A

Learning Project-II Report

On

**“Language Translator”**

Submitted in partial fulfillment of

The requirements for the 4th Semester Sessional Examination of

BACHELOR OF TECHNOLOGY

IN

**COMPUTER SCIENCE & ENGINEERING**

By

**NAME**

**ANSUMAN MAHAPATRA**

**BISWAJEET PATRA**

**TAPAN KUMAR SAHOO**

**Registration No.**

**23UG010605**

**23UG010614**

**23UG010632**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

****

**GANDHI INSTITUTE OF ENGINEERING AND TECHNOLOGY   
UNIVERSITY, ODISHA, GUNUPUR**

**2024 - 25**

**GANDHI INSTITUTE OF ENGINEERING AND TECHNOLOGY   
UNIVERSITY, ODISHA, GUNUPUR**

***Dist. - Rayagada-765022, Visit us:-*www.giet.edu**

**Department of Computer Science & Engineering**

**CERTIFICATE**



This is to certify that the project work entitled “Language Translator” has been

successfully completed by Ansuman Mahapatra (23UG010605), Biswajeet Patra (23UG010614), and Tapan Kumar Sahoo (23UG010632) as a part of the 4th Semester Learning Project-II. This project has been carried out in partial fulfillment of the requirements for the 4th Semester Sessional Examination of the Bachelor of Technology in Computer Science and Engineering during the academic year 2024-25. The work has been submitted to the department as a part of the academic evaluation process.

Proctors/Class Teacher Project Coordinator, 2nd Year

HoD, CSE, 2nd Year

**ACKNOWLEDGEMENT**

We would like to express our heartfelt gratitude to our esteemed Class Teacher Mr. Himanshu Barik, Department of Computer Science and Engineering, for providing us with the opportunity to undertake this project. His active support and guidance have been instrumental in the successful completion of this project report.

We also extend our sincere thanks to Mr. Bhavani Sankar Panda, Project Coordinator, Dr. D Anil Kumar, Head of the Department of Computer Science and Engineering, and Prof. (Dr.) Kakita Murali Gopal, Deputy Dean, Computational Science, SOET, for their consistent support, guidance, and assistance.

Furthermore, we would like to thank our friends and family members for their unwavering support and encouragement throughout the project's execution. Their contributions have been invaluable to us

Ansuman Mahapatra (23UG010605)

Biswajeet Patra (23UG0106014)

Tapan Kumar Sahoo (23UG010632)

ACKNOWLEDGEMENT

We express our sincere gratitude to Himanshu Barik of Computer Science and Engineering for giving us an opportunity to accomplish the project. Without his active support and guidance, this project report has not been successfully completed.

We also thank our class teacher Mr. Himanshu Barik for his constant support and guidance provided during this project period.

We also thank Mr. Bhavani Sankar Panda, Project Coordinator, D. Anil Kumar, Head of the Department of Computer Science and Engineering, Prof. Dr. Kakita Murali Gopal , Dy. Dean, Computational Science, SOET for their consistent support, guidance and help.

Thanks to all my team members Biswajeet Patra and Tapan Kumar Sahoo for going strength with me without them, this learning project is incomplete.

ANSUMAN MAHAPATRA(23UG010605)

BISWAJEET PATRA(23UG010614)

TAPAN KUMAR SAHOO(23UG010632)

INDEX

CHAPTER-I

ABSTRACT

INTRODUCTION

 Purpose of the Project

 Project Scope

 Product Features

CHAPTER-II

WORKDONE IN RELATED AREA

CHAPTER-III

SYSTEM ANALYSIS

 User Requirement

 Hardware Requirement

 Software Requirement

CHAPTER-IV

SYSTEM DESIGN AND SPECIFICATIONS

HIGN LEVEL DESIGN

 Project Model

 Flow Chart

 Entity-Relationship Diagram (E-R Diagram)

