kumar Shukla

CENTRE FOR ARTIFICIAL INTELLIGENCE

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE
GWALIOR - 474005 (MP) est. 1957

JULY-DEC. 2023

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

CERTIFICATE

This is certified that **Rubin Jain** (0901AD211044) and **Chandan Jat** (0901AD211009) has submitted the project report titled **Desktop Voice Assistant** under the mentorship of **Dr. Sunil Kumar Shukla**, in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in **Artificial Intelligence and Data Science** from Madhav Institute of Technology and Science, Gwalior.

23/11/2023

Dr. Sunil Kumar Shukla
Faculty Mentor
Assistant Professor
Centre for Artificial Intelligence

28/11/23

Dr. R. R. Singh
Coordinator
Centre for Artificial Intelligence

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in **Artificial Intelligence And Data Science** at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Dr. Sunil Kumar Shukla**, **Assistant Professor**, Centre for Artificial Intelligence.

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Rubin Jain
0901AD211044
3rd Year,
Centre for Artificial Intelligence

Chandan Jat
0901AD211009
3rd Year,
Centre for Artificial Intelligence

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)

NAAC Accredited with A++ Grade

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Centre for Artificial Intelligence,** for allowing me to explore this project. I humbly thank **Dr. R. R. Singh**, Coordinator, Centre for Artificial Intelligence, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentors. I am grateful to the guidance of **Dr. Sunil Kumar Shukla**, **Assistant Professor**, Centre for Artificial Intelligence, for his continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

Rubin Jain
0901AD211044
3rd Year,
Centre for Artificial Intelligence

Chandan Jat
0901AD211009
3rd Year,
Centre for Artificial Intelligence

ABSTRACT

The project introduces "Echo," a desktop voice assistant powered by natural language processing and machine learning techniques. Echo is designed to enhance user interaction with the computer, providing hands-free control and assistance. The system integrates speech recognition, intent recognition, and task execution to understand and respond to user commands effectively.

The core functionality is driven by a neural network model trained on a diverse set of intents. The training data, comprising user queries and associated intents, is structured in the form of a JSON file. The neural network employs bag-of-words representation to recognize user intents, enabling Echo to respond appropriately to various commands and inquiries.

Echo's capabilities span a wide range of tasks, including providing real-time information such as time, date, and day, executing Wikipedia searches, conducting Google searches, opening websites, YouTube Automation and facilitating interactions on WhatsApp. The system leverages external libraries and modules for functionalities like speech synthesis (`speaking` module), speech recognition (`Hearing` module), and task-specific operations (`Task.py` module).

The project adopts a modular architecture, allowing for easy extension of capabilities by adding new intents and corresponding actions. The continuous loop structure ensures Echo remains active, awaiting user input and delivering prompt responses.

In conclusion, Echo offers a versatile and user-friendly voice-controlled interface, bridging the gap between users and their desktop environment. The system's adaptability and expandability make it a promising tool for simplifying tasks and enhancing the overall user experience. Future developments may focus on refining existing features, expanding the range of supported actions, and further optimizing the system's performance.

Keyword: Desktop Voice Assistant**,** Natural Language Processing (NLP)**,** Machine Learning**,** Neural Network**,** Intent Recognition**,** Bag-of-Words Representation**,** Speech Recognition**,** Task Automation**,** Speech Synthesis**,** Task-specific Operations**,** Intent-Action Mapping**,** Speech-to-Text**,** Text-to-Speech**,** Python Programming**,** JSON Data Structure.

सार:

यह परियोजना एक डेस्कटॉप आवाज सहायक समाधान है जो नैतिक भाषा प्रसंस्करण (**NLP**), मशीन लर्निंग, और न्यूरल नेटवर्क का उपयोग करके उपयोगकर्ता को उनके डेस्कटॉप को बोलकर संवाद करने की सुविधा प्रदान करता है। इस आवाज सहायक ने उच्च-स्तरीय उदाहरणों और क्रियाओं का उपयोग करके बातचीत की समझ को बढ़ाया है, जैसे कि इंटेंट पहचान और क्रिया निर्धारण, नैतिक भाषा प्रसंस्करण के लिए टास्क, और विशेष कार्रवाई की पहचान करना।

इस परियोजना ने वाक्यांशों को शब्दों में ट्रांसफ़ॉर्म करने के लिए एक बैग-ऑफ-वर्ड्स प्रस्तुत करके न्यूरल नेटवर्क मॉडल को विकसित किया है, जो उपयोगकर्ता के बोले गए वाक्यों को समझता है और उचित इंटेंट और क्रिया में क्रियाशील होता है। इसमें विभिन्न इंटेंट्स और क्रियाओं को निर्धारित करने के लिए उपयोगकर्ता द्वारा बोले गए वाक्यों को विभाजित करने के लिए एक ज्ञान का उपयोग किया गया है, जिससे यह उपयोगकर्ता को विभिन्न कार्रवाईयों को सरलता से नियंत्रित करने का संवाद प्रदान करता है।

इस परियोजना में मोड्यूलर और परिष्कृत संरचना का उपयोग हुआ है, जिससे नए इंटेंट्स और क्रियाएं आसानी से जोड़ी जा सकती हैं, और यह उपयोगकर्ता को उनके डेस्कटॉप कार्यों को सरलता से नियंत्रित करने का एक उचित तरीका प्रदान करता है।

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: Introduction and Objective

1.1 Introduction

In the dynamic landscape of contemporary technology, the advent of intelligent voice assistants has redefined the way humans interact with computing systems. This project introduces "Echo," an advanced desktop assistant that amalgamates cutting-edge technologies, including machine learning, natural language processing (NLP), and a repertoire of Python libraries, to create an intelligent and responsive voice interface.

