A
Learning Project-II Report
On

# "Real Time Language Translator"

Submitted in partial fulfillment of
The requirements for the 4th Semester Sessional
Examination of

BACHELOR OF TECHNOLOGY
*IN*
COMPUTER SCIENCE & ENGINEERING

By
**ANSUMAN MAHAPATRA (23UG010605)**
**BISWAJEET PATRA (23UG010614)**
**TAPAN KUMAR SAHOO(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 "Real Time Language Translator" is done by Ansuman Mahapatra (23UG010605), Biswajeet Patra (23UG010614), and Tapan Kumar Sahoo (23UG010632) in partial fulfillment of the requirements for the 4th Semester Sessional Examination of Bachelor of Technology in Computer Science and Engineering during the academic year 2024-25. This work is submitted to the department as a part of evaluation of 4th Semester Learning Project-II.*

*Proctors/Class Teacher* *Project Coordinator,2nd Year*

*HoD, CSE,* 2nd Year

# ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our esteemed Proctor/Class Teacher, Mr. Himansu 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 (23UG010614)
Tapan Kumar Sahoo (23UG010632)

# 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, a splash screen, dynamic visual effects, and an integrated admin login system with MySQL support, offering a robust and interactive translation tool.

## Objectives:

The primary objectives of this project are:

a) To create a user-friendly and efficient real-time translation tool.
b) To integrate Google Translator API for accurate translations.
c) To implement speech recognition for hands-free text input.
d) To develop an admin panel for user activity monitoring using MySQL.
e) To ensure a responsive and interactive graphical interface using Tkinter, including splash screen and dynamic visuals.

## Methodology:

The project follows a structured approach:

a) User Input Handling: Users can enter text manually or use speech-to-text for automatic input.
b) Language Selection: A dropdown menu allows selection of source and target languages.
c) Translation Process: The Google Translator API is used to fetch and display real-time translations.
d) Speech Recognition: Implemented using the SpeechRecognition library to convert spoken words into text.
e) Login Tracking: User logins are recorded with timestamps using a MySQL database.
f) Admin Panel: Provides login access for administrators to view user login activity via a dashboard.

g) Graphical User Interface: Designed with Tkinter for an intuitive and dynamic experience, including a splash screen animation.
h) Multithreading: Ensures smooth performance by running speech recognition on a separate thread to prevent UI lag.
i) UI Management: Dynamic background color and footer animations are handled using after() methods and are safely canceled on exit to avoid memory errors.

## Outcomes:

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

a) Accurate text translation for multiple languages.
b) Voice-to-text input, allowing hands-free interaction.
c) Integrated admin panel with credential verification and real-time user login logs.
d) Splash screen on startup for a polished launch experience.
e) Dynamic background color cycling and footer animations to enhance visual engagement.
f) Error-free exit handling by properly canceling after() callbacks.
g) Future Enhancements
h) Offline translation support to eliminate internet dependency.
i) Enhanced voice recognition accuracy with AI-driven models.
j) Expanded language support beyond Google Translator’s capabilities.
k) Advanced admin analytics with usage statistics and export options.

## CONTENTS PAGE NO.

CHAPTER – I

# 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, supported by admin-level insights and interactive visual features.

The purpose of this project is to:

a) Develop an interactive language translation tool for text and speech translation.
b) Facilitate smooth communication across various languages using Google Translator API.
c) Provide a user-friendly and responsive graphical interface (GUI) with splash animation and dynamic visuals.
d) Enhance efficiency through speech recognition to convert spoken words into translated text.
e) Log user activity into a MySQL database and provide admin-level login with access to usage reports.

## 2. Project Scope :

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

a) Students and Academicians: To assist in language learning and academic research.
b) Professionals and Business Users: For multilingual communication in workplaces.
c) Travelers: To help overcome language barriers during travel.
d) General Users: To aid in day-to-day conversations with speakers of different languages.
e) Administrators: To monitor usage logs and ensure secure login and access control.

**Scope includes:**

a) Translation of text input from one language to another.
b) Speech recognition and speech-to-text translation.

c) User-friendly UI with customizable features like dynamic background colors and user prompts.
d) Multi-language support leveraging the Google Translator API.
e) Admin dashboard for reviewing user login records stored in a MySQL database.
f) Splash screen on launch and smooth termination with safe cleanup of background tasks.

