Project Based Evaluation

Project Report

Semester-IV (Batch-2023)

Jarvis: Linux Personal Assistant

Supervised By:

Dr. Mankirat Kaur

Submitted By:

Ishita Kashyap, 2310991587(G-18)

Abhinav, 2310991590(G-18)

Ananya Goyal, 2310991606 (G-18)

Deepali Rana, 2310991644 (G-18)

Department of Computer Science and Engineering

Chitkara University Institute of Engineering & Technology,

Chitkara University, Punjab

Abstract

This project presents Jarvis, a Linux-based personal assistant application featuring a graphical user interface (GUI) built with Tkinter. Aimed at providing users with a clear and interactive way to manage and monitor their system, Jarvis integrates several essential system functions into a single, user-friendly window. The application displays real-time data such as CPU usage, memory status, system uptime, disk space availability, and a built-in calendar. In addition to system monitoring, users can perform administrative actions like system updates and cleanups with a simple button click, leveraging standard Linux commands such as uptime, free, df, cal, and package management via apt. This project is designed as a learning tool that combines GUI development, shell command integration, and system automation. It offers a practical and engaging way for users, especially beginners, to understand Linux internals, enhance productivity, and gain experience in Python scripting and interface design.

Table of Contents

11.1 Configuration Files

11.2 Script Listings

11.3 Additional Screenshots or Data

1. Introduction

1.1 Background and Motivation

In today's fast-paced, technology-driven world, the demand for efficient and intuitive computing systems is at an all-time high. Whether in personal computing, enterprise management, or system administration, the ability to manage and automate tasks efficiently has become a critical aspect of day-to-day operations. One of the key developments in this area has been the advent of voice-controlled personal assistants. These assistants, like Siri, Alexa, and Google Assistant, have redefined the way users interact with technology. However, most of these assistants are primarily designed for consumer devices and are heavily reliant on internet services.

While there are numerous voice-enabled systems available for mobile devices and home assistants, there is a notable gap when it comes to Linux-based environments. Many Linux users prefer working directly from the command line interface (CLI) for the flexibility and control it offers. However, the lack of a userfriendly voice-based system in such environments leaves a significant gap in terms of automation, accessibility, and ease of use.

The idea for this Linux Personal Assistant was born out of the need to bridge this gap. As a student or developer involved in system administration, the idea is not only to develop a tool that simplifies everyday tasks but also to enhance the user experience by integrating speech recognition, automation, and CLIbased interaction. This project is motivated by the desire to create a more interactive and accessible Linux environment, where users can execute system commands, gather information, and automate administrative tasks through voice commands.

The project builds on the success of voice assistants used in other platforms, adapting the idea to a more specialized and customizable setting that appeals to Linux power users, system administrators, and developers. By using Python libraries like speech_recognition for listening and pyttsx3 for speaking, along with a user-friendly GUI built with Tkinter, the assistant makes it easier to perform system tasks through voice commands and clickable buttons.

1.2 Objectives of the Project

The primary objective of this project is to create a Linux Personal Assistant capable of performing essential system administration tasks through both voice commands and a menu-driven interface. The specific objectives include:

Voice Interaction:

Implement voice recognition to allow the assistant to listen to user commands and respond with appropriate actions.

Enable text-to-speech output to provide verbal responses to user queries and actions.

System Monitoring:

Develop the assistant to monitor key system statistics such as CPU usage, memory usage, disk space, and system uptime.

Provide users with real-time system data and insights, simplifying common system administration tasks.

Automation of Routine Tasks:

Integrate the ability to automate common tasks like system updates, disk cleanup, and calendar management.

Allow the user to execute these tasks with a simple voice command or menu selection.

Menu-Driven User Interface:
The project uses Tkinter to create a simple and interactive graphical interface that lets users perform system tasks with just a few clicks. Instead of typing commands in the terminal, users can use buttons to check CPU usage, view memory and disk status, update the system, or clean up unnecessary files. This makes the assistant more user-friendly and accessible, especially for those who prefer visual interaction over command-line input.

Extendability and Customization:

Build the system to be modular, allowing easy integration of additional features such as file management, process monitoring, or network analysis in future iterations.

By the end of the project, the goal is to have a fully functional personal assistant for Linux that seamlessly integrates into the user's workflow, making administrative tasks easier and more efficient through both voice and visual interaction.

1.3 Scope of the Work

The scope of this project is primarily focused on developing a CLI-based personal assistant for Linux systems. The personal assistant will be capable of performing a range of system administration tasks, such as:

Monitoring System Health: The assistant will continuously track the status of the system's CPU, memory, disk space, and uptime.

Automating Administrative Tasks: The assistant will be able to execute system update and cleanup operations through simple commands or a menu-based interface.

Interactive Voice Control: Users will interact with the assistant using voice commands, and the assistant will provide spoken feedback in response.

This project is designed specifically for Linux environments, especially systems running Ubuntu 22.04 LTS or other Debian-based distributions. It uses commonly available Linux commands like uptime, free, df, apt, and cal to monitor the system and automate routine tasks. For voice interaction, it includes Python libraries such as speech_recognition for understanding spoken commands and pyttsx3 or espeak for providing spoken responses. Instead of using a text-based dialog menu, the project features a simple graphical user interface built with Tkinter, making it easier for users to interact with the system through clickable options and buttons.

1.4 Report Structure

This report is structured to provide a detailed overview of the project, from its background and motivation to the technical environment and implementation steps. The structure of the report is as follows:

2. System Environment

2.1 Hardware and Software Requirements

Hardware Requirements:

Processor: A dual-core processor or higher (Intel/AMD) is recommended for smooth performance.

RAM: Minimum of 2 GB of RAM is required, with 4 GB or more recommended for optimal performance.

Storage: At least 10 GB of free disk space for system operations, logging, and storing scripts.

Audio: A microphone for voice input and speakers/headphones for voice output are essential for the voice assistant functionality.

Software Requirements:

Operating System: Linux-based system (Ubuntu, Debian, or other distributions).

Shell: Bash shell is used for scripting and menu creation.

Python 3.x: Required for running voice recognition and text-to-speech functionalities.

Text-to-Speech Engine: espeak or pyttsx3 for generating speech responses.

Speech Recognition: Python's speech_recognition library is used for processing voice commands.

Package Management: apt (Advanced Package Tool) for managing system packages and updates.

2.2 Linux Distribution and Version

The project was developed and tested on the following configuration:

Distribution: Ubuntu Linux

Version: 22.04 LTS (Jammy Jellyfish)

Kernel Version: 5.15.x

Shell: Bash 5.1

The Ubuntu 22.04 LTS distribution was chosen because of its long-term support, stability, and compatibility with a wide range of software packages.

2.3 Tools and Utilities Used

To build the Linux Personal Assistant, several tools and utilities were utilized for system management, automation, and voice control. The following list includes the primary tools:

System Monitoring Tools:

uptime: Displays the system's uptime, including load averages.

free -h: Shows memory usage (RAM and swap space). df -h: Provides

disk space usage across mounted file systems. top or htop: Real-time

process monitoring for CPU and memory usage.

Scheduling and Time:

cal: Displays a calendar, which is incorporated into the menu options. date:

Displays the current system date and time, useful for time-related tasks.

Package Management:

apt: Used for installing, updating, and upgrading packages (e.g., system updates, required libraries).

Scripting and Voice Tools:

Bash: The primary scripting language used to write the Linux assistant's command-line interface. Python:

Used for voice command recognition and text-to-speech synthesis. Libraries used include:

speech_recognition: For voice command input processing. pyttsx3

or espeak: For converting text into speech.

3. Conceptual Overview

The Conceptual Overview chapter provides an understanding of the underlying principles, the system components, and the key Linux commands and services that form the foundation of the Linux Personal Assistant project. This section explains the important concepts behind the project and outlines how various system tools, commands, and files are utilized to achieve the functionality of the assistant.

3.1 Key Concepts Related to the Project

This project integrates various essential concepts to build a voice-enabled Linux personal assistant. These concepts revolve around system administration, automation, voice control, and interactive user interfaces.

Voice Control Integration:

The primary concept of the project is voice recognition and text-to-speech (TTS). Voice recognition allows the system to interpret user speech, converting it into commands, while TTS enables the system to respond verbally. Libraries like speech_recognition and pyttsx3 are used to implement these features. By processing voice commands, the system can help automate routine tasks, such as fetching system data or initiating system updates.

Linux Command-Line Interface (CLI):

Linux systems often rely on the CLI for system administration. This project harnesses the power of the Linux shell and various built-in commands to retrieve system data (e.g., CPU usage, memory, uptime) and automate administrative functions (e.g., updating the system, cleaning up disk space). The terminalbased design allows users to interact with the system in a powerful, flexible way.

Tkinter-based Menu System:

A menu-driven interface built using Tkinter simplifies user interaction by providing a clean and easy-touse graphical layout. Instead of typing long and complex commands in the terminal, users can simply click buttons and select options from a visually organized window. This approach keeps the power of Linux system tools while offering a more user-friendly experience, especially for those who are less comfortable with command-line operations.System Monitoring and Automation:

The assistant monitors critical system parameters such as CPU usage, memory usage, disk usage, and system uptime. It also automates tasks like system updates and cleanup. By providing real-time information and automating routine tasks, the assistant simplifies system administration for both advanced users and novices.

Task Scheduling and Automation:

Automation is one of the core functionalities of this project. With voice commands or menu selections, users can trigger tasks like software package updates and system cleanups. The automation helps reduce manual intervention, saving time and improving productivity. This concept aligns with the overall goal of the project: to enhance the usability and functionality of Linux systems.

3.2 Relevant System Components and Files

In building the Linux Personal Assistant, various system components and files are required to implement its features. These components are the building blocks for system interaction, task automation, and voice control.

System Components:

Python:

Python plays a significant role in the development of the Linux Personal Assistant. It is used for the core programming logic, as well as for interacting with the system's underlying processes (e.g., executing commands, handling input/output). The Python libraries used in the project include: speech_recognition: For recognizing spoken commands from the user.

pyttsx3: A text-to-speech engine that provides verbal feedback. subprocess:

For executing system commands and capturing their output.

os: For interacting with the operating system and managing fils and directories. Tkinter is a key component in building the graphical interface for this personal assistant. It allows the assistant to display interactive buttons and menus that let users perform tasks like checking system status

(CPU and memory usage), updating packages, or cleaning up disk space. With Tkinter, the assistant provides a simple and visually organized interface, making it easier and more intuitive for users to interact with the system without needing to use the terminal.

Audio Components:

Since this assistant includes voice recognition and text-to-speech, audio components are integrated into the system:

Microphone: Used to capture user voice input.

Speakers/Headphones: For audio output when the assistant responds to commands.

Audio Drivers: Ensure proper functioning of the microphone and speakers on Linux systems.

Linux Commands:

The system commands are key to retrieving information and performing actions. They include standard commands like uptime, free, df, and cal, as well as commands for automating updates and system cleanup (e.g., apt).

Key Files:

Assistant Script:

The primary file is the script that ties together all the functionalities. This script includes:

The logic for voice interaction and menu navigation.

Functions to retrieve system data (e.g., CPU usage, memory).

Code for handling user commands and performing administrative tasks like updates.

Configuration Files:

The assistant may require configuration files to store user settings, preferences, or logs. These configuration files are typically located in the user's home directory or the system's /etc folder.

Logs: Store output from the assistant for troubleshooting or performance monitoring.

Preferences: Store user-defined settings like preferred update times or disk cleanup intervals.