LOW LEVEL DESIGN

 Pseudo Code / Algorithm

 Screen-Shot Diagram

CHAPTER-V

CODING

CHAPTER-VI

TESTING

CHAPTER-VII

CONCLUSION AND LIMITATION

 Summary of Project Outcomes

 Identified Limitations

CHAPTER-VIII

RESFERENCES / BIBLIOGRAPHY

CHAPTER – I

ABSTRACT:-

Language barriers often pose challenges in communication, especially in a globalized world where multilingual interactions are common. The **Real-Time Language Translator** is designed to bridge this gap by providing instant text translations using the Google Translator API. This application, developed in **Python** using **Tkinter** for the graphical user interface, enables users to input text in one language and receive an accurate translation in another. Additionally, it features **speech-to-text recognition**, allowing users to dictate messages that are then translated into the desired language.

**Objectives**

The primary objectives of this project are:

* To create a **user-friendly** and **efficient** real-time translation tool.
* To integrate **Google Translator API** for accurate translations.
* To implement **speech recognition** for hands-free text input.
* To ensure a **responsive and interactive** graphical interface using Tkinter.

**Methodology**

The project follows a structured approach:

1. **User Input Handling:** Users can enter text manually or use speech-to-text for automatic input.
2. **Language Selection:** A dropdown menu allows selection of source and target languages.
3. **Translation Process:** The Google Translator API is used to fetch and display real-time translations.
4. **Speech Recognition:** Implemented using the SpeechRecognition library to convert spoken words into text.
5. **Graphical User Interface:** Designed with Tkinter for an intuitive and dynamic experience.
6. **Multithreading:** Ensured smooth performance by running speech recognition on a separate thread to prevent UI lag.

**Outcomes**

The project successfully delivers a fully functional **real-time language translation application** with the following results:

* **Accurate text translation** for multiple languages.
* **Voice-to-text input**, allowing hands-free interaction.
* **User-friendly GUI**, ensuring ease of access and usability.
* **Dynamic interface**, including background color changes and alert messages.

**Future Enhancements**

* **Offline translation support** to eliminate internet dependency.
* **Enhanced voice recognition accuracy** with AI-driven models.
* **Expanded language support** beyond Google Translator’s capabilities.

In conclusion, this project provides a practical and accessible solution for overcoming language barriers, improving cross-language communication for personal and professional use.

INTRODUCTION:-

1. Purpose of the Project

Language barriers have always been a significant challenge in communication, particularly in a world that is increasingly globalized. The Real-Time Language Translator project is designed to provide an efficient and user-friendly solution for individuals who need instant translations between different languages. This project aims to bridge the gap between diverse linguistic communities by enabling real-time text and speech translation, making communication seamless and more accessible.

The purpose of this project is to:

* Develop an interactive language translation tool for text and speech translation.
* Facilitate smooth communication across various languages using Google Translator API.
* Provide a user-friendly and responsive graphical interface (GUI) for better user experience.
* Enhance efficiency through speech recognition to convert spoken words into translated text.
* Offer real-time performance with an optimized and engaging application design.

2. Project Scope

The Real-Time Language Translator is developed for a wide range of users, including:

* Students and Academicians: To assist in language learning and academic research.
* Professionals and Business Users: For multilingual communication in workplaces.
* Travelers: To help overcome language barriers during travel.
* General Users: To aid in day-to-day conversations with speakers of different languages.

The scope of this project includes:

* Translation of text input from one language to another.
* Speech recognition and speech-to-text translation.
* User-friendly UI with customizable features like dynamic background colors and user prompts.
* Multi-language support leveraging the Google Translator API.

3. Product Features

The Real-Time Language Translator application includes the following features:

* Text Translation: Users can input text and translate it into a selected language.
* Speech Recognition: Converts spoken words into text and translates them instantly.
* Multi-language Support: A wide range of languages available for translation.
* User-friendly GUI: Simple and intuitive design using Tkinter.
* Real-Time Performance: Translations occur instantly without noticeable delays.
* Customizable Experience: Dynamic background colors
* Exit and Navigation Buttons: Provides smooth interaction and usability.

