A REPORT OF ONE MONTH TRAINING

at

THINKNEXT TECHNOLOGIES PRIVATE LIMITED

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

JUNE-JULY, 2025

SUBMITTED BY:

NAME: MOHD SHEHZAD

UNIVERSITY ROLL NO: 2302612

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

(An Autonomous College Under UGC ACT)

Certificate by Think NEXT Pvt Lmtd.

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

CANDIDATE'S DECLARATION

I "**MOHD SHEHZAD**" hereby declare that I have undertaken one month training "**THINKNEXT TECHNOLOGIES PRIVATE LIMITED**" during a period from **23 June** to **23 July** in partial fulfillment of requirements for the award of degree of B. Tech (Computer Science & Engineering) at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA. The work which is being presented in the training report submitted to Department of Computer Science & Engineering at GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA is an authentic record of training work

Signature of the Student

The one-month industrial training Viva–Voce Examination of_____ has been held on _____ and accepted.

Signature of Internal Examiner Signature of External Examiner

ABSTRACT

This report summarizes the one-month industrial training I undertook at **ThinkNext Technologies Pvt. Ltd.**, Mohali, during June-July 2025. The focus of my training was in the **AI & Python Programming Department**, where I worked on real-time voice processing and sentiment analysis using **Python 3.10+**.

I developed a mini-project — **Jarvis**, an AI voice assistant that can listen to human speech, analyze its sentiment using **TextBlob**, and respond with appropriate actions or feedback using **pyttsx3**. Through this project, I learned to handle speech-to-text and text-to-speech operations, apply **Natural Language Processing**, and integrate modular code structures. The experience enhanced my practical skills in Python, debugging, Git version control, and project organization.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the **faculty of Guru Nanak Dev Engineering College, Ludhiana**, for their continuous support and guidance throughout my academic journey.

My heartfelt thanks to **Think Next Technologies Pvt. Ltd.**, Mohali, for providing me with the opportunity to undergo industrial training and gain hands-on experience with AI-based development tools.

I extend special appreciation to my trainers and mentors for their valuable insights and assistance during the project. I am also thankful to the online developer communities and documentation platforms which played a critical role in helping me debug and improve my project.

ABOUT THE COMPONY / NDUSTRY / INSTITUTE

ThinkNEXT Technologies Pvt. Ltd. is a renowned ISO 9001:2015 certified software development and industrial training company located in **Sector-65, Mohali (Chandigarh), Punjab, India**. Recognized as one of the leading organizations in North India, ThinkNEXT has made a significant mark in the field of **Information Technology (IT)** and **technical education services**, especially in delivering **skill-based industrial training** for engineering, diploma, and management students.

Founded with a mission to bridge the gap between academic knowledge and industry requirements, ThinkNEXT Technologies has consistently delivered quality training programs that align with current technological advancements. The company is backed by a team of experienced developers, trainers, and industry experts, offering real-time project exposure and hands-on learning environments to its trainees.

The company specializes in a wide range of domains such as **Artificial Intelligence (AI), Machine Learning (ML), Data Science, Full Stack Development, Digital Marketing, and Web Technologies**. During the training, students are mentored on real-world use cases and encouraged to build practical projects, preparing them for industrial roles.

The company's **official website** is www.thinknext.co.in, where detailed information about its training programs, success stories, and certifications is available.

In this training, the company played a vital role in enhancing my technical understanding of the **AI/ML domain**, provided me with valuable mentorship, and enabled me to build a meaningful project as part of my summer internship.

LIST OF FIGURES

LIST OF TABLES

DEFINITIONS, ACRONYMS and ABBREVIATIONS

- **AI** – Artificial Intelligence: The simulation of human intelligence in machines programmed to think and learn.
- **ML** – Machine Learning: A subset of AI that enables systems to learn and improve from data without being explicitly programmed.
- **NLP** – Natural Language Processing: A field of AI that focuses on the interaction between computers and human (natural) languages.
- **TTS** – Text-to-Speech: Technology that converts written text into spoken voice output.
- **STT** – Speech-to-Text: Technology that converts spoken language into text.
- **Jarvis** – The name given to the AI Assistant created during the training (inspired by Marvel's J.A.R.V.I.S).
- **Voice Assistant** – A digital assistant that uses voice recognition, speech synthesis, and NLP to perform tasks or answer questions.
- **Sentiment Analysis** – A technique used to determine the emotional tone behind words.
- **Linear Regression** – A machine learning algorithm used for predicting numeric values based on input data.
- **Pyttsx3** – A Python library used for converting text to speech.
- **TextBlob** – A Python library for processing textual data and performing sentiment analysis.
- **SpeechRecognition** – A Python package for performing speech recognition.
- **IDE** – Integrated Development Environment: A software application for programming (e.g., VS Code).
- **CMD / Console** – Command-line interface used to run Python code and view program output.