## 3. Product Features:

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

a) Text Translation: Users can input text and translate it into a selected language.
b) Speech Recognition: Converts spoken words into text and translates them instantly.
c) Multi-language Support: A wide range of languages available for translation.
d) User-friendly GUI: Simple and intuitive design using Tkinter.
e) Real-Time Performance: Translations occur instantly without noticeable delays.
f) Admin Login and Dashboard: Restricted access for administrators to monitor login activity.
g) Customizable Experience: Includes splash animation, dynamic background color transitions, and real-time footer updates.
h) Safe Exit Mechanism: Prevents after() callback errors by cancelling all loops before app termination.

This project successfully combines language translation with backend tracking and enhanced UI principles, making it a comprehensive, efficient, and professional-grade application.

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.

## 6.Comparison with Our Project

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

a) Interactive and user-friendly GUI built with Tkinter.

b) Real-time speech recognition for hands-free translation.

c) Multi-threaded implementation ensuring smooth performance.

d) Dynamic UI customization with theme variations.

e) 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, now enhanced with administrative features and improved interface handling.

## 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:

a) Accurate and fast text translation between multiple languages.
b) Speech-to-text functionality to convert spoken words into translatable text.
c) User-friendly graphical interface (GUI) with an intuitive layout.
d) Custom error handling and feedback messages for a better user experience.
e) Multi-language selection options with a dropdown list.
f) Dynamic theme customization to enhance user engagement.
g) Secure admin login to access user login activity and translation logs.
h) Initial splash screen to improve user onboarding experience.
i) Smooth exit handling with automatic cancellation of background UI updates.

## 2. Hardware Requirements

The hardware components required to run the project efficiently include:

a) Processor: Intel Core i3 or higher (or equivalent AMD processor)
b) RAM: Minimum 4GB (8GB recommended for smoother performance)
c) Storage: At least 500MB of free disk space
d) Microphone: Required for speech input functionality
e) 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:**

a) tkinter (For GUI design)

b) deep_translator (Google Translate API integration)

c) speech_recognition (Speech-to-text conversion)

d) random (For dynamic UI customization)

e) mysql.connector (For MySQL database integration and admin login dashboard)

**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. With the addition of admin-level access, splash screen interaction, and properly managed background tasks, the system is now even more robust and tailored to meet modern translation needs efficiently.

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, now enhanced with secure admin access, splash screen integration, and stable UI lifecycle management.

## 1. High-Level Design

### 1.1 Project Model

The system follows a modular architecture, consisting of:

a) User Interface (UI): Designed with Tkinter, providing input/output text fields, speech input status, admin access, and interactive language selection.

b) Translation Module: Uses Google Translate API (via deep_translator) for real-time text translation.

c) Speech Recognition Module: Captures and processes audio input using the speech_recognition library.

d) Admin Module: Provides secure login access for administrators and fetches login records from a MySQL database.

e) Splash Screen: Displays an animated introduction at app launch to improve user experience.

f) Multi-threading: Ensures smooth performance without UI freezing during speech processing.

g) UI Timer Management: Handles background color transitions and footer animations using after() calls, which are safely canceled on exit.

### 1.2 Flowchart

Flow of Execution:

a) User inputs text or speech.

b) The system identifies the source language.

c) The Google Translate API translates the text.

d) The translated text is displayed in the output field.

e) User (optionally) logs in as Admin to access usage data.

### 1.3 Entity-Relationship (E-R) Diagram

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

a) User: Inputs text or speech.