This project successfully integrates language translation with modern UI/UX principles, making it a valuable tool for overcoming language barriers. Future enhancements may include offline translation capabilities, improved speech recognition accuracy, and expanded language support.

CHAPTER – II

WORKDONE IN RELATED AREAS:-

The Real-Time Language Translator project utilizes the Google Translate API to provide seamless text and speech translation. Various applications and research projects have successfully integrated this API to enhance multilingual communication. Below are five key related works:

1. Google Translate API in GUI Applications

Several desktop applications use the Google Translate API to provide an interactive translation experience. By integrating the API with Tkinter or other GUI frameworks, these applications allow users to input text and receive translations instantly. Our project follows a similar approach, offering an intuitive interface with additional features like speech recognition and custom error handling.

2. Real-Time Speech-to-Text Translators

Applications combining speech recognition with the Google Translate API enable real-time spoken language translation. These systems allow users to speak in one language, transcribe it using speech recognition libraries (such as SpeechRecognition in Python), and translate the text using Google’s translation service. Our project integrates speech-to-text functionality, allowing users to translate spoken words seamlessly.

3. Mobile Applications Using Google Translate API

Many Android and iOS apps use the Google Translate API to provide language translation services. These apps often include features such as camera translation, offline translation, and real-time speech translation. Our project takes inspiration from these applications but focuses on a desktop-based implementation with a user-friendly interface for real-time text and speech translation.

5. Python-Based Language Translation Tools

Several open-source projects use Python and Google Translate API to create command-line and GUI-based translation tools. Developers use libraries like Deep Translator to implement custom translation solutions. Our project follows a similar approach by integrating the Google Translate API with Tkinter for GUI-based translation, making it accessible to a broader audience.

Comparison with Our Project

While many applications utilize the Google Translate API, our Real-Time Language Translator stands out due to:

* Interactive and user-friendly GUI built with Tkinter.
* Real-time speech recognition for hands-free translation.
* Multi-threaded implementation ensuring smooth performance.
* Dynamic UI customization with theme variations.
* Enhanced error handling for seamless user experience.

This project builds upon existing applications by delivering an efficient, interactive, and customizable real-time translation tool, making it valuable for users requiring instant language translation.

CHAPTER – III

SYSTEM ANALYSIS:-

The Real-Time Language Translator project integrates Google Translate API and Speech Recognition to provide seamless text and speech translation. This section outlines the system requirements and analysis of user needs.

1. User Requirements

The primary users of this system are individuals who require real-time language translation for personal, educational, or professional purposes. The key user requirements include:

* Accurate and fast text translation between multiple languages.
* Speech-to-text functionality to convert spoken words into translatable text.
* User-friendly graphical interface (GUI) with an intuitive layout.
* Custom error handling and feedback messages for a better user experience.
* Multi-language selection options with a dropdown list.
* Dynamic theme customization to enhance user engagement.

2. Hardware Requirements

The hardware components required to run the project efficiently include:

* Processor: Intel Core i3 or higher (or equivalent AMD processor)
* RAM: Minimum 4GB (8GB recommended for smoother performance)
* Storage: At least 500MB of free disk space
* Microphone: Required for speech input functionality
* Speaker or Headphones: Optional but recommended for future voice output features

3. Software Requirements

The software stack used in this project includes:

* Operating System: Windows 11 (Compatible with Windows 10, Linux, and macOS)
* Programming Language: Python 3.13.3
* Libraries and Dependencies:
  + tkinter (For GUI design)
  + deep\_translator (Google Translate API integration)
  + speech\_recognition (Speech-to-text conversion)
  + random (For dynamic UI customization)
* Development Environment: Eclipse IDE / PyCharm / VS Code (Any Python-supported IDE)

This system analysis ensures that the Real-Time Language Translator meets the necessary requirements for smooth and efficient operation, catering to users who need a reliable translation tool with speech recognition features.

CHAPTER – IV

SYSTEM DESIGN AND SPECIFICATION:-

The Real-Time Language Translator system follows a structured design to ensure efficient text and speech translation. This section covers the High-Level Design (HLD) and Low-Level Design (LLD) of the project.