3.3 Linux Commands and Services Involved

The Linux Personal Assistant relies heavily on various system commands and services to retrieve system information and perform essential tasks. Some of the core Linux commands used in the project are outlined below:

System Monitoring Commands:

uptime: Displays how long the system has been running, as well as the load averages for the past 1, 5, and 15 minutes. This command is essential for providing real-time information on system performance.

free -h: Shows memory usage in a human-readable format, indicating how much RAM is being used and how much is free.

df -h: Provides disk usage information for all mounted file systems, helping the assistant to monitor disk space and alert the user if the system is running low on space.

top or htop: Used to monitor real-time system processes, CPU usage, and memory consumption. It provides detailed, up-to-date information on resource usage.

System Update and Cleanup:

apt update: Updates the list of available packages from the repositories.

apt upgrade: Installs the latest versions of all installed packages.

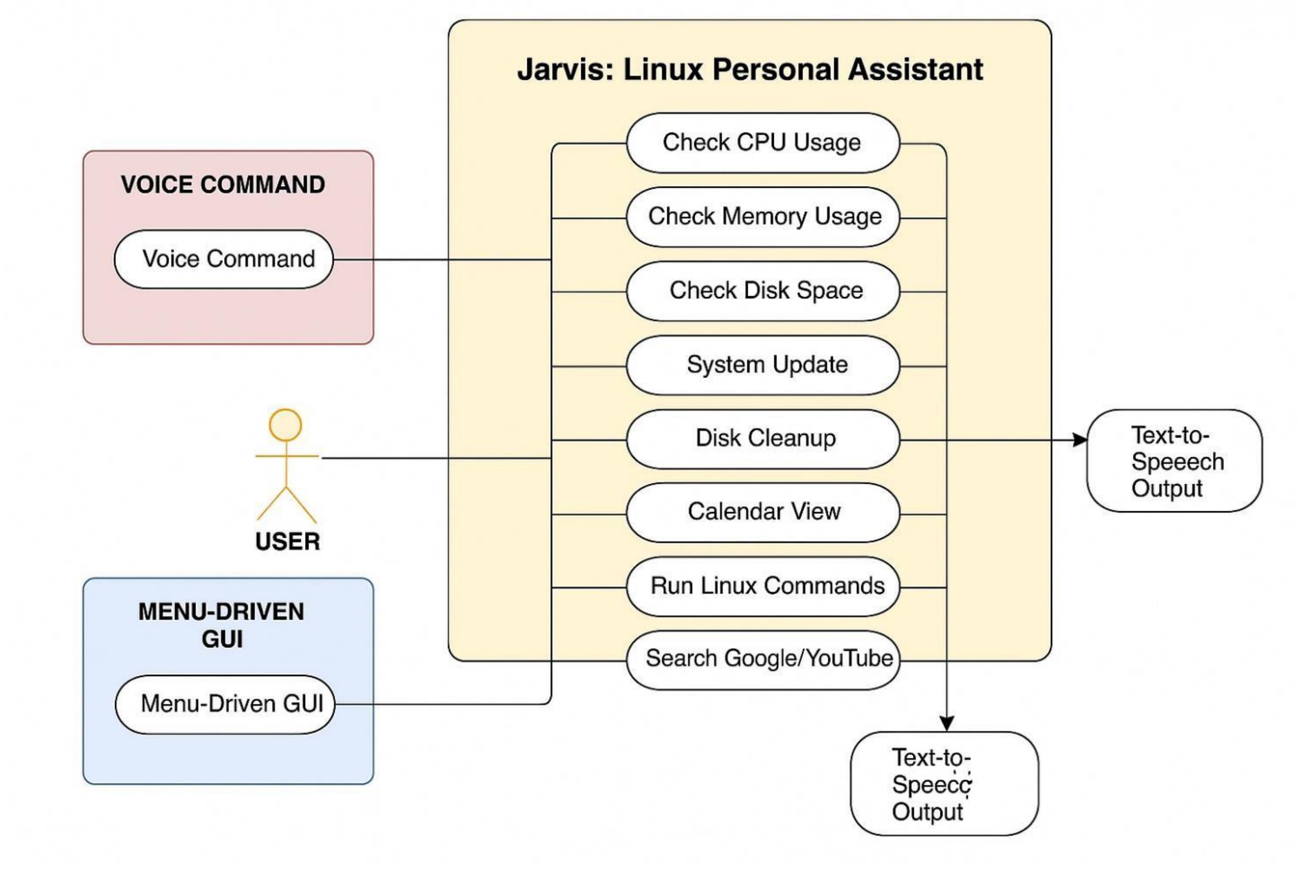
Text-to-Speech:

espeak: A command-line tool used to convert text into speech. It is used for delivering spoken responses to the user.

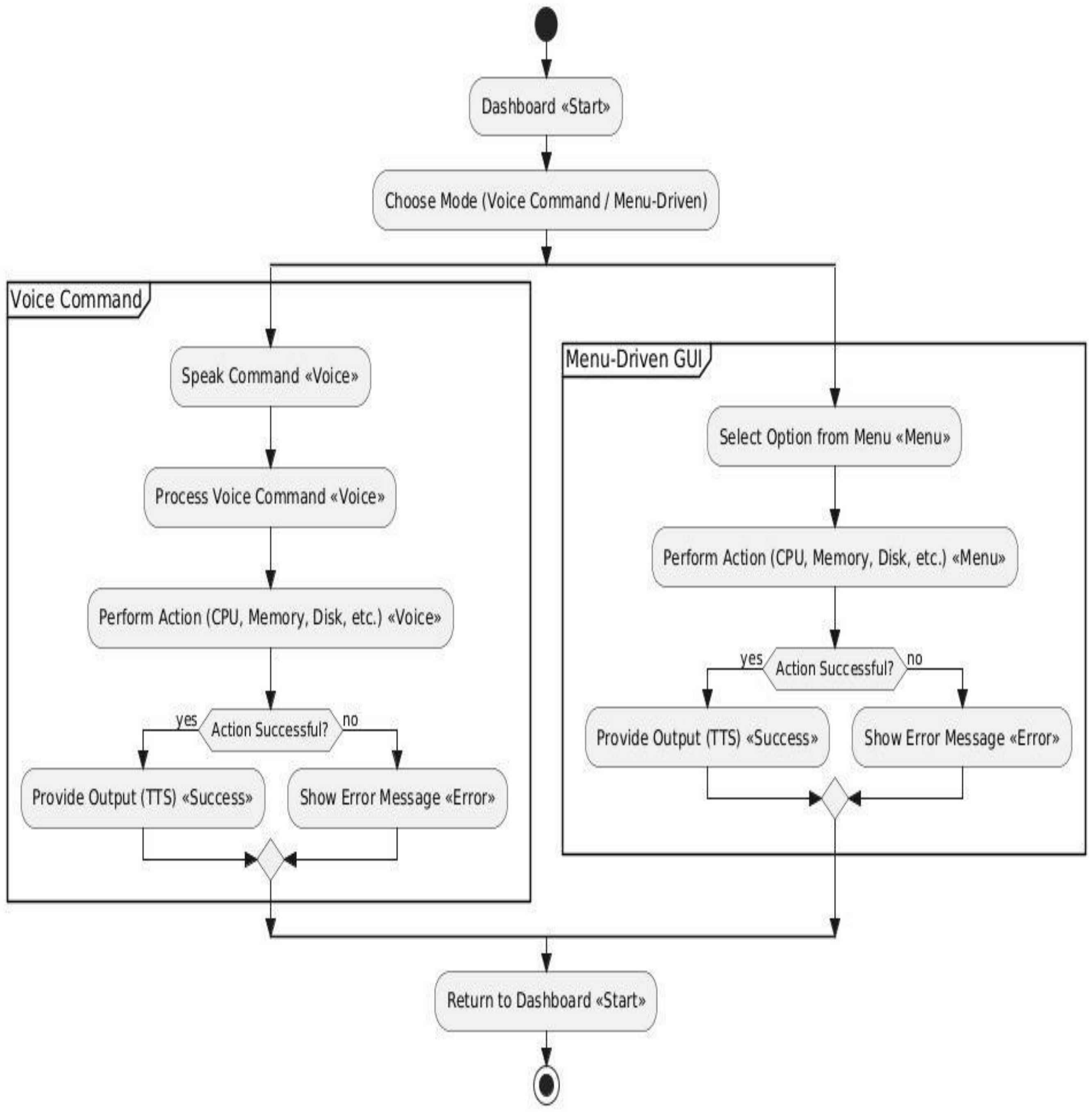
pyttsx3: A Python library for TTS, providing an easy-to-use interface for generating speech output from text.