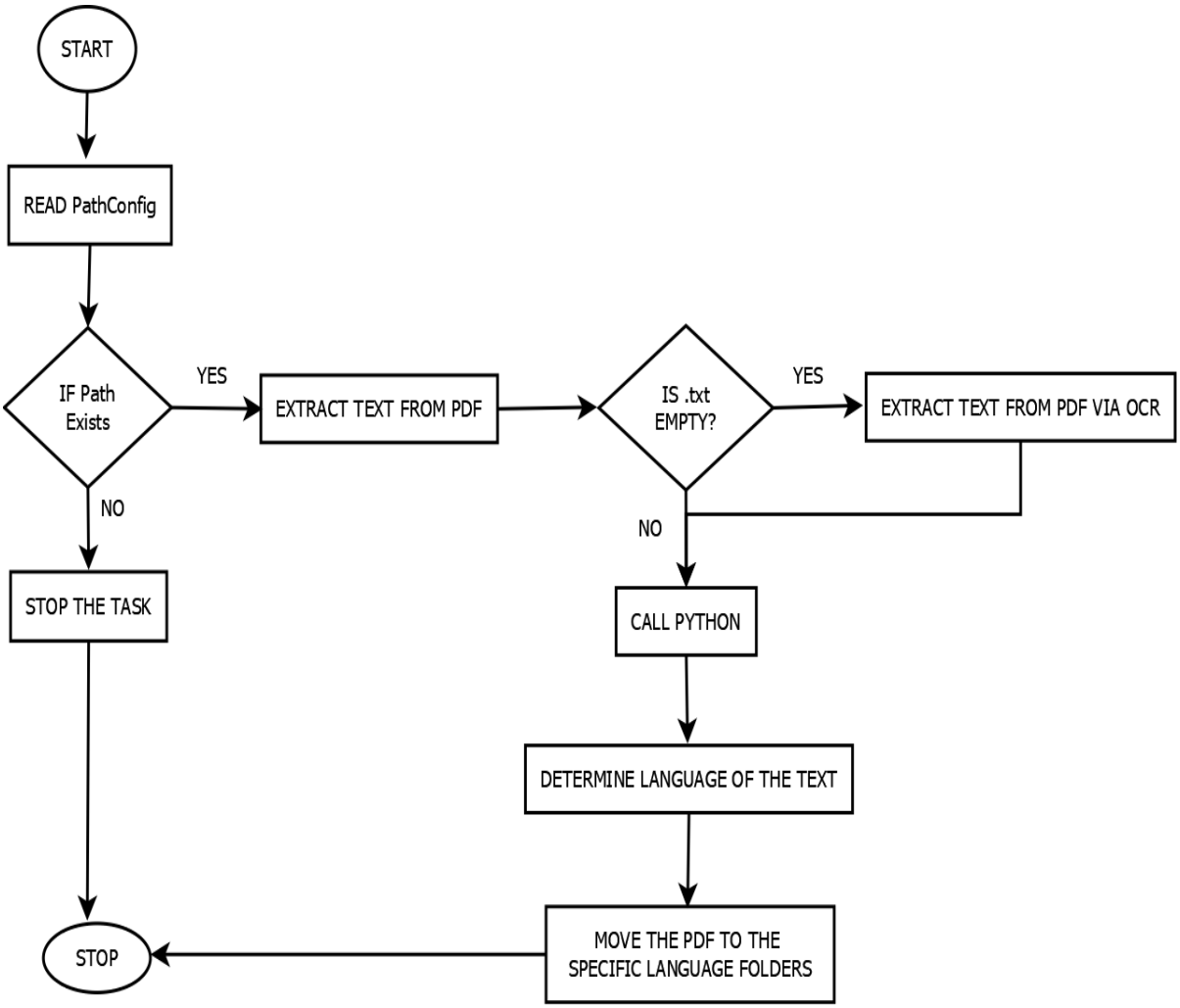
b) Translator Module: Processes text using Google Translate API.

c) Speech Recognition Module: Converts voice to text.

d) Admin: Logs in securely and monitors login data via MySQL.

e) Login Info Table: Stores username and timestamp for each session.