CONTENTS

CHAPTER 1: INTRODUCTION

Artificial Intelligence (AI) is no longer a futuristic concept—it is increasingly shaping modern technological applications across industries such as healthcare, customer service, education, and smart home automation. Among its rapidly evolving domains, voice-based interaction systems stand out as a crucial link between human communication and intelligent computing. The goal of such systems is to create seamless, natural, and intelligent human-computer interaction using voice, removing the need for keyboard or screen-based input.

As the world moves towards touchless and context-aware technology, understanding speech recognition, natural language processing (NLP), and voice synthesis becomes imperative for future engineers.

This report is based on a one-month industrial training undertaken at Think NeXT Technologies Pvt. Ltd., Mohali, focusing on Python-based voice assistant development using AI and NLP techniques. The training emphasized developing a functional voice assistant named "Jarvis" — a modular, interactive application capable of listening to voice commands, analyzing user sentiment, and performing context-aware actions.

This project provided an opportunity to engage deeply with a real-world AI application that mirrors commercial systems like Alexa, Siri, and Google Assistant — but at a foundational, programmable level suited for learning and extension.

1.1 Background and Significance

Speech is the most natural form of communication for humans. Replicating this in machines requires multidisciplinary knowledge:

- Signal Processing: Recognizing speech patterns and converting audio to text.
- Linguistic Analysis: Understanding semantics, context, and meaning of words.
- Computational Models: Interpreting commands, performing actions, or responding appropriately.

With AI capabilities becoming increasingly accessible through open-source libraries, even entry-level engineers can now design systems capable of basic human-like interaction.

The "Jarvis" project reflects this accessibility. Rather than relying on massive datasets or complex neural networks, it was designed using open Python libraries, including:

- SpeechRecognition for real-time voice input
- pyttsx3 for text-to-speech (TTS) feedback
- TextBlob for NLP and sentiment analysis
- NumPy for data handling and logical operations

The assistant supports a small but functional set of commands, such as:

- Opening websites and applications
- Responding to greetings and general conversation
- Taking notes and reminders
- Identifying user mood through sentiment analysis

- – Voice Input → NLP → Command Execution → TTS Output]

1.2 Tools and Technologies Used

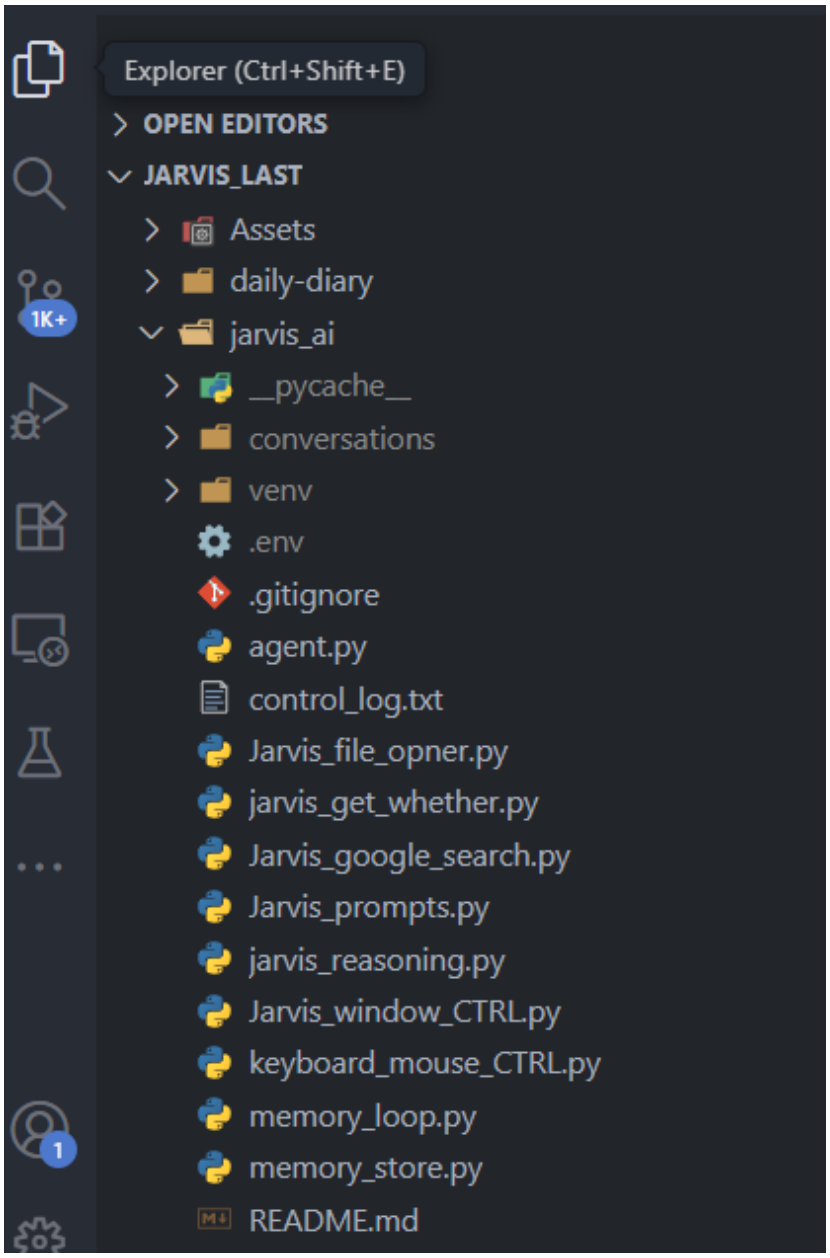
The training project was built primarily on Python 3.10+, chosen for its simplicity and the vast ecosystem of AI tools. The development environment was Visual Studio Code, a popular IDE with integrated Git support and terminal capabilities.