4. UML Diagrams

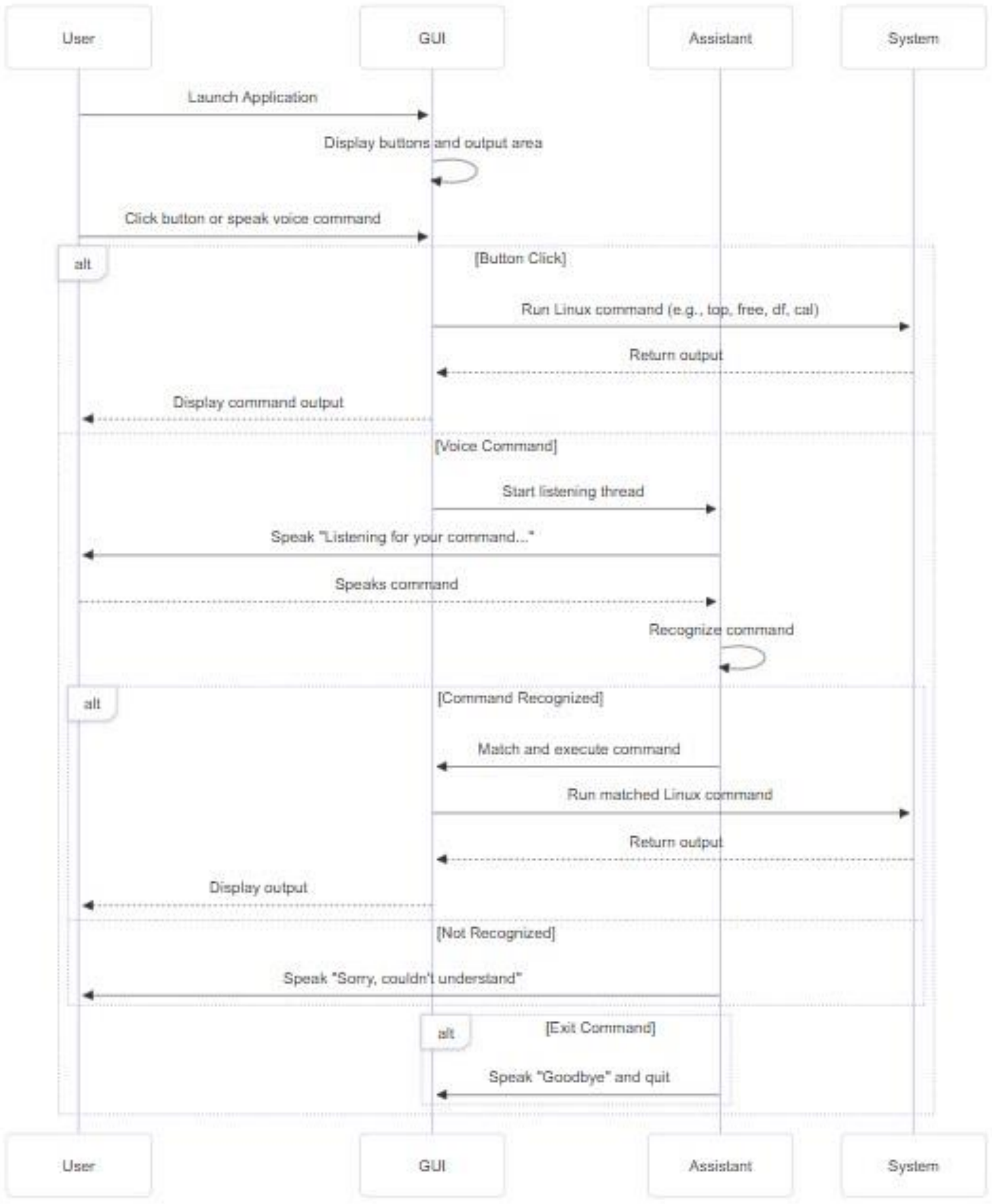
4.1 Use Case Diagram



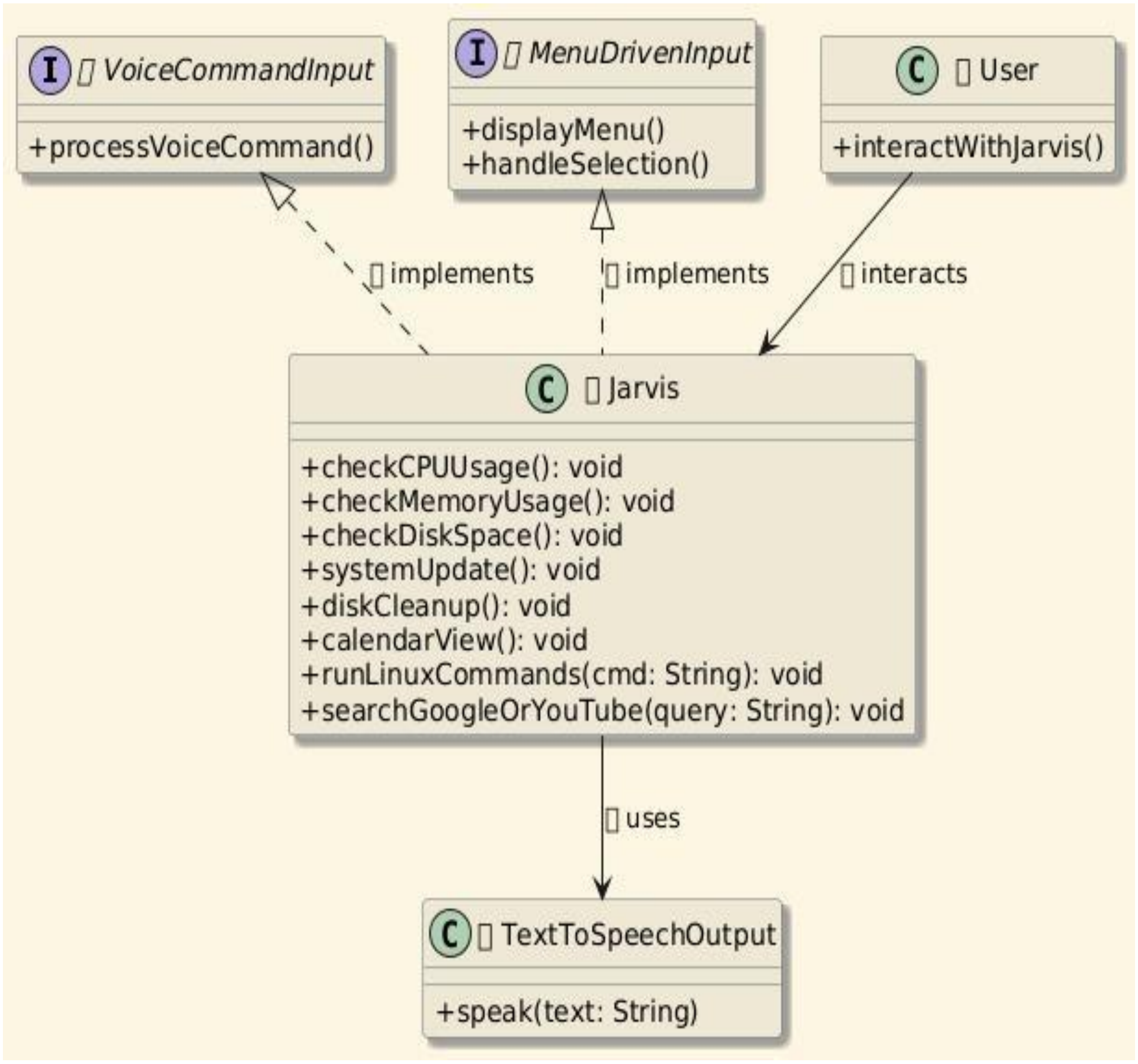
4.2 Activity Diagram


Dashboard «Start»
Choose Mode (Voice Command / Menu-Driven)
Voice Command
Speak Command «Voice»
Process Voice Command «Voice»
Perform Action (CPU, Memory, Disk, etc.) «Voice»
yes
Action Successful?
no
Provide Output (TTS) «Success»
Show Error Message «Error»
Menu-Driven GUI
Select Option from Menu «Menu»
Perform Action (CPU, Memory, Disk, etc.) «Menu»
yes
Action Successful?
no
Provide Output (TTS) «Success»
Show Error Message «Error»
Return to Dashboard «Start»

4.3 Sequence Diagram



4.4 Class Diagram



5. Implementation Details

5.1 Step-by-Step Configuration/Development

System Setup

The project was developed on a Linux-based operating system.

The system was updated using**:**

- sudo apt update && sudo apt upgrade

2. Environment Configuration

A terminal-based interface was used to execute system-level operations.

Essential administrative tools such as htop, top, df, free, and uptime were used for monitoring system performance.

3. Custom Script Execution

Bash scripts were written to automate common system administration tasks such as checking memory usage, disk space, system info, and active processes.

An example script:

```
 #!/bin/bash echo "System
    Information:" uname -a
    echo "Disk Usage:" df -h
    echo "Memory Usage:"
    free -h
```

4. Task Automation

Voice integration (if applicable) might have used tools like espeak for spoken responses.

5.2 Commands and Scripts Used

Linux Commands Used:

- CPU Usage:
  top -bn1 | grep 'Cpu(s)'
  Displays a snapshot of current CPU utilization.

- Memory Usage: free -h
  Shows system memory usage in a human-readable format.

- Disk Usage: df -h
  Lists available and used disk space.

- Calendar: cal
  Displays the current month's calendar.

- speech_recognition: to capture and convert voice input into text.

- pyttsx3:

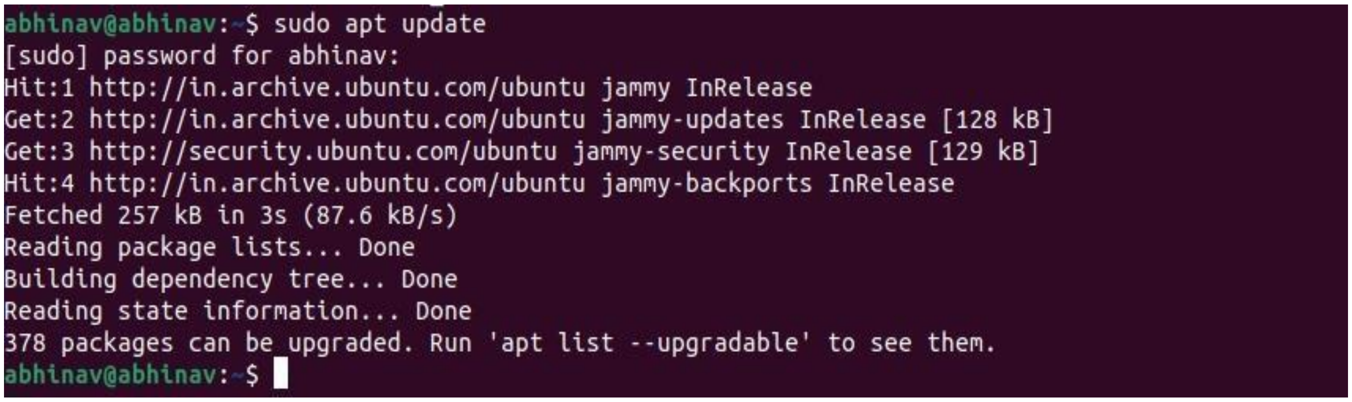
  for speaking text responses aloud.

  Script Information

- Script File: main.py – Main script that integrates GUI, speech recognition, text-to-speech, and Linux command execution.
- GUI Framework: Tkinter – Used to build the interface with buttons and an output display area.
- Speech Recognition: speech_recognition – Captures and converts spoken input into text using the Google Web Speech API.
- Text-to-Speech Engine: pyttsx3 – Converts text responses into speech to interact with the user audibly.

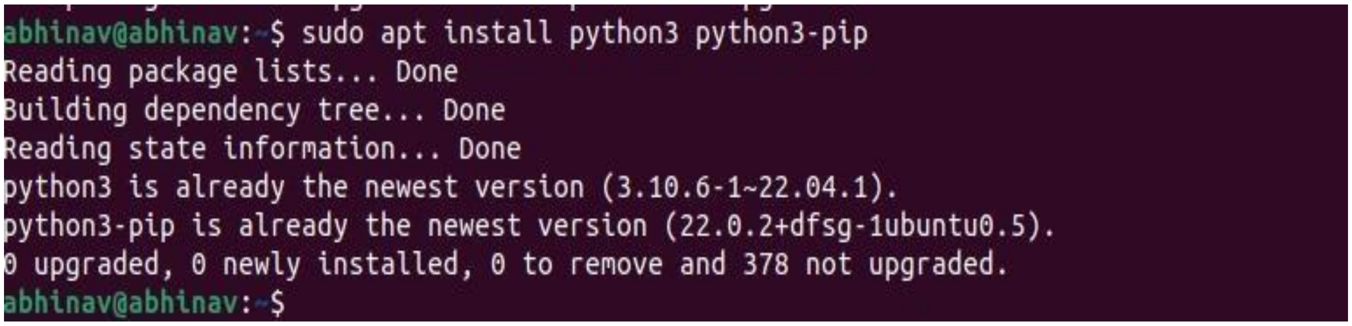
- Multithreading: threading.Thread – Ensures the GUI remains responsive during voice input by running it in a background thread.