Voice-enabled assistants have witnessed a transformative evolution, transcending from mere curiosities to integral components of our daily lives. With Echo, we aim to contribute to this evolution by crafting a versatile and powerful voice assistant that seamlessly integrates into the user's desktop environment. Echo is designed to understand and respond to natural language commands, providing users with an intuitive and hands-free method of interacting with their computers.

1.2 Libraries Used

The Echo voice assistant integrates a diverse set of Python libraries, each playing a crucial role in enhancing its functionality:

I. **torch:** This library forms the backbone for constructing and training neural networks, serving as the core engine driving Echo's intelligent responses.

II. **pytube:** Enabling seamless interactions with YouTube, `pytube` empowers Echo to perform tasks such as video searching and downloading.

III. **pywhatkit:** With this library, Echo executes web searches and manages operations on the YouTube platform, expanding its capabilities in online content retrieval.

IV. **wikipedia:** Providing access to a vast pool of information, `wikipedia` enables Echo to retrieve detailed and accurate information from Wikipedia.

V. **webbrowser:** Essential for opening websites, this library allows Echo to navigate the web and fetch information as needed.

VI. **keyboard:** Through the simulation of keyboard inputs, this library facilitates various tasks, including controlling media playback and executing specific commands.

VII. **pyautogui:** Automation of mouse and keyboard actions is made possible by `pyautogui`, enhancing Echo's ability to interact with the desktop environment.

VIII. **datetime:** Utilized for time and date-related functionalities, the `datetime` library enhances Echo's capacity to provide real-time information.

IX. **json:** Essential for working with JSON data, the `json` library is employed for loading and processing intent data from external files.

X. **os:** The `os' library is used for interacting with the operating system, facilitating tasks such as file operations and system-level interactions.

XI. **random:** For introducing an element of randomness, the `random` library is utilized to select responses or actions in certain scenarios.

XII. **`YouTube` (from `pytube`):** Specifically utilized for fetching video-related information from YouTube during interactions.

XIII. **datetime:** Enhancing Echo's capability to provide real-time information, the `datetime` library is crucial for time and date-related functionalities.

1.3 Purpose and Scope

Purpose:

The primary objective of Echo is to deliver a sophisticated and user-friendly voice assistant that leverages advanced technologies to enhance human-computer interactions. By combining machine learning, natural language processing, and a carefully curated selection of Python libraries, Echo aspires to redefine the way users engage with their desktop environments. Through a conversational interface, Echo aims to offer a seamless and intuitive experience, empowering users to execute a wide array of tasks with voice commands.

Scope:

I. *Voice-Activated Tasks:* Echo excels in executing commands through natural language input, eliminating the need for manual interactions and providing users with a hands-free computing experience.

II. *Versatile Functionality:* Echo's capabilities extend across various domains, encompassing information retrieval from Wikipedia, conducting Google searches, managing YouTube video operations, controlling media playback, accessing real-time data (time, date, day), and interacting with messaging platforms like WhatsApp.

III. *Extensibility:* The modular architecture of Echo ensures adaptability, allowing for the seamless integration of new features and functionalities. This ensures that Echo can evolve to meet emerging user requirements and technological advancements.

IV. *Enhanced User Experience:* By offering a conversational interface, Echo aims to elevate the overall user experience. The goal is to make the interaction with the desktop assistant accessible, enjoyable, and suitable for a diverse user base.

V. *Open-Source Nature:* Echo embraces an open-source ethos, encouraging collaboration and inviting developers to contribute to its continual growth and improvement. This open nature facilitates the expansion of Echo's capabilities through community-driven efforts.

1.4 Enhanced User Interaction and Productivity

One of the primary goals of Echo is to facilitate enhanced user interaction in a UI-less environment. By relying on advanced voice recognition and natural language processing, Echo strives to make user interactions more intuitive, efficient, and productive. The absence of a traditional graphical interface places a premium on voice commands, making it imperative to optimize this aspect for a seamless and user-friendly experience.

1.5 Customization and Extensibility

Echo is designed with a focus on customization and extensibility to cater to diverse user preferences and requirements. Users should have the flexibility to tailor Echo to their specific needs, integrating additional functionalities and expanding its capabilities. This objective ensures that Echo is not a one-size-fits-all solution but rather a versatile tool that can adapt to various user contexts and scenarios.

1.6 Information Retrieval and Versatility

A crucial aspect of Echo's objectives is its proficiency in information retrieval and versatility. Users should be able to obtain relevant and accurate information through natural language queries. Whether it's retrieving data from the web, accessing local information, or providing insightful responses, Echo aims to be a comprehensive source of information, showcasing its versatility across a spectrum of user inquiries.