f) 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

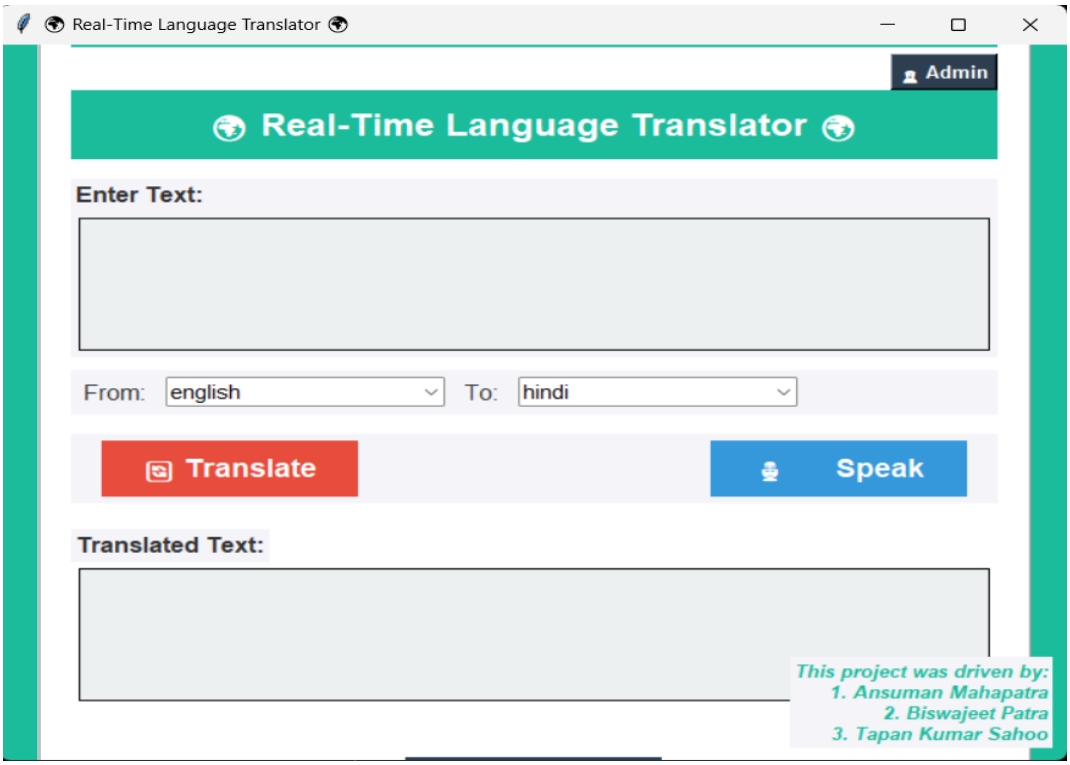
Algorithm for Translation and Admin Flow:

a) Start
b) User enters text or speaks into the microphone
c) The system retrieves input and detects source language
d) Google Translate API translates the text
e) Display the translated text in the output field
f) If Admin option is used, verify credentials and display login records
g) On exit, cancel active UI callbacks (after() methods) to prevent errors
h) End