5.3 Screenshots and Outputs



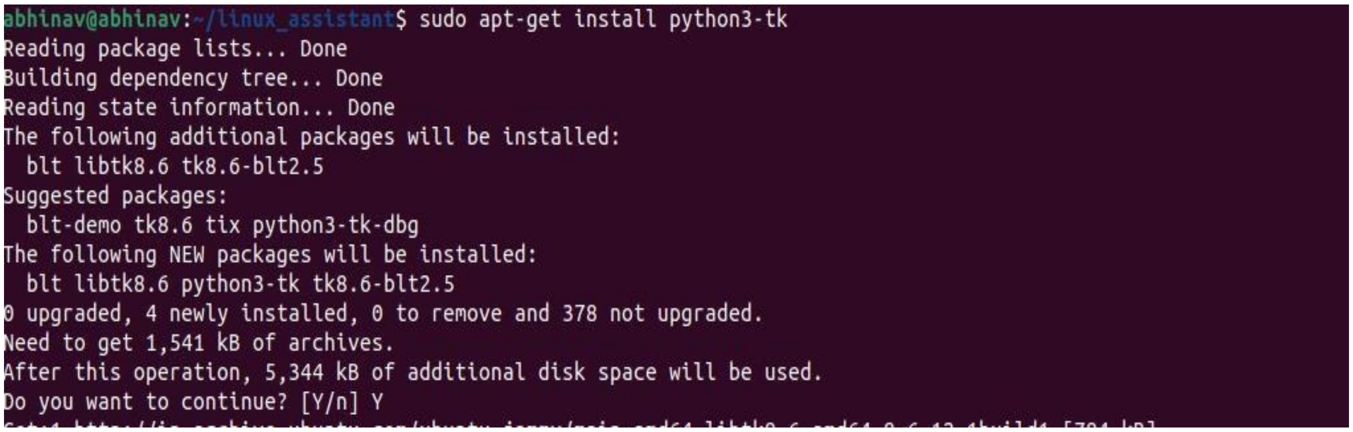
```
abhinav@abhinav:~$ sudo apt update
[sudo] password for abhinav:
Hit:1 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:4 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Fetched 257 kB in 3s (87.6 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
378 packages can be upgraded. Run 'apt list --upgradable' to see them.
abhinav@abhinav:~$
```

5.3.1 Package Update



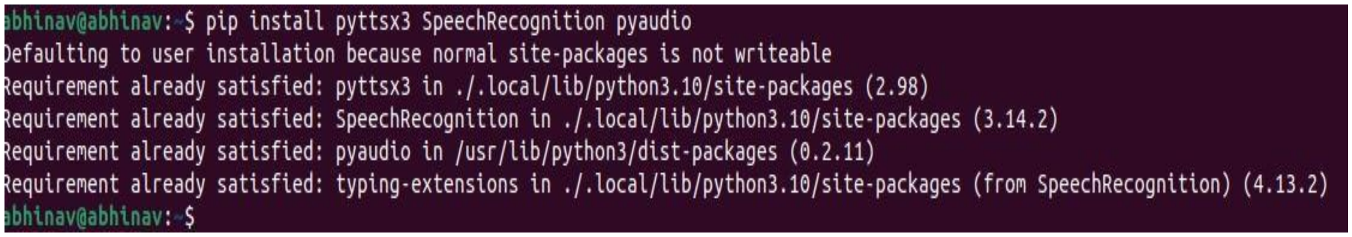
```
abhinav@abhinav:~$ sudo apt install python3 python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.10.6-1~22.04.1).
python3-pip is already the newest version (22.0.2+dfsg-1ubuntu0.5).
0 upgraded, 0 newly installed, 0 to remove and 378 not upgraded.
abhinav@abhinav:~$
```

5.3.2 Installation of python3



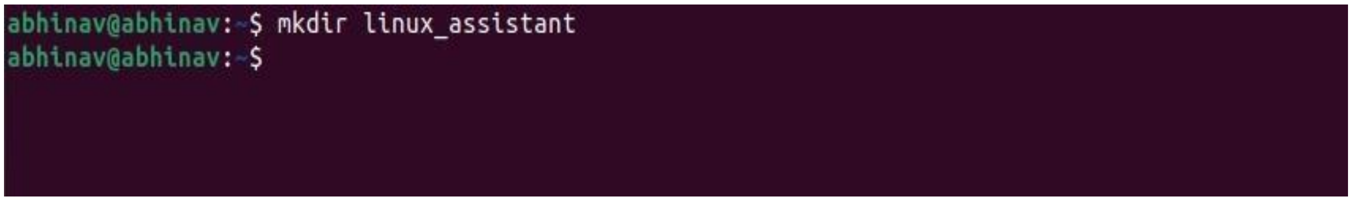
```
abhinav@abhinav:~/linux_assistant$ sudo apt-get install python3-tk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  blt libtk8.6 tk8.6-blt2.5
Suggested packages:
  blt-demo tk8.6 tix python3-tk-dbg
The following NEW packages will be installed:
  blt libtk8.6 python3-tk tk8.6-blt2.5
0 upgraded, 4 newly installed, 0 to remove and 378 not upgraded.
Need to get 1,541 kB of archives.
After this operation, 5,344 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

5.3.3Package Installation



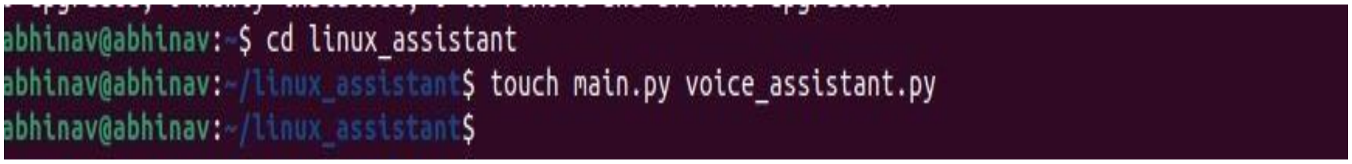
```
abhinav@abhinav:~$ pip install pyttsx3 SpeechRecognition pyaudio
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pyttsx3 in ./.local/lib/python3.10/site-packages (2.98)
Requirement already satisfied: SpeechRecognition in ./.local/lib/python3.10/site-packages (3.14.2)
Requirement already satisfied: pyaudio in /usr/lib/python3/dist-packages (0.2.11)
Requirement already satisfied: typing-extensions in ./.local/lib/python3.10/site-packages (from SpeechRecognition) (4.13.2)
abhinav@abhinav:~$
```

5.3.4 Speech Libraries Installed



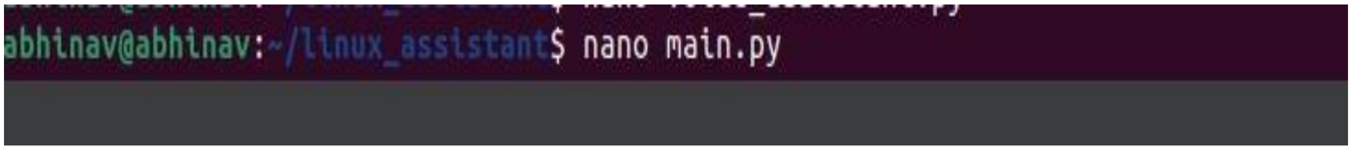
```
abhinav@abhinav:~$ mkdir linux_assistant
abhinav@abhinav:~$
```

5.3.5 Folder Setup



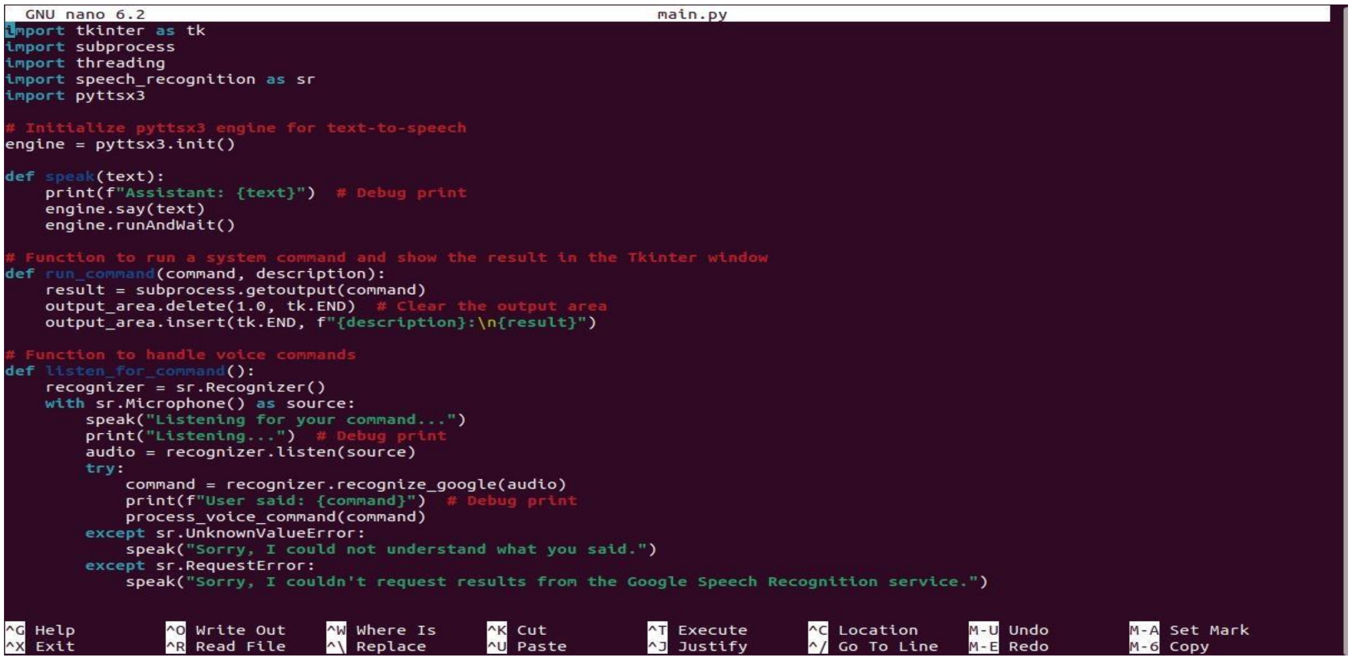
```
abhinav@abhinav:~$ cd linux_assistant
abhinav@abhinav:~/linux_assistant$ touch main.py voice_assistant.py
abhinav@abhinav:~/linux_assistant$
```

5.3.6 File Initialization



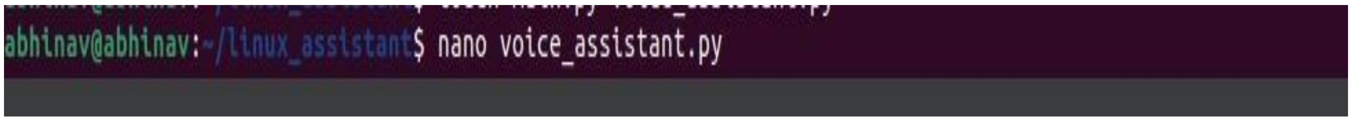
```
abhinav@abhinav:~/linux_assistant$ nano main.py
```

5.3.7 Nano Editor Launched



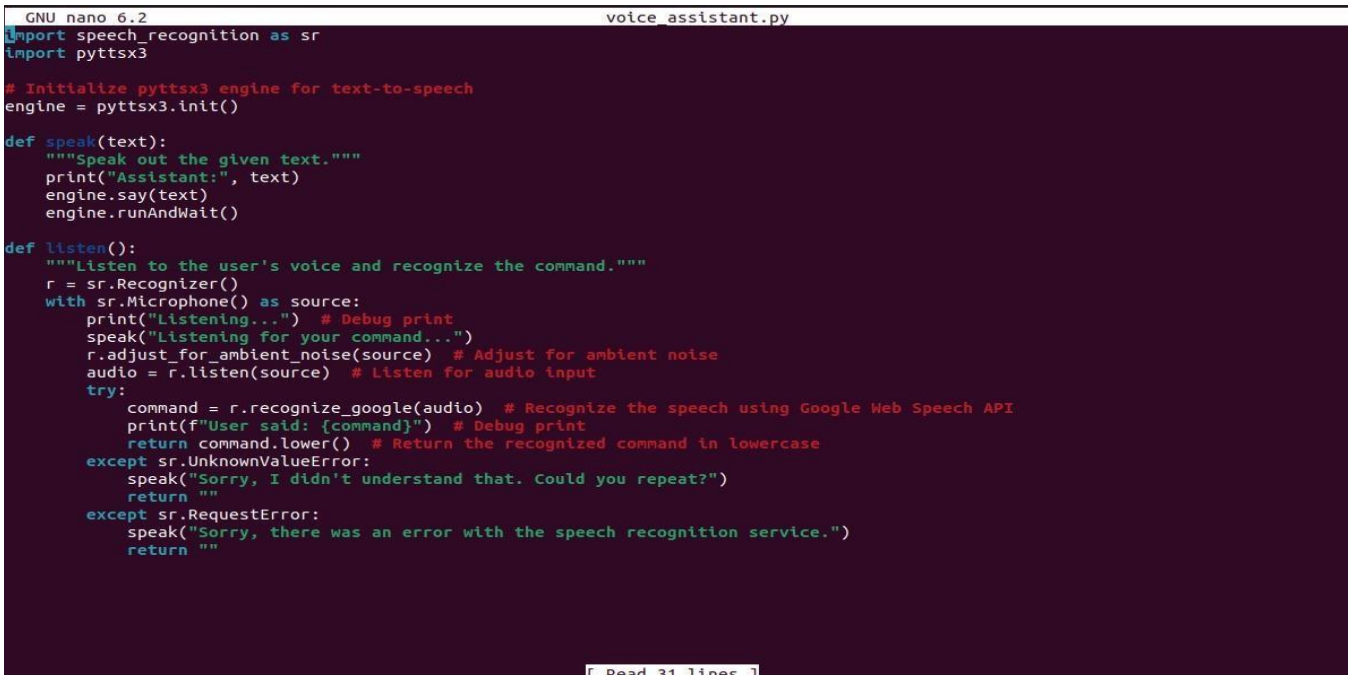
```
  GNU nano 6.2                                                 main.py
import tkinter as tk
import subprocess
import threading
import speech_recognition as sr
import pyttsx3