1.7 Speech Recognition and User Experience

The quality of speech recognition directly influences the overall user experience with a voice assistant. Echo places a strong emphasis on refining its speech recognition capabilities to accurately understand and interpret user commands. By prioritizing a seamless and reliable speech recognition system, Echo aims to elevate the overall user experience, ensuring that interactions are effortless, precise, and frustration-free.

Chapter 2 – Literature Review

Voice assistants have become an integral part of everyday life, leveraging artificial intelligence to enhance user experience and streamline various tasks. This literature review explores several studies and projects that delve into the development and applications of voice assistants.

[1] Agrawal et al. (2023) present a comprehensive study titled "Voice Assistant Using Python." Their work focuses on the implementation of a voice assistant and is notable for its unique approach. The authors discuss the technical aspects of their Python-based voice assistant, providing insights into the programming intricacies involved in creating such systems. This hands-on approach contributes valuable practical knowledge to the field.

[2] Terzopoulos and Satratzemi (Year) bring a broader perspective by examining the role of voice assistants in everyday life and education. Their work, conducted at the University of Macedonia, Greece, investigates the impact of voice assistants and smart speakers on user routines and educational processes. This study is valuable for understanding the societal implications and educational applications of voice assistant technology.

[3] Shende et al. (2019) contribute to the literature with their work on an AI-based voice assistant using Python. The authors explore the integration of artificial intelligence into voice assistants, adding a layer of sophistication to the system's capabilities. This study highlights the evolving nature of voice assistant technology and its intersection with artificial intelligence.

[4] Tulshan and Dhage (2019) present a survey on popular virtual assistants such as Google Assistant, Siri, Cortana, and Alexa. This comprehensive review provides insights into the strengths and weaknesses of existing voice assistants. Understanding the landscape of virtual assistants is crucial for the continuous improvement of these systems, and this survey serves as a valuable resource for researchers and developers.

[5] Kulhalli et al. (2017) contribute to the literature by presenting a "Personal Assistant with Voice Recognition Intelligence." Their work focuses on the integration of voice recognition intelligence into a personal assistant system. This study emphasizes the user-centric design of voice assistants, aiming to enhance personalization and adaptability in real-world scenarios.

In addition to academic publications, the literature also includes practical implementations and tutorials.

[6] The project on "Desktop's Virtual Assistant Using Python" available on ResearchGate provides a hands-on guide for implementing a voice assistant on a desktop platform.

[7] Similarly, the GeeksforGeeks tutorial and [8] the IEEE Xplore document contribute to the literature by offering practical insights and technical details on voice assistant development.

Chapter 3: Methodology

3.1 Technology Selection

3.1.1 Choose Programming Language

The selection of a programming language is a critical decision that influences the efficiency, scalability, and maintainability of the project. Here, we delve into the considerations that led to the choice of a specific programming language. Factors such as community support, available libraries, and the language's alignment with the unique requirements of voice processing tasks are meticulously examined.

3.1.2 Identify Libraries and Tools

An exhaustive analysis of the libraries and tools chosen for Echo's development is presented. We explore the functionality of each tool, its role in voice recognition and natural language processing, and how it aligns with the overarching goals of the project. Considerations for adaptability, ease of integration, and the potential for future enhancements are thoroughly evaluated.

3.2 System Architecture Design

3.2.1 Define System Components

This section dissects the design of Echo's system components, providing a comprehensive understanding of the architecture's building blocks. Each component is scrutinized for its role, interdependencies, and contributions to the overall functionality of the voice assistant.

3.2.2 Specify Data Flow

An in-depth exploration of the data flow within the system is undertaken. This includes a meticulous examination of how information traverses different components, emphasizing the efficiency, reliability, and security of data exchange.

3.2.3 Scalability and Extensibility

Considerations for scalability and extensibility are elucidated. The methodology explores how Echo is designed to handle increased loads, accommodate additional features, and seamlessly integrate with evolving technologies, ensuring longevity and adaptability.

3.3 User Interface Design

3.3.1 Plan Voice and Visual Integration

This segment outlines the meticulous planning involved in integrating voice and visual elements within Echo. Strategies for user interaction in the absence of a traditional graphical user interface are discussed, emphasizing a holistic and inclusive user experience.

3.3.2 Design for Usability

Usability is a cornerstone of effective user interface design. Here, we delve into the principles and practices employed to ensure that Echo is intuitive, user-friendly, and accessible to a diverse user base.

3.3.3 Voice and Visual Integration

The intricacies of seamlessly integrating voice and visual elements are explored. This includes considerations for creating a cohesive and intuitive user experience that optimally leverages both modalities.

3.3.4 Integration with Task Automation

This section delves into the synergies between user interface design and task automation. Emphasis is placed on creating a fluid and intuitive interaction between users and automated tasks, enhancing overall usability.

3.4 Task Automation and Customization

3.4.1 Develop Task Automation Features

The development of task automation features is expounded upon, detailing the identification and implementation of tasks conducive to automation. The methodology ensures that the voice assistant proactively addresses user needs through efficient and intelligent task execution.