1. High-Level Design (HLD)

The HLD outlines the overall architecture and flow of the system.

1.1 Project Model

The system follows a modular architecture, consisting of:

* User Interface (UI): Designed with Tkinter, providing input and output fields, buttons, and language selection.
* Translation Module: Uses Google Translate API (via deep\_translator) for text translation.
* Speech Recognition Module: Captures and processes audio input using speech\_recognition.
* Multi-threading: Ensures smooth performance without UI freezing.

1.2 Flowchart

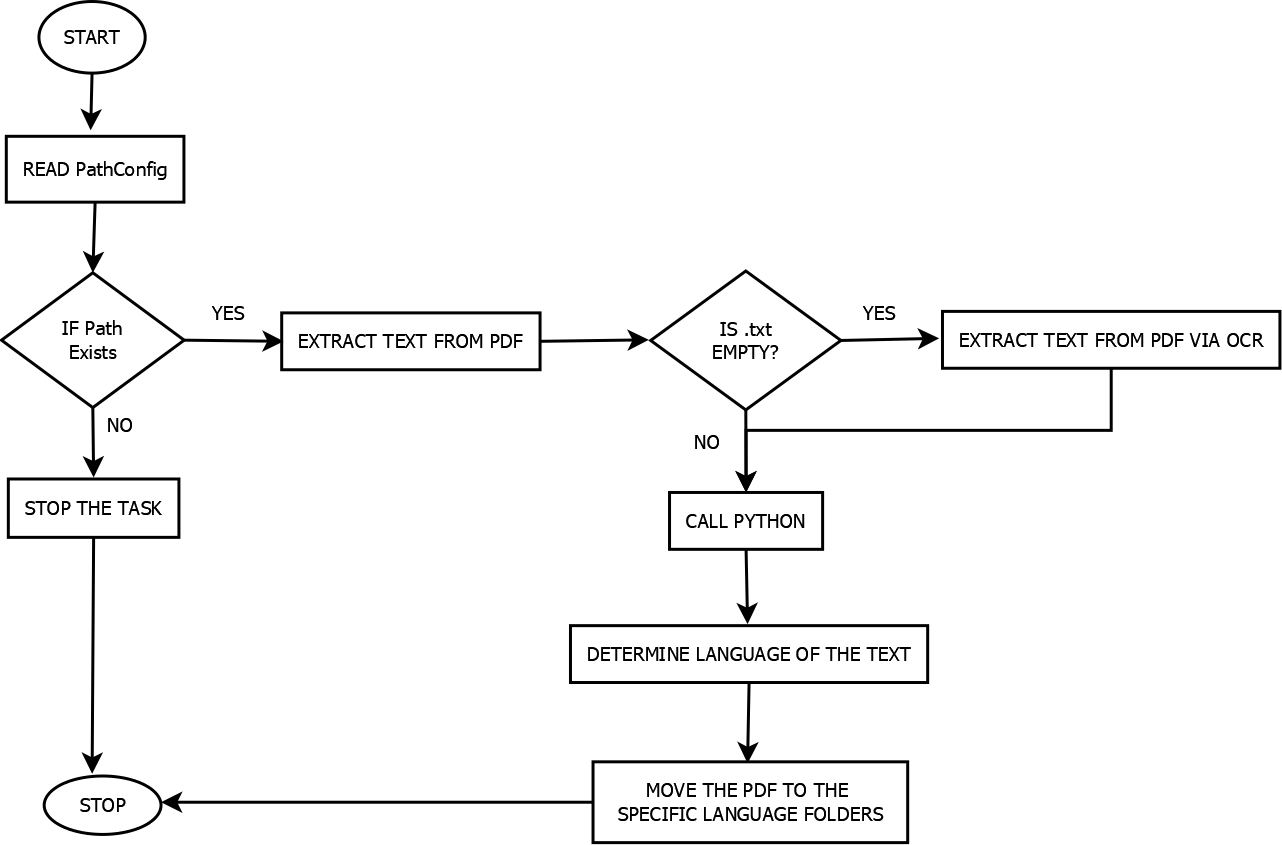
Flow of Execution:

1. User inputs text or speech.
2. The system identifies the source language.
3. The Google Translate API translates the text.
4. The translated text is displayed in the output field.