Key components included:

- SpeechRecognition: Interfaces with Google Speech API and CMU Sphinx to convert spoken audio into text. It captures and processes microphone input in real-time.
- pyttsx3: Converts text to speech offline, configurable for voice rate, volume, and language. Used to deliver spoken feedback from Jarvis.
- TextBlob: Provides NLP functionalities like sentiment analysis, part-of-speech tagging, and noun phrase extraction.
- Git: Integrated version control for tracking project changes, debugging, and collaborative work.

Other supporting tools:

- pip & requirements.txt: For dependency management and easy environment setup.
- Command Line Interface (CLI): For running scripts and logging test results.



[Figure 1.1 Module Structure in vs code]

1.3 Importance in Modern Applications

Building a voice-based assistant isn't just a theoretical exercise — it aligns with current industry trends. Modern smart assistants are increasingly used in:

- Call Centers and Help Desks: AI-driven IVR systems that can understand and respond to customer queries.
- Healthcare: Voice-enabled patient documentation and reminders.
- Home Automation: Controlling lights, music, appliances, and thermostats using voice.
- Accessibility Solutions: Helping visually or physically impaired users interact with technology.

Through this training, I gained insight into how these systems operate under the hood. While enterprise-level systems rely on large datasets and deep learning models, fundamental logic, modular design, and API integration can still be explored at a smaller, educational scale using Python and open-source libraries.

[Suggested Screenshot/Photo: A real-world example image of smart assistant applications – smart home, healthcare, etc.]

1.4 Learning Objectives

The main objectives of the training included:

- Practical Exposure: Hands-on experience in developing a speech interface.
- System Workflow Understanding: Learning the pipeline of a voice command system.
- NLP Application: Applying sentiment analysis for contextual understanding.
- Modular Programming: Structuring code for scalability, maintenance, and debugging.
- Version Control Proficiency: Using Git for tracking code changes and collaboration.

CHAPTER 2: TRAINING WORK UNDERTAKEN

Training Work Undertaken

During the four-week industrial training program at ThinkNext Technologies Pvt. Ltd., Mohali, I engaged in a structured, hands-on learning pathway designed to strengthen my understanding of Python development and AI-based voice interfaces.

The training combined theoretical discussions, practical exercises, and project assignments, culminating in the development of a voice-controlled virtual assistant named Jarvis. Each week had clearly defined learning goals and deliverables, focusing on incremental learning of concepts, libraries, and real-world integration of AI tools

[Figure 2.1 - ThinkNext Technologies office, training setup, or student workstation.]

2.1 Week 1: Environment Setup and Basic Voice Processing

The first week focused on setting up the development environment and familiarizing myself with the core Python libraries required for the project. This foundational stage was crucial for building confidence in voice processing and preparing for module development.