3.4.2 Enable User Customization

This section outlines Echo's approach to user customization, empowering users to tailor the voice assistant to their specific needs and preferences. The methodology ensures a personalized and adaptive user experience.

3.4.3 Task Automation Implementation

Technical intricacies of implementing task automation are unveiled, covering the development of scripts or modules that facilitate seamless automated task execution. The section provides insights into ensuring reliability, responsiveness, and adaptability in the automation framework.

3.4.4 Customization for Voice Interaction

Here, we explore the customization options available for voice interactions, allowing users to personalize their interaction patterns and optimize the voice assistant's responsiveness to individual preferences.

3.5 Core Functionality Implementation

3.5.1 Voice Recognition

The linchpin of Echo's functionality, voice recognition, is discussed in granular detail. This includes the selection of algorithms, training processes, and the integration of voice recognition into the overarching system architecture.

3.5.2 Text-to-Speech Conversion

The methodology behind text-to-speech conversion is meticulously explored, detailing the techniques, tools, and libraries employed to convert textual responses into clear and coherent audible speech.

3.5.3 Task Execution and Automation

This section covers the implementation of task execution and automation, elucidating how user commands are translated into actions and automated processes. The focus is on ensuring accuracy, efficiency, and seamlessness in task execution.

Chapter 4: Code Implementation

File 1 – Hearing.py

```
import speech_recognition as sr

def Listen():

    r=sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening....")
        r.pause_threshold=1
        audio=r.listen(source,0,5)
    try:
        print("Recognizing...")
        query=r.recognize_google(audio,language="en-in")
        print(f"You Said:{query}")
    except:
        return ""
    query=str(query)
    return query.lower()
```

fig. 4.1

The code employs the SpeechRecognition library to create a `Listen` function, capturing and converting spoken words into text. It initializes a speech recognizer, configures the microphone, captures audio, and attempts to recognize speech using Google's Web Speech API. The recognized text, in lowercase, is then returned.

File 2 – speaking.py

```
import pyttsx3


def say(Text):
    engine=pyttsx3.init("sapi5")
    voices=engine.getProperty('voices')
    engine.setProperty('voices',voices[0].id)
    engine.setProperty('rate',160)
    print("   ")
    print(Text)
    engine.say(text=Text)
    engine.runAndWait()
    print("   ")
```

fig 4.2

In this file, we utilizes the pyttsx3 library to define a `say` function for text-to-speech synthesis. It initializes the text-to-speech engine, sets voice properties, and speaks the provided text. The engine's voice is configured for the default Windows SAPI5 synthesizer, and the speech rate is set to 160 words per minute.

File 3 – Mind.py

```
import torch.nn as nn

class NeuralNet(nn.Module):

    def __init__(self,input_size,hidden_size,num_classes):
        super(NeuralNet,self).__init__()
        self.l1=nn.Linear(input_size,hidden_size)
        self.l2=nn.Linear(hidden_size,hidden_size)
        self.l3=nn.Linear(hidden_size,num_classes)
        self.relu=nn.ReLU()

    def forward(self,x):
        out=self.l1(x)
        out=self.relu(out)
        out=self.l2(out)
        out=self.relu(out)
        out=self.l3(out)
        return out
```

fig 4.3

This code defines a neural network class (**NeuralNet**) using the PyTorch library. It inherits from **nn.Module** and has three linear layers (**l1**, **l2**, **l3**) with ReLU activation functions (**relu**). The **forward** method specifies the forward pass of the network, indicating how input data **x** is processed through the layers to produce the output. The network is designed for a classification task with **input_size**, **hidden_size**, and **num_classes** as parameters.

File 4 – neural_network.py

```
import numpy as np
import nltk
from nltk.stem.porter import PorterStemmer


custom_nltk_data_directory =
"C:\\Users\\Dell\\AppData\\Local\\Programs\\Python\\Python310\\Lib\\site-packages\\nltk"

nltk.data.path.append(custom_nltk_data_directory)


stemmer=PorterStemmer()

def tokenize(sentence):
    return nltk.word_tokenize(sentence)
```

```
def stem(word):
    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence,words):
    sentence_word=[stem(word) for word in tokenized_sentence]
    bag=np.zeros(len(words),dtype=np.float32)
    for idx,w in enumerate(words):
        if w in sentence_word:
            bag[idx]=1
    return bag
```

fig 4.4

This code defines functions for tokenization (**tokenize**), stemming (**stem**), and creating a bag of words (**bag_of_words**). Tokenization splits a sentence into individual words. Stemming reduces words to their root form. The bag of words function creates a numerical representation of a sentence based on the presence or absence of words in a predefined set (**words**). The resulting bag is a binary vector indicating word occurrences. The code utilizes the NLTK library and a custom NLTK data directory.

File 5 – intents.json

The JSON structure represents a set of intents for a desktop voice assistant named "Echo." Each intent has a tag, patterns (user input phrases), and corresponding responses. The intents cover a variety of user interactions, including greetings, goodbyes, inquiries about health, commands for Google, WhatsApp, YouTube, playing, opening websites, and more. The assistant also handles tasks like telling jokes, providing information about itself, giving the date, time, and day, responding to compliments and insults, handling user queries about the developers, and offering functionalities like setting timers and asking for suggestions. The responses are designed to make the interaction engaging, informative, and user-friendly.