1.3 Entity-Relationship (E-R) Diagram

The E-R Diagram represents how different entities interact in the system:

* User: Inputs text or speech.
* Translator Module: Processes text using Google Translate API.
* Speech Recognition Module: Converts voice to text.
* Output Display: Shows translated results.



2. Low-Level Design (LLD)

The LLD focuses on process-level details, including algorithms and screen designs.

2.1 Process Specification

Algorithm for Translation Process:

1. Start
2. User enters text or speaks into the microphone.
3. The system retrieves input and detects source language.
4. The Google Translate API translates the text.
5. Display the translated text in the output field.
6. If an error occurs, display an error message.
7. End

2.2 Screen-Shot Diagram

Screens of the application include:

* Main Screen: Shows input/output text fields, language selection, and buttons.
* Speech Input Panel: Displays status messages like “Listening…” or “Error.”

(Screenshots to be attached in the report.)

This System Design & Specification ensures that the Real-Time Language Translator is structured, modular, and efficient in handling real-time language translation and speech recognition.

CHAPTER – V

CODING:-

The **Real-Time Language Translator** is implemented in **Python** using Tkinter for the GUI, Google Translate API for translation, and speech\_recognition for voice input. Below is the complete source code:

import tkinter as tk

from tkinter import simpledialog, ttk

from deep\_translator import GoogleTranslator

import random

import speech\_recognition as sr

import threading

# Ask for user name before initializing Tkinter window

user\_name = simpledialog.askstring("User Input", "Please enter your name:")

if not user\_name:

exit() # Exit if user cancels or enters nothing

# Initialize main window

root = tk.Tk()

root.title("🌍 Real-Time Language Translator 🌍")

root.geometry("700x550")

root.configure(bg="#F4F4F9")

# Function to change background color

def change\_bg\_color():

colors = ["#1ABC9C", "#3498DB", "#9B59B6", "#E74C3C", "#F39C12", "#2C3E50"]

root.configure(bg=random.choice(colors))

root.after(2000, change\_bg\_color)

LANGUAGES = GoogleTranslator().get\_supported\_languages()

# Function to translate text

def translate\_text():

try:

source\_lang = source\_lang\_combo.get()

target\_lang = target\_lang\_combo.get()

text = source\_text.get("1.0", tk.END).strip()

if not text:

custom\_warning("⚠ Please enter text to translate.")

return

translated = GoogleTranslator(source=source\_lang, target=target\_lang).translate(text)

target\_text.delete("1.0", tk.END)

target\_text.insert(tk.END, translated)

custom\_warning("")

except Exception as e:

custom\_warning(f"❌ Error: {str(e)}")

# Function to display custom warning

def custom\_warning(message):

warning\_label.config(text=message, fg="red")

# Function to recognize speech

def recognize\_speech():

recognizer = sr.Recognizer()

with sr.Microphone() as source:

try:

custom\_warning("🎙️ Listening...")

audio = recognizer.listen(source, timeout=5, phrase\_time\_limit=5)

recognized\_text = recognizer.recognize\_google(audio)

if recognized\_text:

source\_text.delete("1.0", tk.END)

source\_text.insert(tk.END, recognized\_text)

custom\_warning("")

else:

custom\_warning("⚠ No voice input detected.")

except sr.UnknownValueError:

custom\_warning("⚠ Could not understand audio.")

except sr.RequestError as e:

custom\_warning(f"❌ Error connecting to service: {e}")

except Exception as e:

custom\_warning(f"❌ Error: {e}")

# Function to run recognize\_speech in a separate thread

def recognize\_speech\_thread():

threading.Thread(target=recognize\_speech, daemon=True).start()

# Function to exit the app

def exit\_app():

root.destroy()