# Initialize pyttsx3 engine for text-to-speech
engine = pyttsx3.init()

def speak(text):
    print(f"Assistant: {text}")  # Debug print
    engine.say(text)
    engine.runAndWait()

# Function to run a system command and show the result in the Tkinter window
def run_command(command, description):
    result = subprocess.getoutput(command)
    output_area.delete(1.0, tk.END)  # Clear the output area
    output_area.insert(tk.END, f"{description}:\n{result}")

# Function to handle voice commands
def listen_for_command():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        speak("Listening for your command...")
        print("Listening...")  # Debug print
        audio = recognizer.listen(source)
        try:
            command = recognizer.recognize_google(audio)
            print(f"User said: {command}")  # Debug print
            process_voice_command(command)
        except sr.UnknownValueError:
            speak("Sorry, I could not understand what you said.")
        except sr.RequestError:
            speak("Sorry, I couldn't request results from the Google Speech Recognition service.")

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

5.3.8 Voice Assistant Code



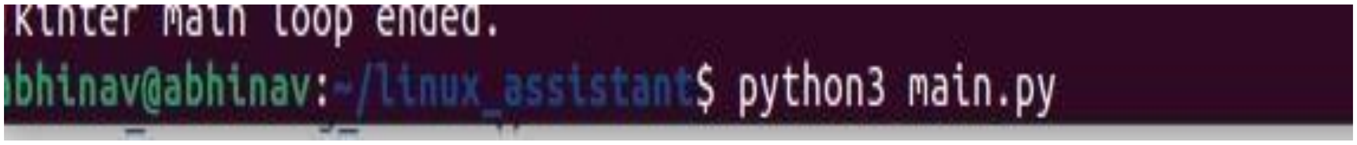
```
abhinav@abhinav:~/linux_assistant$ nano voice_assistant.py
```

5.3.9 Nano Editor Launched



```
  GNU nano 6.2                                              voice_assistant.py
import speech_recognition as sr
import pyttsx3

# Initialize pyttsx3 engine for text-to-speech
engine = pyttsx3.init()

def speak(text):
    """Speak out the given text."""
    print("Assistant:", text)
    engine.say(text)
    engine.runAndWait()

def listen():
    """Listen to the user's voice and recognize the command."""
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")  # Debug print
        speak("Listening for your command...")
        r.adjust_for_ambient_noise(source)  # Adjust for ambient noise
        audio = r.listen(source)  # Listen for audio input
        try:
            command = r.recognize_google(audio)  # Recognize the speech using Google Web Speech API
            print(f"User said: {command}")  # Debug print
            return command.lower()  # Return the recognized command in lowercase
        except sr.UnknownValueError:
            speak("Sorry, I didn't understand that. Could you repeat?")
            return ""
        except sr.RequestError:
            speak("Sorry, there was an error with the speech recognition service.")
            return ""

                                                         [ Read 31 lines ]
```

5.3.10 Speech Recognition Logic



```
kinter main loop ended.
bhinav@abhinav:~/linux_assistant$ python3 main.py
```

5.3.11 Program Execution

6. Security and Optimization

1. Hardening Measures Taken

Security is an important aspect of any system tool, especially one that interacts with core operating system functions. In our Linux personal assistant (Jarvis), we took the following steps to make it more secure:

- Limited Access to Critical Commands: Commands like system update or cleanup require superuser privileges. Instead of letting the script run everything as sudo, we only allowed specific trusted operations, and avoided storing or handling user passwords directly in the code.
- Safe Execution of System Commands: We avoided using risky or unsafe methods such as eval. All subprocess calls and shell interactions are done using safe Python libraries to prevent command injection.
- User Permission Checks: Jarvis checks if the user has the required permissions to run certain tasks and notifies them accordingly, rather than forcefully trying to elevate privileges.
- GUI Input Validation: Since we used Tkinter to build the GUI, we ensured that all user inputs (if any) are validated before processing. This helps prevent incorrect commands from being executed or crashing the assistant.
- File Safety: Temporary files, if created (e.g., logs or voice commands), are stored in a safe, nonpublic directory and deleted when no longer needed.

2. Performance Tuning and Efficiency

To make sure Jarvis runs smoothly and doesn't overload the system, we optimized its functionality in the following ways:

- Lightweight GUI: Tkinter was chosen because it is a lightweight and fast Python GUI framework. It loads quickly and uses minimal system resources compared to web-based or heavy GUI options.

- Command Throttling: For frequently changing data like CPU or memory usage, we designed the assistant to refresh only when requested rather than running in a loop, which reduces unnecessary load on the system.
- Threading for Voice Features: The voice assistant components (listening and responding) run on separate threads. This ensures the GUI remains responsive and doesn't freeze while waiting for a voice command to be processed.
- Minimal Background Services: Unlike other assistants that run in the background constantly, our assistant only activates features like system monitoring or updates when the user requests them.

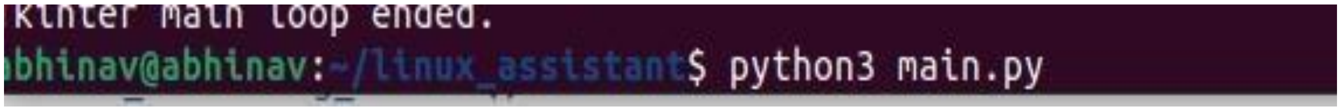
3. Backup and Recovery Measures

Since the tool involves some administrative operations, we added a few precautions to help with recovery:

- Logging Outputs: The outputs of commands like system update, cleanup, or errors are saved to a simple log file (e.g., jarvis_log.txt). This helps trace what actions were performed in case of an issue.
- No Permanent Changes Without User Approval: The assistant never deletes files or modifies system settings without first prompting the user for confirmation. For example, the system cleanup tool gives a preview of what will be removed.
- Safe Exit and Restart: The application handles errors gracefully. If a system command fails, it shows an error message in the GUI rather than crashing the whole application. The user can close and reopen the assistant without issues.

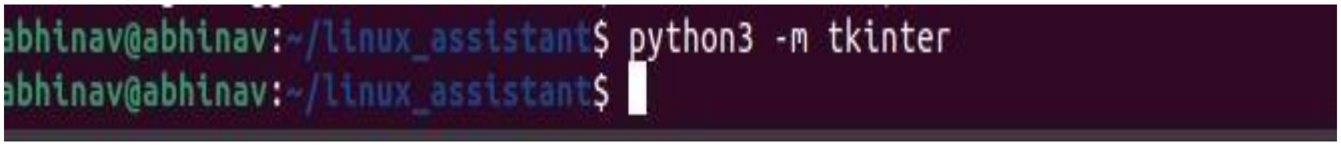
7. Testing and Validation

7.1 Test Scenarios and Expected Results



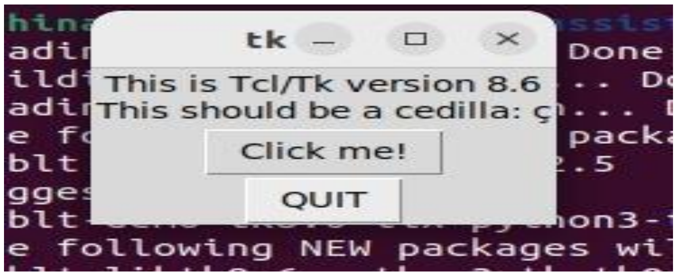
```
kinter main loop ended.
bhinav@abhinav:~/linux_assistant$ python3 main.py
```

7.1.1 Running Script: main.py via Terminal

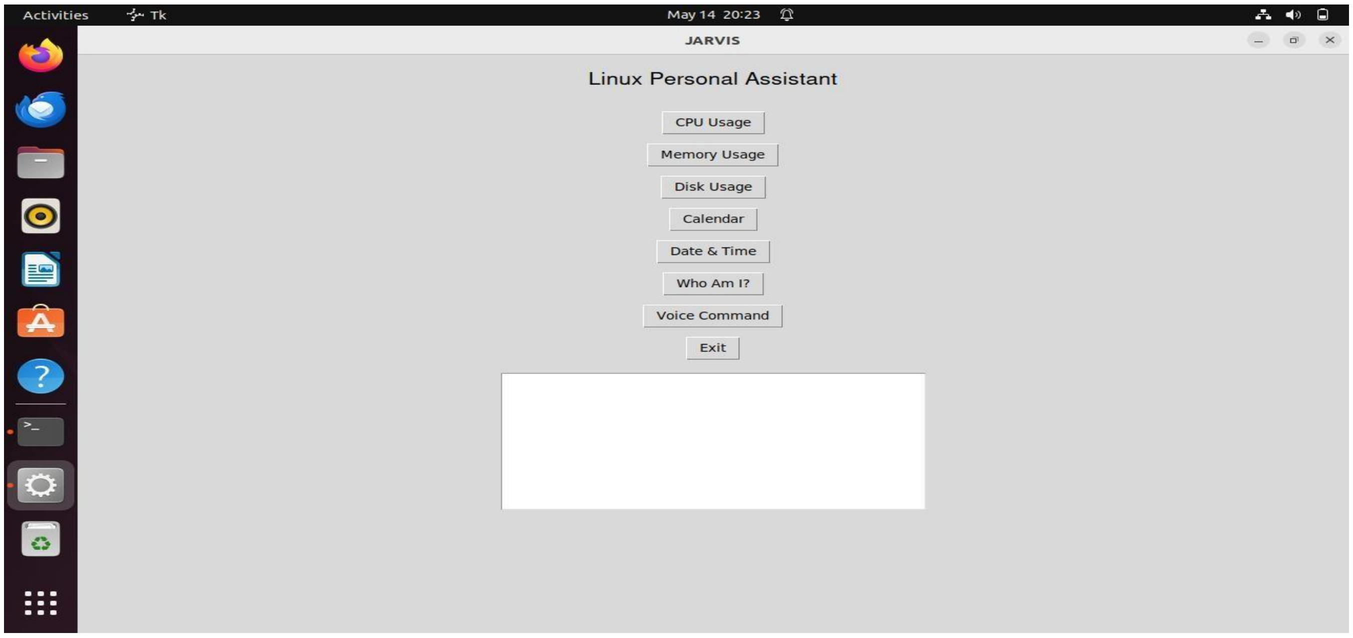


```
abhinav@abhinav:~/linux_assistant$ python3 -m tkinter
abhinav@abhinav:~/linux_assistant$ 
```