This json file is our chat dataset which we will convert into the pth file in the next code.

File 6 – Train.py

```
import numpy as np
import json
import torch
import torch.nn as nn
from torch.utils.data import Dataset,DataLoader
from neural_network import bag_of_words,tokenize,stem
from Mind import NeuralNet


with open('intents.json','r') as f:
    intents = json.load(f)
```

```
all_words = []
tags = []
xy= []

for intent in intents['intents']:
    tag=intent['tag']
    tags.append(tag)

    for pattern in intent["patterns"]:
        w=tokenize(pattern)
        all_words.extend(w)
        xy.append((w,tag))

ignore_words=[',','?','/','.','!']
all_words=[stem(w) for w in all_words if w not in ignore_words]
all_words=sorted(set(all_words))
tags=sorted(set(tags))

x_train=[]
y_train=[]

for (pattern_sentence,tag) in xy:
    bag=bag_of_words(pattern_sentence,all_words)
    x_train.append(bag)

    label=tags.index(tag)
    y_train.append(label)

x_train=np.array(x_train)
y_train=np.array(y_train)

num_epochs=1000
batch_size=8
learning_rate=0.001
input_size=len(x_train[0])
hidden_size=8
output_size=len(tags)

print("Training The Model...")

class chatDataset(Dataset):

    def __init__(self):
        self.n_samples=len(x_train)
        self.x_data=x_train
        self.y_data=y_train
    def __getitem__(self,index):
        return self.x_data[index],self.y_data[index]
    def __len__(self):
        return self.n_samples
dataset=chatDataset()
```

```
train_loader=DataLoader(dataset=dataset,
                        batch_size=batch_size,
                        shuffle=True,
                        num_workers=0)
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model=NeuralNet(input_size,hidden_size,output_size).to(device=device)
criterion=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=learning_rate)

for epoch in range(num_epochs):
    for (words,labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)
        outputs=model(words)
        loss=criterion(outputs,labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    if (epoch+1) % 100==0:
        print(f'Epoch [{epoch+1}/{num_epochs}],Loss: {loss.item():.4f}')
print(f'Final Loss:{loss.item():.4f}')

data={
    "model_state":model.state_dict(),
    "input_size":input_size,
    "hidden_size":hidden_size,
    "output_size":output_size,
    "all_words":all_words,
    "tags":tags
}
FILE="TrainData.pth"
torch.save(data,FILE)
print(f"Training Complete, File Saved To {FILE}")
```

fig 4.5

This python code reads intents from a JSON file, preprocesses the data by tokenizing, stemming, and encoding patterns and tags. It then sets up a neural network using PyTorch, defining a custom dataset and DataLoader. The model is trained with CrossEntropyLoss and Adam optimizer, iterating for a specified number of epochs. The training process involves updating weights based on the calculated loss. After training, the script saves the model's state dictionary, input size, hidden size, output size, and vocabulary to a file. This file, named 'TrainData.pth', can be used to deploy the trained chatbot model for interactive user interactions.