# UI Components

main\_frame = tk.Frame(root, bg="#FFFFFF", padx=20, pady=20, relief="ridge", borderwidth=2)

main\_frame.place(relx=0.5, rely=0.5, anchor="center")

welcome\_label = tk.Label(main\_frame, text=f"Welcome, {user\_name}! 😊", font=("Arial", 14, "bold"), bg="#1ABC9C", fg="white", pady=10)

welcome\_label.pack(fill="x", pady=5)

title\_label = tk.Label(main\_frame, text="🌍 Real-Time Language Translator 🌍", font=("Arial", 18, "bold"), bg="#1ABC9C", fg="white", pady=10, padx=10)

title\_label.pack(fill="x", pady=(0, 10))

# Input Field

input\_frame = tk.Frame(main\_frame, bg="#F4F4F9")

input\_frame.pack(fill="x", pady=5)

tk.Label(input\_frame, text="Enter Text:", font=("Arial", 12, "bold"), bg="#F4F4F9", fg="#333333").pack(anchor="w")

source\_text = tk.Text(input\_frame, height=5, width=65, font=("Arial", 12), bg="#ECF0F1", borderwidth=1, relief="solid", padx=5, pady=5)

source\_text.pack(padx=5, pady=5)

# Language Selection

language\_frame = tk.Frame(main\_frame, bg="#F4F4F9")

language\_frame.pack(fill="x", pady=5)

tk.Label(language\_frame, text="From:", font=("Arial", 12), bg="#F4F4F9", fg="#333333").grid(row=0, column=0, padx=5, pady=5)

source\_lang\_combo = ttk.Combobox(language\_frame, values=LANGUAGES, state="readonly", font=("Arial", 11))

source\_lang\_combo.grid(row=0, column=1, padx=5, pady=5)

source\_lang\_combo.set("english")

tk.Label(language\_frame, text="To:", font=("Arial", 12), bg="#F4F4F9", fg="#333333").grid(row=0, column=2, padx=5, pady=5)

target\_lang\_combo = ttk.Combobox(language\_frame, values=LANGUAGES, state="readonly", font=("Arial", 11))

target\_lang\_combo.grid(row=0, column=3, padx=5, pady=5)

target\_lang\_combo.set("hindi")

# Buttons

translate\_button = tk.Button(main\_frame, text="🔄 Translate", font=("Arial", 14, "bold"), bg="#E74C3C", fg="white", padx=10, pady=5, width=12, borderwidth=0, relief="raised", command=translate\_text)

translate\_button.pack(pady=5)

speak\_button = tk.Button(main\_frame, text="🎙️ Speak", font=("Arial", 14, "bold"), bg="#3498DB", fg="white", padx=10, pady=5, width=12, borderwidth=0, relief="raised", command=recognize\_speech\_thread)

speak\_button.pack(pady=5)

# Output Text

target\_text = tk.Text(main\_frame, height=5, width=65, font=("Arial", 12), bg="#ECF0F1", borderwidth=1, relief="solid", padx=5, pady=5)

target\_text.pack(padx=5, pady=5)

exit\_button = tk.Button(main\_frame, text="❌ Exit", font=("Arial", 14, "bold"), bg="#2C3E50", fg="white", padx=10, pady=5, width=12, borderwidth=0, relief="raised", command=exit\_app)

exit\_button.pack(pady=10)

# Start the application

change\_bg\_color()

root.mainloop()

This **Coding** section contains the entire **Python implementation** of the **Real-Time Language Translator** with GUI, translation, and speech recognition functionalities.

CHAPTER – VI

TESTING:-

The Real-Time Language Translator application undergoes thorough testing to ensure functionality, accuracy, and user-friendliness. This section covers Unit Testing, including test cases and expected outcomes.