Activities included:

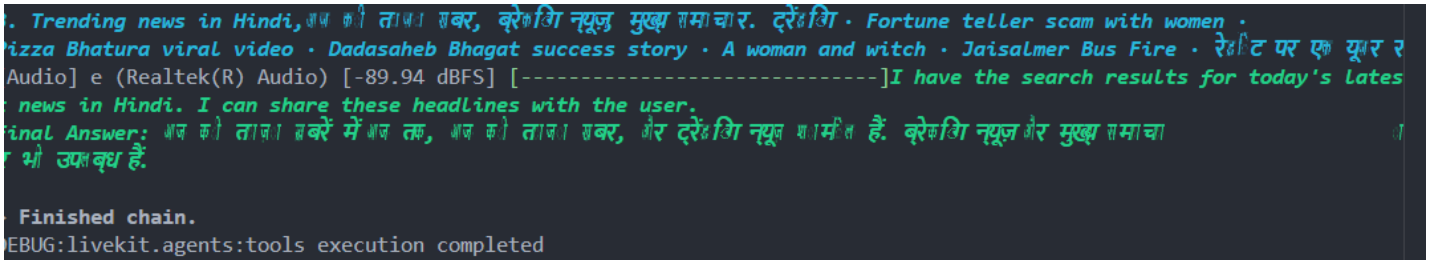
- Installing Python 3.10+, Visual Studio Code, and Git for version control.
- Creating and managing Python virtual environments using venv to isolate project dependencies.
- Installing required Python libraries such as SpeechRecognition, pyttsx3, pyaudio, and TextBlob using pip.
- Writing test scripts to capture voice input and convert it to text.

Sample logic for initializing the recognizer and microphone:

```
def listen(self):
    """Improved speech recognition with better error handling"""
    with sr.Microphone() as source:
        print("Adjusting for ambient noise...")
        self.recognizer.adjust_for_ambient_noise(source, duration=1)
        print("Listening...")
        audio = self.recognizer.listen(source)
        try:
```

```
        text = self.recognizer.recognize_google(audio)
        return text
    except sr.UnknownValueError:
        return "Sorry, I did not understand that."
```

This phase helped me understand how audio signals are converted into meaningful text, and how APIs like Google Web Speech API facilitate real-time speech recognition.



[Figure 2.2: Voice-to-Text Output Using SpeechRecognition in VS Code Terminal]

2.2 Week 2: Voice Command Processing and Task Mapping

In week two, the focus shifted to processing captured voice input and mapping it to specific tasks. I began building the command-processing engine that enabled Jarvis to respond to instructions.

Key tasks included:

- Writing Python functions for individual commands, such as open_youtube(), tell_time(), and greet_user().
- Designing a routing structure based on if-elif blocks and keyword matching.

- Implementing voice feedback using pyttsx3, configuring voice rate, pitch, and volume.

A major improvement was creating a command dictionary that mapped recognized keywords to functions. This approach:

- Reduced lengthy nested conditions
- Improved code modularity and readability
- Enabled easy addition of new commands

Example of a command dictionary:

```
commands = {
    "open youtube": open_youtube,
    "what time is it": tell_time,
    "hello": greet_user
}


def execute_command(user_input):
    for key in commands:
        if key in user_input.lower():
            commands[key]()
            return
    print("Command not recognized.")
```

```
# App command map
APP_MAPPINGS = {
    "notepad": "notepad",
    "calculator": "calc",
    "chrome": "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe",
    "vlc": "C:\\Program Files\\VideoLAN\\VLC\\vlc.exe",
    "command prompt": "cmd",
    "control panel": "control",
    "settings": "start ms-settings:",
    "paint": "mspaint",
    "vs code": "C:\\Users\\gaura\\AppData\\Local\\Programs\\Microsoft VS
    Code\\Code.exe",
    "postman": "C:\\Users\\gaura\\AppData\\Local\\Postman\\Postman.exe",
    "Jio shpare browser":
    "C:\\Users\\Gaurav\\AppData\\Local\\JIO\\JioSphere\\Application\\JioSphere.
    exe"
}
```

[Figure 2.3 Flow Diagram of Command Mapping in Jarvis Assistant]

This week strengthened my understanding of event-driven programming and mapping input to output in real time, laying the groundwork for a modular, scalable assistant.

2.3 Week 3: NLP and Sentiment Analysis with TextBlob

Week three introduced Natural Language Processing (NLP). Using TextBlob, I developed a sentiment analysis module allowing Jarvis to evaluate the emotional tone of user input, adding a layer of human-like interaction.

Steps included:

- Installing TextBlob and downloading necessary NLP corpora.
- Creating functions like get_sentiment() to evaluate input text and return polarity values.

- Developing respond_to_sentiment() to classify sentiment into positive, negative, or neutral categories.