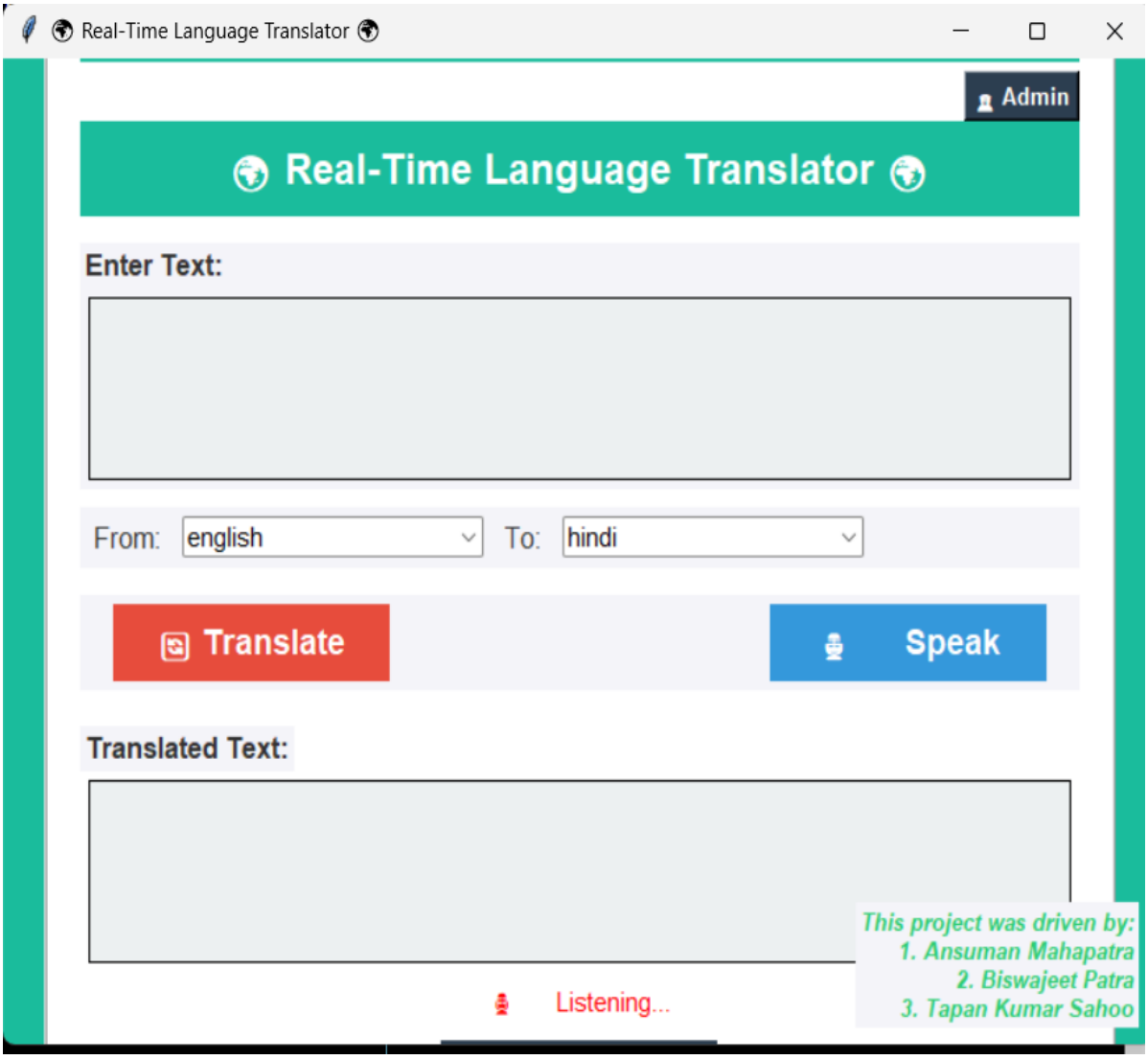
### 2.2 Screen-Shot Diagram

Screens of the application include:

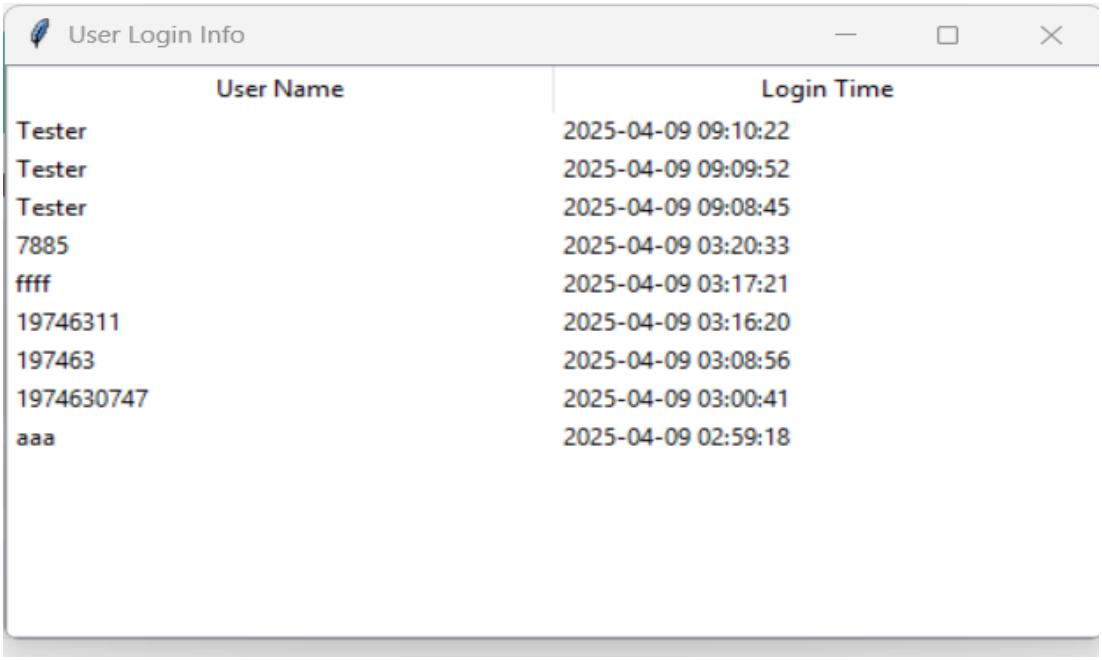
a) **Main Screen**: Input/output text fields, language dropdowns, speech and translate buttons, and admin access icon.