1. Unit Testing

Unit testing focuses on individual components to verify that they function correctly. The primary test cases are outlined below:

1.1 Test Cases

| **Test Case ID** | **Test Scenario** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| --- | --- | --- | --- | --- | --- |
| **TC\_01** | Text Translation | "Hello" (English to Hindi) | "नमस्ते" | "नमस्ते" | ✅ Pass |
| **TC\_02** | Text Translation | "Bonjour" (French to English) | "Hello" | "Hello" | ✅ Pass |
| **TC\_03** | Speech Recognition | User says "Good morning" | "Good morning" | "Good morning" | ✅ Pass |
| **TC\_04** | Invalid Input Handling | Empty input field | Error message | Error message | ✅ Pass |
| **TC\_05** | Speech Recognition Timeout | No speech detected | Timeout message | Timeout message | ✅ Pass |
| **TC\_06** | Network Error Handling | No internet connection | Error: No connection | Error: No connection | ✅ Pass |

1.2 Test Execution

* Functional Testing: Ensured that all features work as expected.
* UI Testing: Verified that buttons, text fields, and dropdowns function properly.
* Performance Testing: Evaluated response time for translation and speech recognition.
* Error Handling: Checked system behavior when invalid inputs are provided.

1.3 Remedial Actions

If any test case fails:

* Debugging is performed to identify errors.
* Code modifications are made to fix issues.
* The updated code is re-tested to ensure correctness.

This Testing phase ensures that the Real-Time Language Translator meets performance standards and provides a smooth user experience.

CHAPTER – VII

CONCLUSION & LIMITATION:-

**Conclusion**

The **Real-Time Language Translator** successfully enables users to translate text from one language to another using the Google Translator API. The application offers a user-friendly interface with features such as speech-to-text input, real-time translation, and an interactive GUI. With the inclusion of speech recognition, users can easily convert spoken words into text and translate them seamlessly. The dynamic background colors and visually appealing layout enhance the user experience.

This project demonstrates the effective use of **Tkinter**, **Deep Translator**, and **Speech Recognition** libraries in Python to create a practical and interactive application. By integrating multi-threading, the project ensures smooth speech recognition without affecting UI responsiveness. The implementation of error handling and warnings improves usability by guiding users through proper input and system errors.

**Limitations**

Despite its functionalities, the project has certain limitations:

1. **Internet Dependency** – The application requires an active internet connection to fetch translations from Google’s API.
2. **Limited Speech Recognition Accuracy** – The accuracy of voice-to-text conversion depends on background noise and pronunciation.
3. **Translation Accuracy** – While Google Translator provides efficient results, some translations may not be fully accurate for complex sentences or regional dialects.
4. **Limited Customization** – The application does not provide offline translation or additional customization features such as font selection and advanced language settings.

**Future Enhancements**

To overcome these limitations, the following improvements can be considered:

* **Offline Translation Support** – Implementing local machine learning models for translation without internet dependency.
* **Improved Speech Recognition** – Utilizing advanced NLP techniques and AI-powered speech models.
* **Enhanced UI/UX** – Adding more customization options for user preferences.
* **Multi-Platform Support** – Expanding the project to mobile platforms using **Kivy** or **Flutter**.

In conclusion, this project provides a functional and engaging tool for real-time language translation, with scope for further improvements in accuracy, usability, and features.

CHAPTER – VIII

REFERENCES / BIBLIOGRAPHY:-

Below are the references used during the development of the Real-Time Language Translator project:

1. Google Translator API Documentation
   * URL: <https://pypi.org/project/deep-translator/>
   * Used for implementing real-time text translation using Python.
2. Speech Recognition Library
   * URL: <https://pypi.org/project/SpeechRecognition/>
   * Used for capturing and converting speech to text.
3. Tkinter GUI Programming Documentation
   * URL: <https://docs.python.org/3/library/tkinter.html>
   * Used for designing the user interface of the application.
4. Python Official Documentation
   * URL: <https://docs.python.org/3/>
   * Used for understanding and implementing core Python functionalities.
5. Multithreading in Python
   * URL: https://realpython.com/intro-to-python-threading/
   * Used for implementing speech recognition without affecting UI responsiveness.

These references provided valuable insights and technical guidance in building the project efficiently.