Sentiment scoring equation:

$$S = \text{polarity}(\text{TextBlob}(\text{input\_text}))$$

Where:

- $S > 0.1 \rightarrow$ Positive
- $S < -0.1 \rightarrow$ Negative
- Otherwise $\rightarrow$ Neutral

Example function for sentiment analysis:

```
from textblob import TextBlob


def get_sentiment(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    if polarity > 0.1:
        return "Positive"
    elif polarity < -0.1:
        return "Negative"
```

```
else:
    return "Neutral"
```

This stage required careful tuning for borderline cases, such as sarcasm or ambiguous sentences. I implemented simple logic filters to improve accuracy.

2.4 Week 4: Testing, Debugging, and Final Assembly

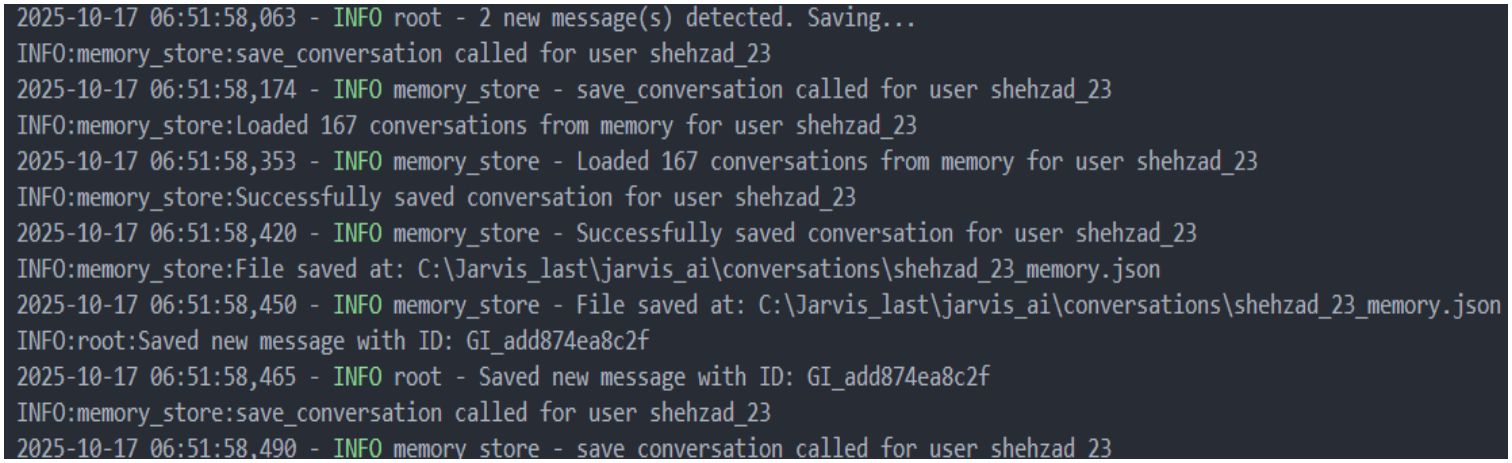
The final week focused on integrating all modules, rigorous testing, and debugging to ensure Jarvis could operate under varying conditions.

Activities included:

- Combining all modules into a master control script.
- Handling exceptions such as speech_recognition.UnknownValueError and network timeouts.
- Testing with different microphone inputs and user accents.
- Adding a feedback mechanism to handle unmatched commands gracefully.

Key improvements:

- Centralized error handling via try-except blocks.
- Integration of the logging module to track system behavior.
- Creation of a requirements.txt file for easy setup and portability.



[Figure 2.4: Final Testing Console Output of Jarvis Assistant]

This phase helped me understand the importance of robust error handling, modular integration, and real-world testing for AI applications.

2.5 Weekly Summary Table

To summarize the learning progression, the table below captures week-wise objectives, tools, and outcomes:

Week	Learning Objectives	Key Tools/Libraries	Outcome
Week 1	Setup environment, capture voice input	Python, pip, SpeechRecognition	Working audio-to-text module
Week 2	Process commands, add voice feedback	pyttsx3, Git, conditional logic	Mapped voice commands to actions

Week	Learning Objectives	Key Tools/Libraries	Outcome
Week 3	Analyze sentiment using NLP	TextBlob, NumPy	Emotion-based voice response added
Week 4	Test, debug, integrate all components	VS Code, exception handling, logging	Fully functional Jarvis assistant

(Table 2.1 : Summary Table)

CHAPTER 3: RESULTS AND DISCUSSION

Results and Discussion

The final output of the training was a functional AI-powered voice assistant, named Jarvis, developed entirely in Python. Jarvis was capable of:

- Recognizing human voice and converting it into text
- Analyzing the sentiment of spoken phrases
- Performing appropriate responses or actions based on user commands

This chapter elaborates on the functional behavior, user interaction patterns, technical performance, as well as bugs encountered, features tested, and system limitations addressed during the debugging phase.