b) **Speech Input Panel**: Displays real-time status like “Listening…” or “Could not understand audio.”



c) **Admin Dashboard**: Displays a table view of user login history with timestamps fetched from the database.




d) **Splash Screen**: A 3-second animated intro screen before the main interface loads.

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, messagebox
from deep_translator import GoogleTranslator
import random
import speech_recognition as sr
import threading
from PIL import Image, ImageTk, ImageSequence
import mysql.connector
from datetime import datetime

# ----------------------- Database Configuration ----------------------- #
DB_CONFIG = {
    'user': 'root',
    'password': '0747',
    'host': '127.0.0.1',
    'port': 3306,
    'database': 'PythonTranslator'
}

# Global variables to manage scheduled tasks
bg_color_job = None
footer_style_job = None

# ----------------------- Root Window Setup ----------------------- #
root = tk.Tk()
```

```
root.withdraw()  # Hide the main window initially (will be shown after splash screen)

# ---------------------- Prompt User for Name ---------------------- #
user_name = simpledialog.askstring("User Input", "Please enter your name:")
if not user_name:
    exit()

# ---------------------- Log User Login Info into MySQL ---------------------- #
try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor()
    login_time = datetime.now()
    cursor.execute("INSERT INTO login_info (username, login_time) VALUES (%s, %s)",
(user_name, login_time))
    conn.commit()
    cursor.close()
    conn.close()
except Exception as e:
    print("Database Error:", e)

# ---------------------- Splash Screen Function ---------------------- #
def show_splash():
    splash_root = tk.Toplevel()
    splash_root.overrideredirect(True)
    splash_root.geometry("600x400+400+200")
    canvas = tk.Canvas(splash_root, width=600, height=400, highlightthickness=0)
    canvas.pack()

    try:
        gif = Image.open("splash.gif")
```

```
        frames = [ImageTk.PhotoImage(frame.copy().resize((600, 400))) for frame in ImageSequence.Iterator(gif)]
        def update_frame(i):
            canvas.create_image(0, 0, anchor=tk.NW, image=frames[i])
            splash_root.after(100, update_frame, (i + 1) % len(frames))
        update_frame(0)
        splash_root.after(3000, lambda: [splash_root.destroy(), root.deiconify()])
        splash_root.mainloop()
    except FileNotFoundError:
        splash_root.destroy()
        root.deiconify()
        messagebox.showwarning("Splash Missing", "splash.gif not found. Skipping splash screen.")

# ---------------------- Main GUI Setup ---------------------- #
root.title("\U0001F30D Real-Time Language Translator \U0001F30D")
root.geometry("700x550")
root.configure(bg="#F4F4F9")

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

def change_bg_color():
    global bg_color_job
    root.configure(bg=random.choice(colors))
    bg_color_job = root.after(2000, change_bg_color)

# ---------------------- Supported Languages ---------------------- #
LANGUAGES = GoogleTranslator().get_supported_languages()

# ---------------------- Translation Function ---------------------- #
```

```
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("\u26A0 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"\u274C Error: {str(e)}")

# ------------------------ Display Warning Messages ------------------------ #
def custom_warning(message):
    warning_label.config(text=message, fg="red")

# ------------------------ Speech Recognition ------------------------ #
def recognize_speech():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        try:
            custom_warning("\U0001F399\uFE0F 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("\u26A0 No voice input detected.")
    except sr.UnknownValueError:
        custom_warning("\u26A0 Could not understand audio.")
    except sr.RequestError as e:
        custom_warning(f"\u274C Error connecting to service: {e}")
    except Exception as e:
        custom_warning(f"\u274C Error: {e}")

# Run speech recognition on a separate thread

def recognize_speech_thread():
    threading.Thread(target=recognize_speech, daemon=True).start()