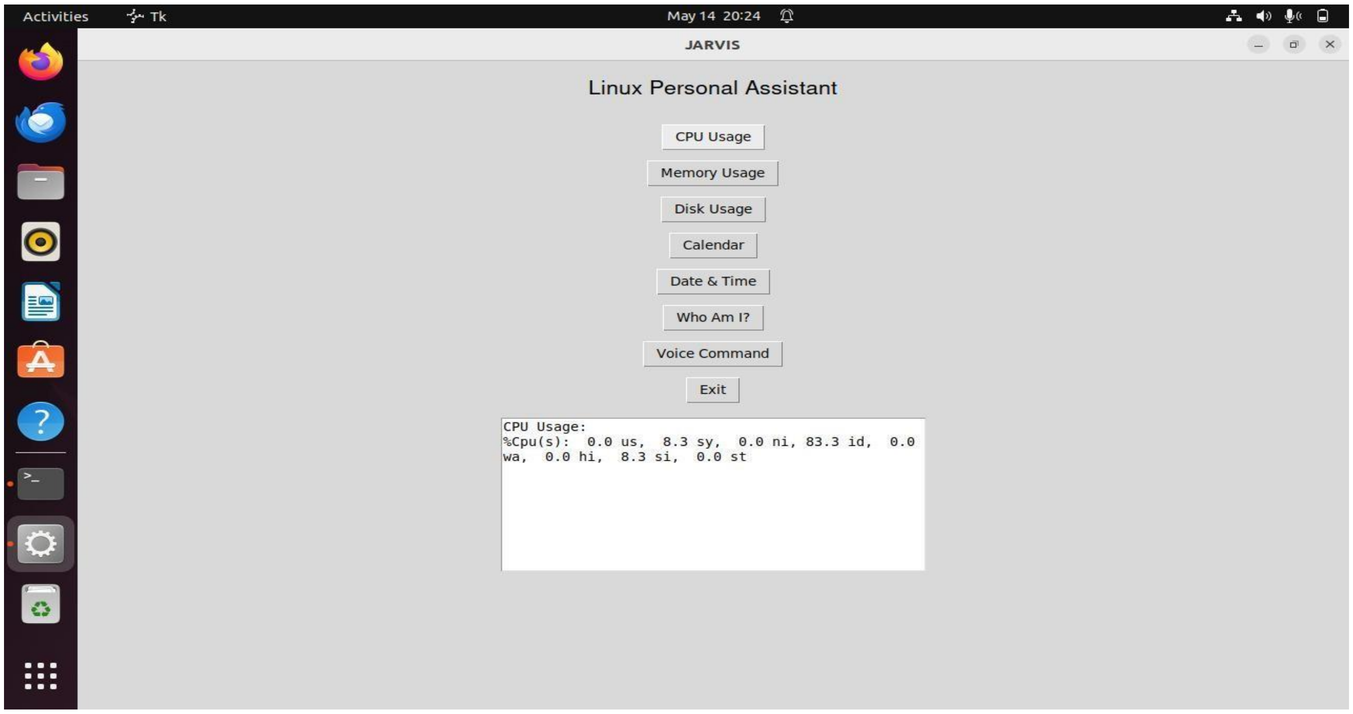
7.1.2 Tkinter Test Command



7.1.3 Tkinter GUI Test Window

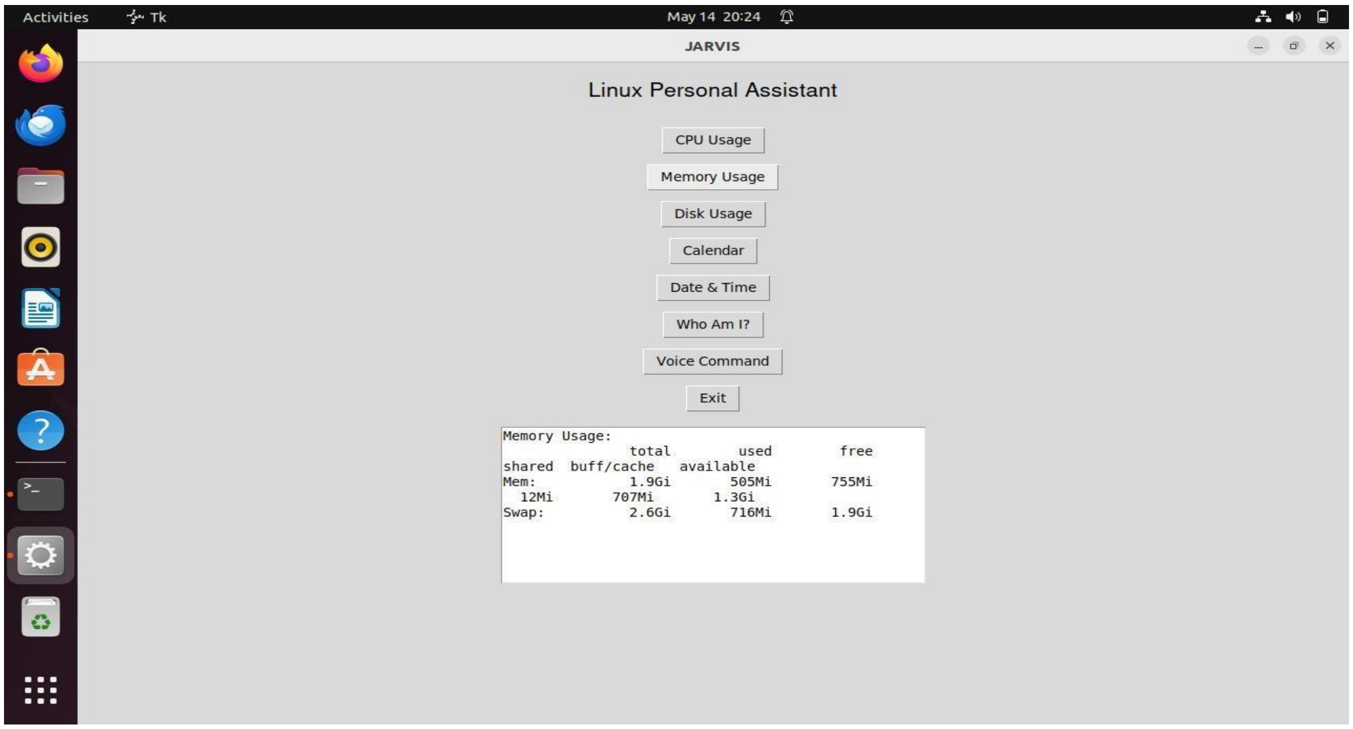


7.1.4 Linux Assistant Main GUI Window



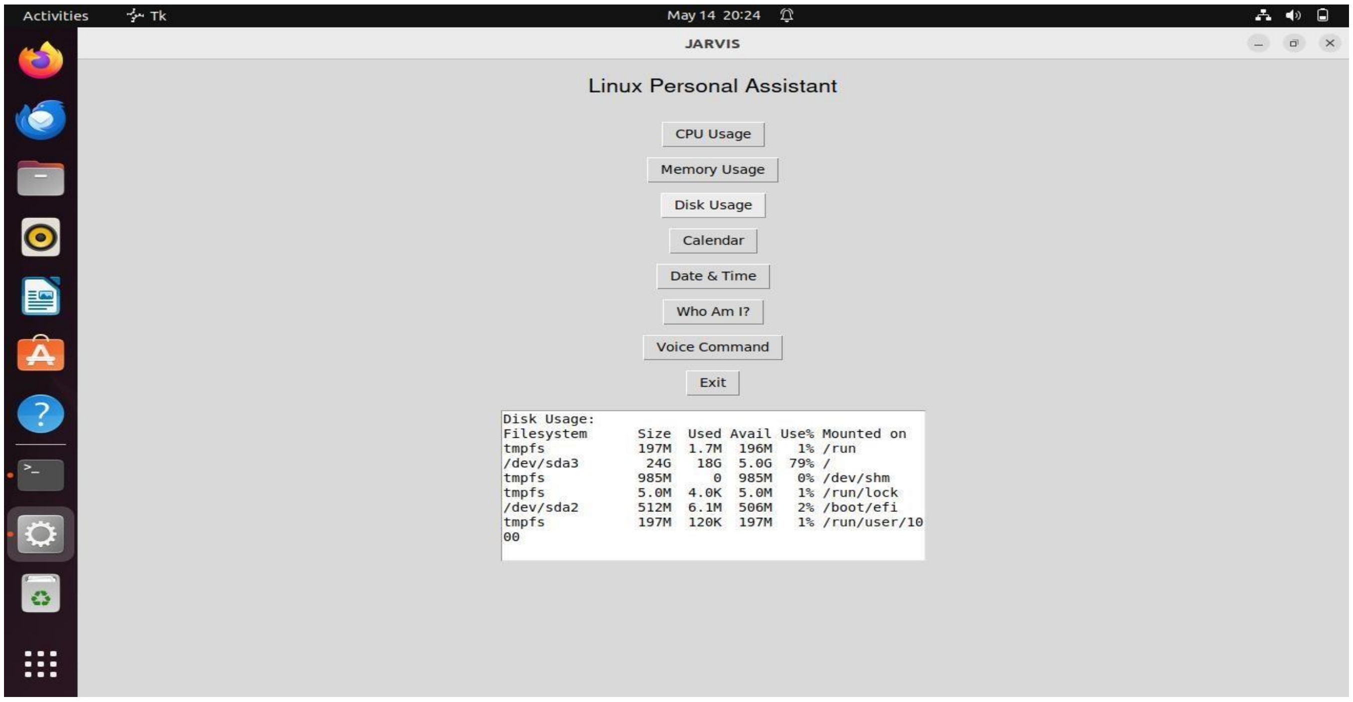
```
CPU Usage:
%Cpu(s):  0.0 us,  8.3 sy,  0.0 ni, 83.3 id,  0.0
wa,  0.0 hi,  8.3 si,  0.0 st
```