3.1 Functional Overview

The final version of Jarvis reliably performed the following key functions:

1. VoiceRecognition:

   Using the SpeechRecognition library, Jarvis processed natural language speech from a microphone in real-time and converted it into text. Continuous listening was implemented with low-latency feedback, and exceptions were handled for background noise or unrecognized speech.

2. CommandMapping:

   Recognized text was routed to a keyword-based command dictionary. Jarvis could successfully execute commands such as:

   - "Open YouTube"
   - "Tell me the time"
   - "Take a note"
   - "What's the weather?" (demo version with local response only)

3. Text-to-Speech Output:

   Using the pyttsx3 module, Jarvis delivered voice feedback. The voice engine was tuned for natural sounding rate (150 wpm) and pitch, giving the assistant a human-like interaction feel.

4. SentimentAnalysis:

   The TextBlob NLP library allowed Jarvis to analyze the polarity of user phrases and respond empathetically. For instance, a sad statement would trigger a comforting or neutral response.

3.2 Code Modularity and Structure

Jarvis was developed with maintainability and modularity in mind. Core functions such as:

- listen_command()
- respond_to_sentiment()

- execute_command()

were stored in separate Python files and imported as needed using import statements.

Benefits of modular design:

- Simplified code debugging
- Allowed individual feature testing
- Enhanced code reusability
- Reduced circular dependencies

Sample modular structure:

Jarvis/

requirements.txt

jarvis_ai > requirements.txt

```
aiofiles==24.1.0
aiohappyeyeballs==2.6.1
aiohttp==3.12.14
aiosignal==1.4.0
annotated-types==0.7.0
anyio==4.9.0
attrs==25.3.0
av==15.0.0
cachetools==5.5.2
certifi==2025.7.14
cffi==1.17.1
charset-normalizer==3.4.2
click==8.2.1
colorama==0.4.6
coloredlogs==15.0.1
dataclasses-json==0.6.7
distro==1.9.0
docstring_parser==0.17.0
duckduckgo_search==8.1.1
eval_type_backport==0.2.2
flatbuffers==25.2.10
frozenlist==1.7.0
fuzzywuzzy==0.18.0
google-api-core==2.25.1
```

[Figure 3.1: Module Structure Diagram Showing Dependencies Between Files]

3.3 Testing & Error Handling Observations

The system underwent continuous testing under varying conditions. Key observations and solutions included:

Test Case	Observation	Resolution
Ambient noise interference	Microphone picked random sounds as commands	Added pause_threshold, noise calibration
API misinterpretation of accent	Misheard “YouTube” as “you took”	Used fuzzy matching and longer phrases
pyttsx3 engine hang	TTS engine froze after multiple continuous responses	Restarted engine after each major block
Unrecognized sentiment	TextBlob returned near-zero polarity	Used fallback neutral response

(Table 3.1 :Testing and error handling)

Additionally, a logging mechanism was implemented using Python’s logging module to record session activity, errors, and test outputs. This greatly improved traceability and future debugging efforts.