# ------------------------ Exit Application Safely ------------------------ #
def exit_app():
    if bg_color_job:
        root.after_cancel(bg_color_job)
    if footer_style_job:
        root.after_cancel(footer_style_job)
    root.destroy()

# ------------------------ Admin Login ------------------------ #
def open_admin_login():
    login_win = tk.Toplevel(root)
    login_win.title("Admin Login")
    login_win.geometry("300x200")

    tk.Label(login_win, text="Admin Username").pack()
    username_entry = tk.Entry(login_win)
```

```
    username_entry.pack()

    tk.Label(login_win, text="Password").pack()
    password_entry = tk.Entry(login_win, show="*")
    password_entry.pack()

    def verify_login():
        user = username_entry.get()
        passwd = password_entry.get()

        try:
            conn = mysql.connector.connect(**DB_CONFIG)
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM admin WHERE username=%s AND
password=%s", (user, passwd))
            result = cursor.fetchone()
            if result:
                show_admin_table()
                login_win.destroy()
            else:
                messagebox.showerror("Error", "Invalid Credentials")
            cursor.close()
            conn.close()
        except Exception as e:
            messagebox.showerror("DB Error", str(e))

    tk.Button(login_win, text="Login", command=verify_login).pack(pady=10)

# ---------------------- Admin Dashboard ---------------------- #
def show_admin_table():
    admin_win = tk.Toplevel(root)
```

```
    admin_win.title("User Login Info")
    admin_win.geometry("500x300")

    tree = ttk.Treeview(admin_win, columns=("User", "Time"), show="headings")
    tree.heading("User", text="User Name")
    tree.heading("Time", text="Login Time")
    tree.pack(fill="both", expand=True)

    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor()
        cursor.execute("SELECT username, login_time FROM login_info ORDER BY login_time DESC")
        for row in cursor.fetchall():
            tree.insert('', 'end', values=row)
        cursor.close()
        conn.close()
    except Exception as e:
        messagebox.showerror("Error", str(e))

# ---------- Main 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}! \U0001F60A", font=("Arial", 14, "bold"), bg="#1ABC9C", fg="white", pady=10)
welcome_label.pack(fill="x", pady=5)

admin_icon = tk.Button(main_frame, text="\U0001F46E Admin", font=("Arial", 10, "bold"), bg="#2C3E50", fg="white", command=open_admin_login)
```

```
admin_icon.pack(anchor="ne")

title_label = tk.Label(main_frame, text="\U0001F30D Real-Time Language Translator \U0001F30D", font=("Arial", 18, "bold"), bg="#1ABC9C", fg="white", pady=10, padx=10)
title_label.pack(fill="x", pady=(0, 10))

# Text input area
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 for Translate and Speak
button_frame = tk.Frame(main_frame, bg="#F4F4F9")
button_frame.pack(fill="x", pady=10)

translate_button = tk.Button(button_frame, text="\U0001F504 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(side="left", padx=20, pady=5)

speak_button = tk.Button(button_frame, text="\U0001F399\uFE0F 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(side="right", padx=20, pady=5)

# Output display
tk.Label(main_frame, text="Translated Text:", font=("Arial", 12, "bold"), bg="#F4F4F9", fg="#333333").pack(anchor="w", pady=(10, 0))
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)

# Warning or status messages
tk.Label(main_frame, text="", font=("Arial", 11), fg="red", bg="#FFFFFF")
warning_label = tk.Label(main_frame, text="", font=("Arial", 11), fg="red", bg="#FFFFFF")
warning_label.pack(pady=(5, 0))
```

```
# Exit button
exit_button = tk.Button(main_frame, text="\u274C 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)

# ---------------------- Dynamic Footer Styling ---------------------- #
def change_footer_style():
    global footer_style_job
    colors = ["#E74C3C", "#3498DB", "#2ECC71", "#F39C12", "#9B59B6", "#1ABC9C", "#D35400"]
    styles = ["bold", "italic", "underline", "bold italic", "italic underline"]
    color = random.choice(colors)
    font_style = random.choice(styles)
    footer_label.config(fg=color, font=("Arial", 10, font_style))
    footer_style_job = root.after(2000, change_footer_style)