File 7 – Task.py

```
import datetime
from pytube import YouTube
from speaking import say
from keyboard import press,press_and_release,write
import Hearing
import wikipedia
import pywhatkit
import webbrowser as web
from os import startfile
from pyautogui import click
from time import sleep
def Time():
    time=datetime.datetime.now().strftime("%H:%M")
    say(time)

def Date():
    date=datetime.date.today()
    say(date)

def Day():
    day=datetime.datetime.now().strftime("%A")
    say(day)


def NonInputExecution(query):
    query=str(query)

    if "time" in query:
        Time()
    elif "date" in query:
        Date()
    elif "day" in query:
        Day()
def InputExecution(tag,query):

    if "wikipedia" in tag:
        name=str(query).replace("wikipedia","").replace("wiki","")
        result=wikipedia.summary(name)
        say(result)
    elif "Google" in tag:
         name=str(query).replace("Google","").replace("Google search","").replace("google
search","").replace("google","")
         query=query.replace("search","")
         query=query.replace("google","")
         pywhatkit.search(query)
    elif "youtube" in tag:
        query=query.replace("youtube","").replace("search","").replace("open","")
        result="https://www.youtube.com/results?search_query="+query
        web.open(result)
```

```
    elif "play" in tag:
        query=query.replace("play","")
        url=pywhatkit.playonyt(query)
        say("Welcome To youtube prompt")
        while True:
            say("Listening")
            query=Hearing.Listen()
            if  'pause' in query:
                press('k')
            elif  'play' in query:
                press('k')
            elif 'mute' in query:
                press('m')
            elif 'unmute' in query:
                press('m')
            elif 'increse volume' in query:
                press('ArrowUp')
            elif 'decrease volume' in query:
                press('ArrowDown')
            elif 'full screen' in query:
                press('f')
            elif 'exit full screen' in query:
                press('Esc')
            elif 'seek forward' in query:
                press('l')
            elif 'seek backward' in query:
                press('j')
            elif 'fast forward' in query:
                press('L')  # Shift + 'l'
            elif 'rewind' in query:
                press('J')  # Shift + 'j'
            elif 'next' in query:
                press('Shift + n')
            #elif 'previous' in query:
             #   press('Shift + p')
            elif 'increase playback speed' in query:
                press_and_release('Shift + .')  # Greater than (>)
            elif 'decrease playback speed' in query:
                press_and_release('Shift +k ,')  # Less than (<)
            elif 'go to beginning' in query:
                press('Home')
            elif 'go to end' in query:
                press('End')
            elif 'toggle theater mode' in query:
                press('t')
            elif 'toggle miniplayer' in query:
                press('i')
            elif 'toggle autoplay' in query:
                press('Shift + a')
            #elif 'like video' in query:
                press('Shift + l')
            #elif 'dislike video' in query:
```

```
                press('Shift + d')
            elif 'toggle captions' in query:
                press('c')
            elif 'cycle through captions' in query:
                press('Shift + c')
            elif 'move forward by frame' in query:
                press('.')
            elif 'download' in query:
                yt = YouTube(url)
                video_stream = yt.streams.get_highest_resolution()
                say(f"Downloading: {yt.title}")
                say(f"Resolution: {video_stream.resolution}")
                say(f"File Size: {video_stream.filesize / (1024 * 1024):.2f} MB")
                video_stream.download('D:\\Dekstop Assistant using
ML,NN,NLP,DL\\yt_download')
                say("Download complete!")

            elif 'exit' in query:
                say("Youtube Propt Closes.")
                break
    elif "website" in tag:
        query=query.replace("open","")
        web2=query.replace("website","")
        web2=web2.strip()
        web1='https://www.'+web2+'.com'
        web.open(web1)
    elif "whatsapp" in tag:
        while True:
            click(x=1340, y=1050)
            sleep(10)
            click(x=399, y=148)
            sleep(2)
            say("Tell me the name of the person with whom you wanna interact")
            query=Hearing.Listen()
            write(query)
            click(x=381,y=231)
            sleep(2)
            say("Say message ,voice call or video call to perform messaging , voice call
and video call respectively or say quit/exit to close whatsapp promt")
            res=Hearing.Listen()
            def whatsappMsg():
                click(x=768, y=980)
                sleep(2)
                say("Tell me what you wanna message?")
                query=Hearing.Listen()
                write(query)
                press("enter")
            def whatsappVoiceCall():
                click(x=381,y=231)
                sleep(2)
                click(x=1811,y=92)
                say("Calling.....")
```

```
            def whatsappVideocall():
                click(x=381,y=231)
                sleep(2)
                click(x=1735,y=92)
                say("Calling......")
            if "message" in res or "msg" in res:
                whatsappMsg()
            elif "voice call" in res or "voice" in res or "voicecall" in res:
                whatsappVoiceCall()
            elif "video call" in res or "video" in res or "videocall" in res:
                whatsappVideocall()
            elif "exit" in res or "quit" in res:
                break
            else:
                say("Can not understand you completely")
```

fig 4.6

This script contains functions for various voice-activated commands. It uses external libraries like pytube, pyttsx3, keyboard, and pywhatkit. The functions include retrieving and speaking the current time, date, and day, searching and summarizing information from Wikipedia, executing Google searches, opening YouTube videos, controlling YouTube playback, opening websites, and interacting with WhatsApp by sending messages or making calls. The script provides hands-free control over several applications and web services, enhancing user convenience through voice commands.

File 8 – Echo.py

This is our main file , after running Train.py file which will convert our json file into pth file and train our neural network upto 10 epochs , we run this file to finally start our “Echo”.

```
import random
import json
import torch
from Mind import NeuralNet
from neural_network import bag_of_words,tokenize
from Task import NonInputExecution
from Task import InputExecution
import os

device =torch.device('cuda' if torch.cuda.is_available() else 'cpu')
with open("intents.json",'r') as json_data:
    intents=json.load(json_data)
FILE="TrainData.pth"
data=torch.load(FILE)

input_size=data["input_size"]
hidden_size=data["hidden_size"]
output_size=data["output_size"]
```

```
all_words=data["all_words"]
tags=data["tags"]
model_state=data["model_state"]

model=NeuralNet(input_size,hidden_size,output_size).to(device)
model.load_state_dict(model_state)
model.eval()


Name = "Echo"
from speaking import say
from Hearing import Listen
def Main():
    sentence = Listen()
    result=str(sentence)
    if sentence in ["bye","goodbye","good bye","exit","close","terminate"]:
        exit()
    sentence=tokenize(sentence)
    X=bag_of_words(sentence,all_words)
    X=X.reshape(1,X.shape[0])
    X=torch.from_numpy(X).to(device)

    output=model(X)
    _ , predicted=torch.max(output,dim=1)

    tag=tags[predicted.item()]

    probs=torch.softmax(output,dim=1)
    prob=probs[0][predicted.item()]

    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag==intent["tag"]:
                reply=random.choice(intent["responses"])
                if "time" in reply:
                    NonInputExecution(reply)
                elif "date" in reply:
                    NonInputExecution(reply)
                elif "day" in reply:
                    NonInputExecution(reply)
                elif "wikipedia" in reply:
                    InputExecution(reply,sentence)
                elif "Google" in reply:
                    InputExecution(reply,result)
                elif "youtube" in reply:
                    InputExecution(reply,result)
                elif "playing" in reply:
                    InputExecution(reply,result)
                elif "website" in reply:
                    InputExecution(reply,result)
                elif "whatsapp" in reply:
                    InputExecution(reply,result)
```

```
                else:
                    say(reply)

while True:
    Main()
```