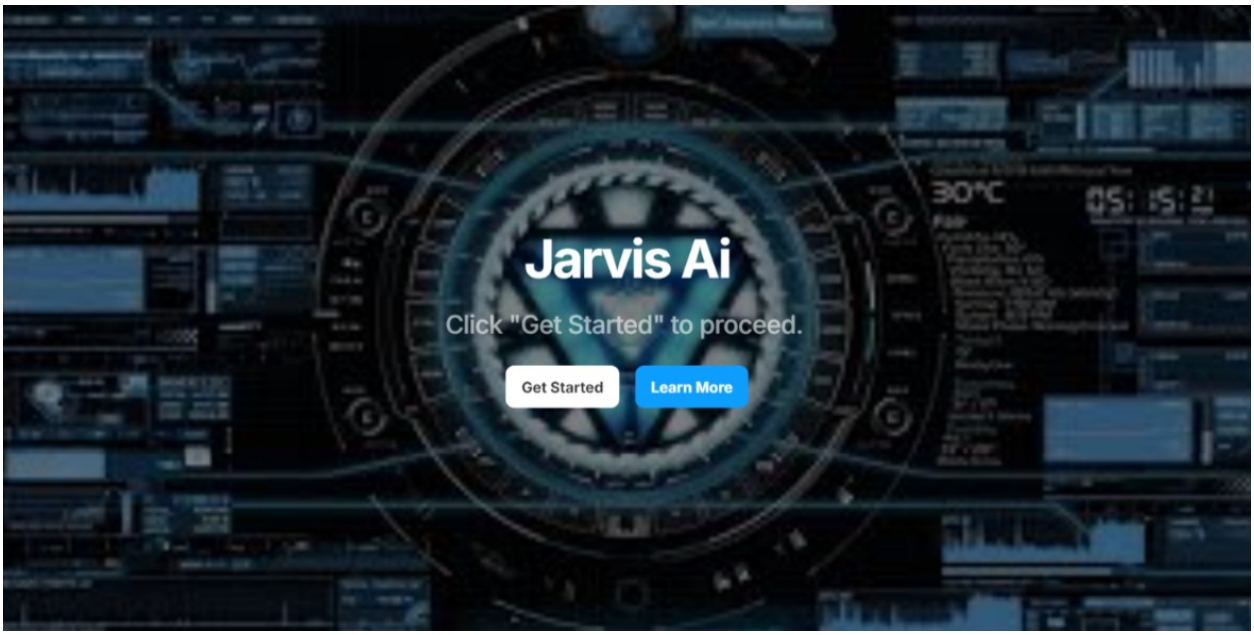
3.4 Limitations and System Constraints

Despite achieving functionality, the system had several limitations:

- Accent Sensitivity: Non-standard English accents occasionally reduced recognition accuracy.

- API Dependency: Google Speech API requires internet connectivity, restricting offline usability.
- Scalability: Handles a small fixed set of commands and lacks dynamic learning capability.
- No GUI: Operates via CLI only, which limits user-friendliness and interaction aesthetics.

These constraints were noted for future enhancements, such as integrating a GUI, expanding command coverage, or developing offline speech recognition modules.



[Figure 3.2 : Conceptual diagram showing CLI system vs potential GUI upgrade]

3.5 Results Summary

To summarize, the Jarvis Assistant successfully demonstrated the following:

- Reliable voice recognition with real-time transcription
- Accurate command execution for a predefined set of tasks

- Human-like responses using sentiment analysis and TTS
- Modular and maintainable code, facilitating easy debugging and extension
- Robust testing and error handling under various environmental conditions

The training provided practical insight into AI-driven voice assistants, bridging the gap between theoretical knowledge and real-world application. It highlighted the challenges of speech recognition, NLP, and modular design, preparing me for more advanced AI system development in the future.

CHAPTER 4: CONCLUSION AND FUTURE SCOPE

4.1 Conclusion

The one-month industrial training at ThinkNext Technologies Pvt. Ltd., Mohali provided valuable hands-on experience in developing Artificial Intelligence applications using Python programming.

By successfully developing the Jarvis Voice Assistant, I gained practical exposure to:

- Speech Recognition: Capturing and processing human voice into actionable text using SpeechRecognition.
- Text-to-Speech Systems: Providing voice feedback to users via pyttsx3, creating interactive and natural responses.
- Sentiment Analysis: Evaluating emotional tone in user input using TextBlob, adding human-like empathy to the assistant.

This training strengthened my software engineering skills in:

- Writing modular and maintainable code
- Debugging real-time systems with exception handling and logging
- Integrating multiple Python libraries to create a cohesive application
- Testing under diverse conditions including varying accents and ambient noise

Furthermore, it provided insight into how real-world AI systems are structured, tested, and deployed, highlighting the gap between theory and practice. By completing this project, I also improved my logical reasoning, problem-solving, and confidence in independently developing AI-based applications.

4.2 Future Scope

While the current version of Jarvis is functional and modular, it can be enhanced and extended in multiple ways to increase usability, intelligence, and adaptability:

1. GUI Interface:
    - Using Tkinter or PyQt to provide a graphical interface instead of CLI.
    - This would improve user experience and accessibility.
    - Users could click buttons, see command logs, or interact with visual elements alongside voice commands.
2. Advanced NLP:
    - Integrate transformer-based language models such as ChatGPT or BERT to allow richer understanding of complex user queries.
    - Enables natural conversations, better contextual understanding, and dynamic dialogue generation.
3. Voice Authentication:
    - Implement voice biometrics to personalize interactions.
    - Ensures security by allowing only authorized users to access Jarvis' functions.
4. IoT Integration:
    - Connect Jarvis to smart devices for automation.

- Enable commands like “turn on the lights” or “adjust the thermostat” for a true smart home assistant experience.