# Footer text
footer_label = tk.Label(
    root,
    text="This project was driven by:\n1. Ansuman Mahapatra\n2. Biswajeet Patra\n3. Tapan Kumar Sahoo",
    font=("Arial", 10, "italic"),
    bg="#F4F4F9",
    fg="#555555",
    justify="right"
)
footer_label.place(relx=1.0, rely=1.0, anchor="se", x=-10, y=-10)

# ---------------------- Start Dynamic Events ---------------------- #
```

```
change_footer_style()
change_bg_color()
show_splash()
root.mainloop()
```

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

a) Functional Testing: Ensured that all features including translation, speech recognition, splash screen, and admin login perform as expected.

b) UI Testing: Verified that buttons, text fields, dropdowns, and the admin login interface work smoothly.

c) Performance Testing: Evaluated response times for translation, login logging to MySQL, and speech recognition.
d) Error Handling: Checked how the application responds to invalid inputs, speech recognition timeouts, and network failures.
e) Database Interaction: Confirmed user login info is correctly recorded in the MySQL database and can be retrieved by the admin.

### 1.3 Remedial Actions

If any test case fails:

a) Debugging is performed to identify the source of the issue.
b) Required code modifications are applied to resolve errors.
c) The corrected features are re-tested to ensure proper functionality.

This Testing phase ensures that the Real-Time Language Translator meets functionality and performance standards, provides a smooth user experience, and handles both visual and backend operations effectively.

CHAPTER – VII

# CONCLUSION & LIMITATION

## 1. 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.

Additionally, the system integrates a secure Admin Login mechanism with MySQL-based user logging, allowing administrators to monitor login activity via a built-in dashboard. A splash screen adds professionalism at startup, and scheduled UI updates like animations are properly terminated on exit, ensuring clean program shutdown without runtime warnings.

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.

## 2. Limitations

Despite its extensive functionalities, the project has certain limitations:

a) Internet Dependency – The application requires an active internet connection to fetch translations from Google's API, making it unusable offline.
b) Limited Speech Recognition Accuracy – The accuracy of the voice-to-text conversion is affected by background noise, user accent, and pronunciation clarity.
c) Translation Accuracy – While Google Translator offers reliable translations, complex sentences, idioms, and regional dialects may not always translate accurately.
d) Limited Customization – Users cannot yet customize interface settings such as font style, font size, color themes, or layout preferences.

e) Admin Features Are Basic – The admin dashboard is limited to viewing login logs, lacking features such as user management, activity tracking, or data export capabilities.
f) No User Authentication System – Besides the admin login, there is no user authentication or profile system, limiting personalized experiences and security for regular users.
g) Lack of Offline Mode – The application does not currently support offline translation or voice recognition, which restricts usage in low-connectivity areas.
h) No Language Auto-Detection – Users must manually select source and target languages, as the app does not automatically detect spoken or written language.
i) Basic Error Handling – The application has limited error reporting and handling for failed API requests, misrecognitions, or incorrect inputs.
j) Platform Limitation – The project is designed for desktop environments and may not be responsive or compatible with mobile or tablet interfaces.

CHAPTER – VIII

# REFERENCES / BIBLIOGRAPHY

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


**a) Google Translator API Documentation**

1. URL: https://pypi.org/project/deep-translator/
2. Used for implementing real-time text translation using Python's deep_translator module.

b) **Speech Recognition Library**

1. URL: https://pypi.org/project/SpeechRecognition/
2. Used for capturing and converting speech to text via microphone input.

**c) Tkinter GUI Programming Documentation**

1. URL: https://docs.python.org/3/library/tkinter.html
2. Used for designing the graphical user interface, handling widgets, and managing layout and events.

d) **Python Official Documentation**

1. URL: https://docs.python.org/3/
2. Used for understanding and implementing core functionalities such as file handling, threading, and exception management.

e) **Multithreading in Python**

1. URL: https://realpython.com/intro-to-python-threading/
2. Used to run speech recognition in a separate thread, ensuring that the UI remains responsive.

**f) MySQL Connector for Python**

1. URL: https://pypi.org/project/mysql-connector-python/
2. Used to implement user login logging and enable admin dashboard access through a MySQL database.

g) **Pillow (PIL) for Image Handling**

1. URL: https://pypi.org/project/Pillow/
2. Used for displaying the animated splash screen at application startup using .gif rendering.