7.1.5 CPU Usage



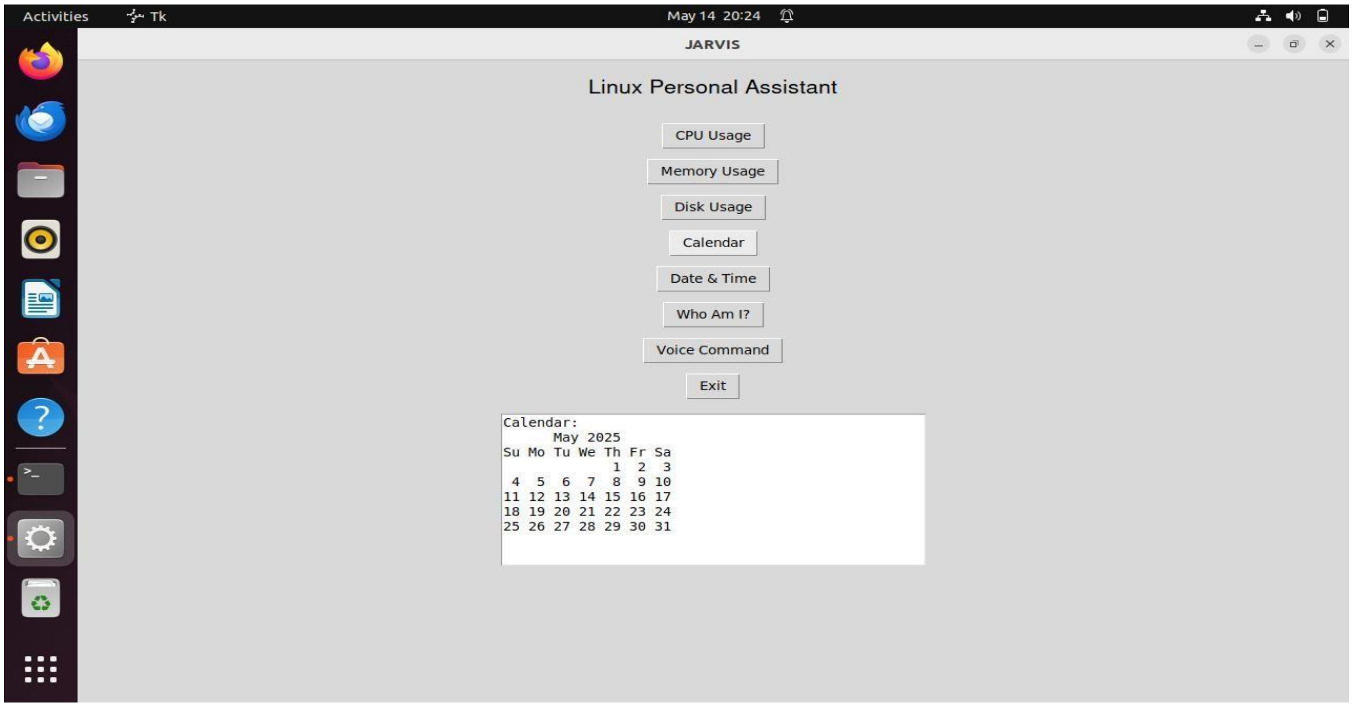
```
Memory Usage:
               total        used        free
shared  buff/cache   available
Mem:           1.9Gi       505Mi       755Mi
  12Mi       707Mi       1.3Gi
Swap:          2.6Gi       716Mi       1.9Gi
```

7.1.6 Memory Usage



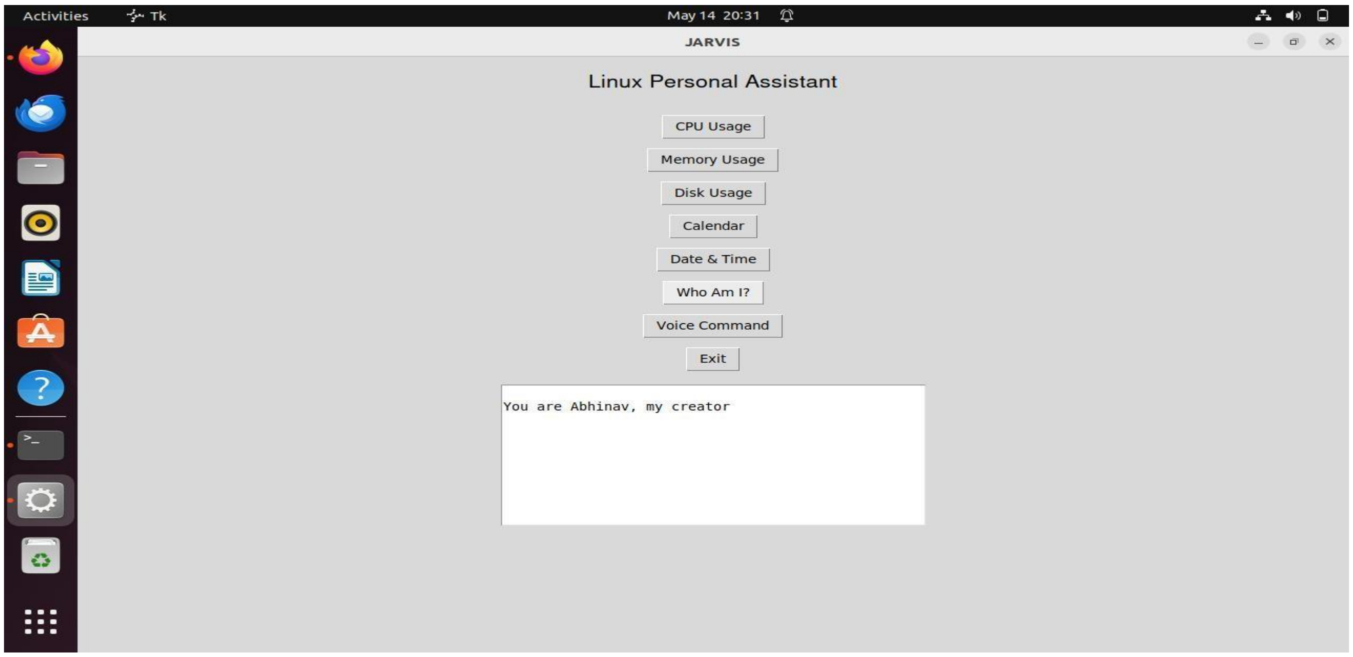
```
Disk Usage:
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           197M  1.7M  196M   1% /run
/dev/sda3        24G   18G  5.0G  79% /
tmpfs           985M     0  985M   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
/dev/sda2       512M  6.1M  506M   2% /boot/efi
tmpfs           197M  120K  197M   1% /run/user/10
00
```

7.1.7 Disk Usage



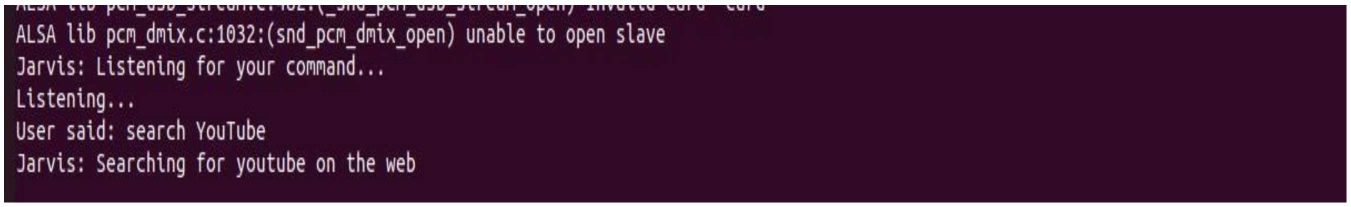
```
Calendar:
      May 2025
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

7.1.8 Calendar

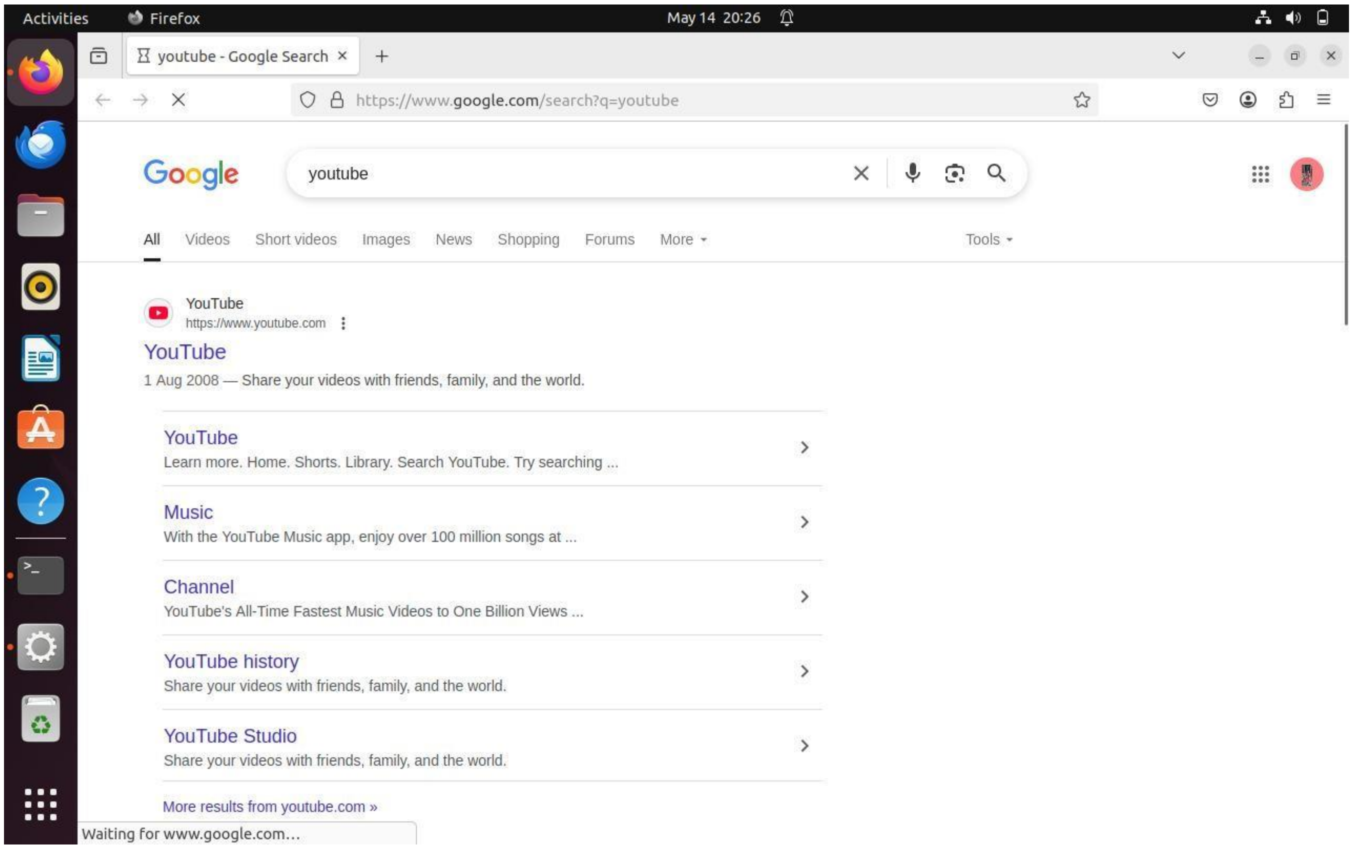