fig 4.7

This Python Code implements a voice-activated assistant using a trained neural network for intent recognition. It loads a pre-trained model and processes user input to determine the intent. If the confidence of the prediction is high, it executes the corresponding action, such as providing information, searching Wikipedia, or interacting with specific applications like Google, YouTube, or WhatsApp. The assistant continues to listen for user input in a loop, allowing for continuous interaction. The script enhances user experience by enabling voice-controlled commands for various tasks.

Output

On running Train.py file:

```
PS D:\Dekstop Assistant using ML,NN,NLP,DL> python -u "d:\Dekstop Assistant using ML,NN,NLP,DL\Train.py"
Training The Model...
Epoch [100/1000],Loss: 0.0551
Epoch [200/1000],Loss: 1.2102
Epoch [300/1000],Loss: 0.0103
Epoch [400/1000],Loss: 0.0003
Epoch [500/1000],Loss: 0.0001
Epoch [600/1000],Loss: 0.0010
Epoch [700/1000],Loss: 0.0001
Epoch [800/1000],Loss: 0.0000
Epoch [900/1000],Loss: 0.0000
Epoch [1000/1000],Loss: 0.0000
Final Loss:0.0000
Training Complete, File Saved To TrainData.pth
PS D:\Dekstop Assistant using ML,NN,NLP,DL>
```

fig 4.8

On running Echo.py file , you get this kind of responses where you need to give input through your voice.

```
Listening....
Recognizing...
You Said:hello echo

i am your assistant here to help.

Listening....
Recognizing...
You Said:tarikh kya hai

2023-11-23

Listening....
Recognizing...
You Said:time batao

04:49
```

fig 4.9

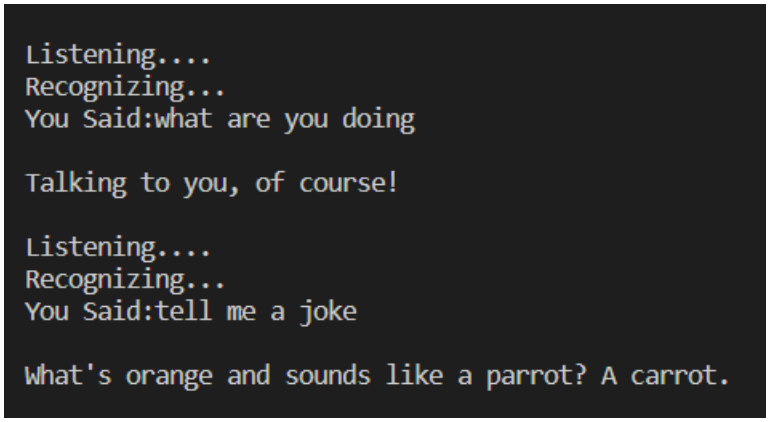

fig. 4.10

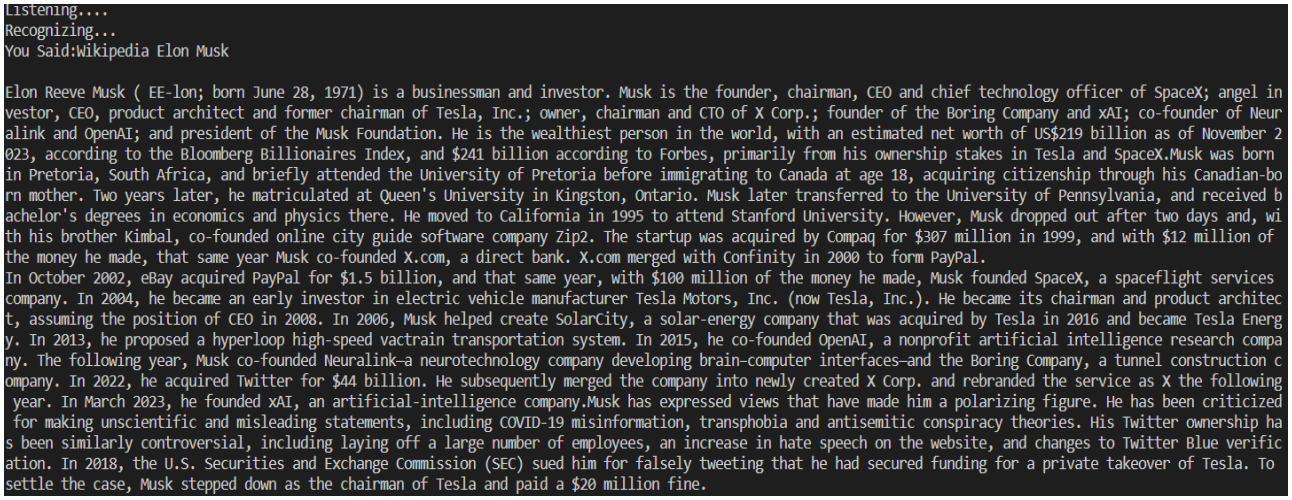

fig 4.11

we can chat with the “Echo” similarly and ask it to perform a variety of tasks.