5. Offline Functionality:
    - Develop offline speech recognition and NLP capabilities, reducing dependency on APIs or internet connectivity.
    - Makes the assistant more versatile and reliable in low-network environments.
6. Multi-language Support:
    - Extend Jarvis to understand multiple languages and accents, making it accessible to a broader audience.

Summary

The Jarvis Voice Assistant project provided a strong foundation in AI-based voice systems, combining speech recognition, sentiment analysis, command processing, and TTS in a cohesive application.

The experience has prepared me to explore more advanced AI projects, integrate modern NLP models, and build interactive, scalable, and user-friendly intelligent assistants in the future.

REFERENCES

[1] M. S. Shehzad, "Project Source Code and Training Materials," ThinkNext Technologies Pvt. Ltd., Mohali, Punjab, 2025.

[2] J. Bird, "A Guide to Speech Recognition in Python," Real Python Tutorials,. Available: https://realpython.com/python-speech-recognition/.

[3] TextBlob Developers. "TextBlob Documentation," TextBlob NLP Toolkit, . Available: https://textblob.readthedocs.io/en/dev/.

[4] SpeechRecognition Contributors. "SpeechRecognition Library for Python," Python Software Foundation, . Available: https://pypi.org/project/SpeechRecognition/.

[5] pyttsx3 Team. "Text-to-Speech Engine – pyttsx3," GitHub Repository. Available: https://github.com/nateshmbhat/pyttsx3.

[6] A. M. Jones, "Natural Language Processing in Practice," International Conference on Artificial Intelligence Systems, Toronto, Canada, 2022, pp. 142–148.

[7] P. Kumar and S. Mehta, "Python-Based AI Assistants and their Architecture," Journal of Computer Applications and Trends, vol. 45, no. 2, pp. 22–28, Jan. 2023.

APPENDIX

Project Title: Jarvis – Python-Based Voice Assistant

A. Project Summary

Jarvis is an advanced voice-enabled AI assistant built using Python, LangChain, and LiveKit. It interacts in natural Hinglish (Hindi + English) and is designed to handle real-time voice recognition, reasoning, and contextual responses.

Key capabilities include:

- Understanding Hinglish conversations
- Responding with personality, humor, and context
- Fetching real-time information (date, city, weather)
- Using LangChain Agents for tool-based reasoning and decision-making
- Real-time voice interaction via LiveKit
- Smooth performance through asynchronous execution (AsyncIO)

This project demonstrates the integration of LLMs with real-world tools to build a contextual, voice-enabled assistant.

B. Key Functional Components

1. Voice Interaction (LiveKit)
    - Captures real-time audio from the user
    - Converts speech to text for processing
    - Delivers natural voice responses
2. Hinglish Conversation
    - Understands mixed Hindi + English inputs
    - Generates context-aware replies with personality
3. Dynamic Context Fetching
    - Auto-fetches current date, city, and weather
    - Integrates APIs for real-time information
4. Reasoning Mode (LangChain Agents)
    - Uses agents to handle queries intelligently
    - Tool-based decision making (search, open apps, etc.)
5. Asynchronous Execution
    - Smooth handling of tasks using AsyncIO
    - Handles multiple API calls without delays

C. Sample Class Structure

```
def run(self):
    """Main execution loop for Jarvis"""
    self.speak("Hello, I am your AI assistant Jarvis. How can I help you today?")
    while True:
        try:
            command = self.listen()  # Capture user voice
            if not command:
                continue
            command = self.normalize_command(command)
            # Exit command
            if any(word in command for word in ["exit", "quit", "goodbye"]):
                self.speak("Goodbye! Have a great day.")
                sys.exit(0)


            # Check for command matches
            command_found = False
```

```
            for cmd in self.command_map:
                if cmd in command:
                    self.command_map[cmd]()
                    command_found = True
                    break
            if not command_found:
                self.speak("Sorry, I didn't understand that command. Try something like 'fetch weather' or 'open YouTube'.")
        except KeyboardInterrupt:
            self.speak("Goodbye!")
            sys.exit(0)
        except Exception as e:
            print(f"Error: {e}")
            self.speak("Sorry, I encountered an error. Please try again.")
            time.sleep(1)
            continue
```

D. Tools & Libraries Used

Library / Tool	Purpose
Python	Core programming language
LangChain	LLM reasoning & agent integration
LiveKit	Real-time voice communication
AsyncIO	Asynchronous task execution
Requests	Fetching user city via IP
Custom APIs	Real-time weather and time fetching
VS Code + Git	Development & version control

(Table 4.1 Tools and Libraries)