```
You are Abhinav, my creator
```

7.1.9 Who Am I



```
ALSA lib pcm_dmix.c:1032:(snd_pcm_dmix_open) unable to open slave
Jarvis: Listening for your command...
Listening...
User said: search YouTube
Jarvis: Searching for youtube on the web
```



YouTube
https://www.youtube.com
YouTube
1 Aug 2008 — Share your videos with friends, family, and the world.

YouTube
Learn more. Home. Shorts. Library. Search YouTube. Try searching ...

Music
With the YouTube Music app, enjoy over 100 million songs at ...

Channel
YouTube's All-Time Fastest Music Videos to One Billion Views ...

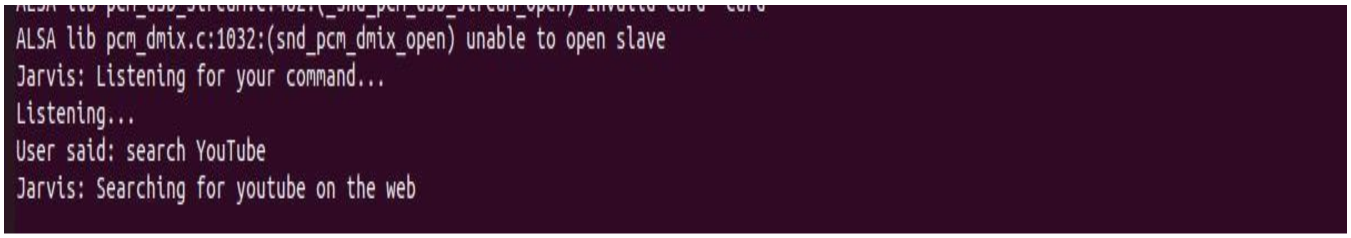
YouTube history
Share your videos with friends, family, and the world.

YouTube Studio
Share your videos with friends, family, and the world.

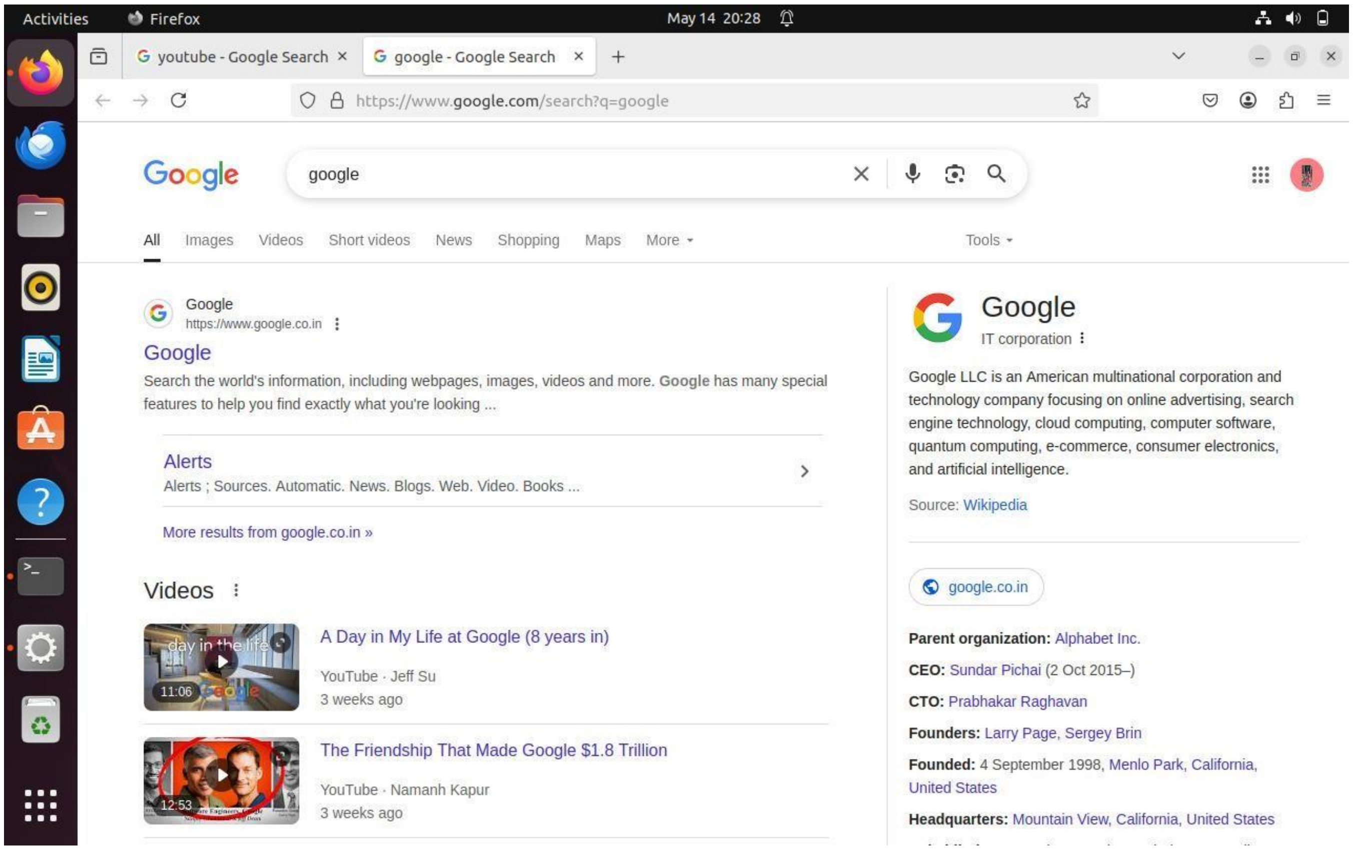
More results from youtube.com »

Waiting for www.google.com...

7.1.10 Search For Youtube



```
ALSA lib pcm_dmix.c:1032:(snd_pcm_dmix_open) unable to open slave
Jarvis: Listening for your command...
Listening...
User said: search YouTube
Jarvis: Searching for youtube on the web
```



Activities   Firefox   May 14 20:28

youtube - Google Search   google - Google Search

https://www.google.com/search?q=google

google

All   Images   Videos   Short videos   News   Shopping   Maps   More   Tools

Google
https://www.google.co.in
Google
Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking ...

Alerts
Alerts ; Sources. Automatic. News. Blogs. Web. Video. Books ...

More results from google.co.in »

Videos

A Day in My Life at Google (8 years in)
YouTube · Jeff Su
3 weeks ago

The Friendship That Made Google $1.8 Trillion
YouTube · Namanh Kapur
3 weeks ago

Google
IT corporation

Google LLC is an American multinational corporation and technology company focusing on online advertising, search engine technology, cloud computing, computer software, quantum computing, e-commerce, consumer electronics, and artificial intelligence.

Source: Wikipedia

google.co.in

Parent organization: Alphabet Inc.
CEO: Sundar Pichai (2 Oct 2015–)
CTO: Prabhakar Raghavan
Founders: Larry Page, Sergey Brin
Founded: 4 September 1998, Menlo Park, California, United States
Headquarters: Mountain View, California, United States

7.1.11 Search For Google

7.2 Troubleshooting Techniques

1. Microphone Issues:

Ensure the microphone is properly connected and selected as the default input device.

2. Internet Connectivity:

A stable internet connection is required for Google Web Speech API to function.

3. Missing Python Packages:

Install required libraries using pip install speechrecognition pyttsx3.

4. Text-to-Speech Errors:

Install additional dependencies like espeak, libespeak1, or pyaudio if pyttsx3 fails.

5. Linux Command Failures:

Verify commands like top, df, and cal work manually in the terminal.

6. GUI Freezing:

Use multithreading to keep the Tkinter GUI responsive during voice input operations.

7. Permission Issues:

Run the script with necessary permissions if access to hardware (like the microphone) is denied.

8. Speech Recognition Accuracy:

Speak clearly and minimize background noise to improve recognition accuracy.

7.3 Logs and Monitoring Tools

1.Print Statements:

Used throughout the script for debugging (e.g., showing recognized voice commands and system status messages).

2.Terminal Output:

Console logs provide real-time feedback on actions like command execution and speech processing.

3.Text Widget Display:

GUI displays command output in the Tkinter text area for user visibility and simple monitoring.

4.Error Handling Logs:

Errors from speech recognition (e.g., UnknownValueError, RequestError) are caught and printed for diagnosis.

5.System Resource Commands:

Tools like top, free, and df monitor CPU, memory, and disk usage via shell commands.

8. Challenges and Limitations

8.1 Problems Faced During Implementation

While creating the Linux Personal Assistant (Jarvis), several difficulties came up:

Voice Functionality Setup: Making Jarvis listen to and speak in a Linux terminal was tricky. Installing tools like speech_recognition and espeak wasn't always smooth. Some systems didn't support them well or required extra setup.

Software Conflicts: Some required packages didn't work well together. Installing one tool sometimes broke another, especially with older or different Linux versions.

Microphone and Speaker Issues: Setting up sound input and output was a challenge. In some systems, the microphone didn't pick up voice properly or speakers didn't play the assistant's voice.

Permission Problems: Jarvis needed special permissions (like admin rights) to run certain tasks such as updating or cleaning the system. Handling this without causing errors or risking security was not simple.

Slow Responses in Some Tasks: Tasks like system updates or cleanup took time, and the script didn't always show progress clearly, making it seem like nothing was happening.

8.2 Workarounds and Fixes

Here's how we tackled those issues:

Backup for Voice Commands: If voice commands didn't work, users could always choose options manually using the terminal menu.

Isolated Setup: We used Python's virtual environments to keep our tools separate from the rest of the system, which helped avoid software clashes.

Better Audio Checks: The script now checks if the microphone and speaker are working before starting the assistant, reducing audio issues.

Safe Use of Admin Commands: Jarvis now asks for the user's password when it needs admin access. If denied, it skips the task safely.

Simplified Feedback: We added simple loading messages so users can see when something is in progress.

8.3 Known Issues or Constraints

Some limitations still exist:

Needs Audio Devices: Voice features won't work without a working mic and speaker, which some systems may not have.

Voice Recognition Isn't Perfect: Jarvis may not always understand every voice command, especially with different accents or background noise.

No Full Graphics: Since it runs in the terminal, there's no fancy graphical interface. Users need to be okay with using the command line.

Root Access Needed: Some features need admin rights, and this can be risky if not used carefully.

Best on Ubuntu: The assistant was mainly tested on Ubuntu. It might not work perfectly on other types of Linux unless tweaked.

9. Conclusion and Future Work

9.1 Summary of Accomplishments

In this project, we successfully created a Linux-based personal assistant named Jarvis that works through the terminal. It allows users to:

- View important system information like CPU and memory usage, uptime, disk space, and calendar.

   Perform system updates and cleanups with simple menu selections.

- Use voice commands to talk to Jarvis and receive spoken responses, making the assistant more interactive and user-friendly.

We were able to combine Linux commands, scripting, a menu-based interface, and voice interaction into one useful and easy-to-use tool.

9.2 Learnings from the Project

This project helped us learn a lot about:

- How to use common Linux commands in real-world applications.
- Automating tasks using bash scripting and Python.
- Working with voice recognition and text-to-speech tools on Linux.
- Managing system permissions, dependencies, and terminal-based user interfaces.
- Troubleshooting audio, command errors, and package conflicts.

It gave hands-on experience with both Linux system administration and software development.

9.3 Future Enhancements

While the assistant is working well, there are a few improvements we'd like to make in the future:

- Improve voice recognition to better understand different accents and work more accurately in noisy environments.
- Add a graphical version (GUI) for users who are not comfortable with terminal menus.
- Include more features like checking battery status, managing installed software, or sending system reports.
- Add multi-language support, so Jarvis can communicate in other languages besides English.
- Integrate with external APIs (e.g., weather, news, or reminders) for more advanced functionality.

10. References

Here are some resources we used during the project:

- Linux Manual Pages
- Ubuntu Documentation
- speech_recognition, pyttsx3, pyaudio – Python voice libraries
- Online forums like Stack Overflow and AskUbuntu for troubleshooting

11. Appendices

11.1 Configuration Files

We used some configuration files for voice tools and system command permissions. For example:

- .asoundrc – to set default audio input/output
- sudoers entries – for allowing certain commands to run with sudo without prompting each time

11.2 Script Listings

Here we include the full code used in the project:

- The main script (Bash or Python)
- Functions for showing CPU, memory, uptime, etc.
- Voice command integration using Python

11.3 Additional Screenshots or Data

This section includes:

- Screenshots of the assistant running in the terminal
- Voice command execution in action
- Sample outputs of system monitoring commands