Chapter 5: Result and Conclusion

The voice-activated assistant project demonstrates successful implementation of intent recognition using a neural network, providing a foundation for natural language understanding. The training process involves preprocessing textual data from **intents.json**, tokenizing and creating a bag-of-words representation. The **NeuralNet** class, trained over epochs, achieves high accuracy in recognizing user intents.

The system integrates seamlessly with external libraries like PyTorch for deep learning, Pytube for YouTube functionality, and Pywhatkit for Google searches. The user interacts with the assistant through voice commands, creating a hands-free and user-friendly experience. The assistant understands a range of commands, from basic queries about time, date, and day to more complex tasks such as searching Wikipedia, playing YouTube videos, and interacting with WhatsApp.

The modular architecture enhances maintainability and scalability. Each functionality, whether related to neural network training, task execution, or external services like YouTube and WhatsApp, resides in separate modules, promoting code organization and reusability. The voice assistant achieves a balance between simplicity and sophistication, making it accessible to users with varying technical backgrounds.

Conclusion:

In conclusion, the voice-activated assistant project offers an interactive and versatile user experience. The successful implementation of intent recognition enables users to perform tasks effortlessly through natural language commands. The utilization of machine learning techniques, particularly neural networks, showcases the potential for voice-based interfaces in everyday applications.

The system's ability to execute non-input tasks and respond to user queries demonstrates the practicality of the project. The integration of external services like YouTube and WhatsApp adds a layer of entertainment and communication, expanding the scope of use beyond information retrieval. The project has reached a stage where it can serve as a proof of concept for voice-controlled assistants in diverse environments.

However, it is important to acknowledge the limitations. The system's performance heavily relies on the quality and diversity of the training data. Enhancements in data preprocessing techniques and the inclusion of more intents can further improve accuracy and broaden the assistant's capabilities. Additionally, fine-tuning hyperparameters and exploring advanced neural network architectures could contribute to better generalization and robustness.

Chapter 6 - Future Scope:

The project has immense potential for future development and expansion. Here are some key areas for improvement and extension:

I. **Enhanced Intent Recognition:**
   a. Incorporate more diverse intents and responses to improve the assistant's understanding of user queries.
   b. Implement sentiment analysis to capture user emotions and respond accordingly.

II. **Continuous Learning:**
   a. Integrate mechanisms for continuous learning, allowing the assistant to adapt to new phrases, terms, and user preferences over time.

III. **Multimodal Interaction:**
   a. Extend capabilities to understand and respond to visual inputs, such as recognizing objects through a camera.

IV. **Advanced Task Execution:**
   a. Expand the range of tasks, including integration with more third-party services, automation of common computer tasks, and support for additional languages.

V. **User Personalization:**
   a. Implement user profiles to personalize responses and provide a tailored experience based on individual preferences.

VI. **Enhanced User Feedback:**
   a. Develop a feedback system to collect user input on the accuracy and helpfulness of responses, enabling continuous improvement.

VII. **Cross-Platform Integration:**
   a. Extend compatibility to work seamlessly across different devices and platforms, such as smartphones, smart speakers, and computers.

VIII. **Security and Privacy:**
   a. Implement robust security measures to protect user data and privacy, especially when dealing with sensitive tasks and information.

IX. Generative AI:
   i. We can implement this project with generative AI’s like ChatGpt and Google Bard e.t.c.

References:

[1]. Harshit Agrawal, Nivedita Singh, Gaurav Kumar, Dr. Diwakar Yagyasen, Mr. Surya Vikram Singh. "Voice Assistant Using Python" An International Open Access-revied, Refereed Journal.Unique Paper ID: 152099, Publication Volume & Issue: Volume 8, Issue 2, Page(s): 419-423.

[2]. George Terzopoulos, Maya Satratzemi "Voice Assistants and Smart Speakers in Everyday Life and In Education",Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece.

[3]. Deepak Shende. Ria Umabiya, Monika Raghorte, Aishwarya Bhisikar. Anup Bhange. "Al Based Voice Assistant Using Python", International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN 2349-5162, Vol.6,Issue 2, page no.506-509, February-2019.

[4]. Tulshan, Amrita & Dhage, Sudhir. (2019). "Survey on Virtual Assistant: Google Assistant, Siri, Cortana, Alexa", 4th International Symposium SIRS 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers. 10.1007/978-981-13-5758-9_17.

[5]. Dr. Kshama V. Kulhalli, Dr.Kotrappa Sirbi, Mr. Abhijit J. Patankar, "Personal Assistant with Voice Recognition Intelligence", International Journal of Engineering Research and Technology. ISSN 0974-3154 Volume 10, Number 1 (2017).

[6].https://www.researchgate.net/publication/372657833_DESKTOP'S_VIRTUAL_ASSISTANT_USING_PYTHON

[7]. https://www.geeksforgeeks.org/voice-assistant-using-python/

[8]. https://ieeexplore.ieee.org/document/9995997