APPLIED MACHINE LEARNING FOR TEXT ANALYSIS PROJECT REPORT

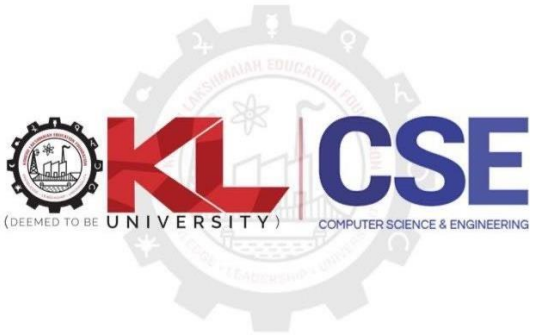
Submitted in partial fulfilment of the requirements for

the course in

BACHELOR OF TECHNOLOGY

SUBMITTED BY

BATTULA SAI ANUSHKA	**(2300030077)**
ERLE CHANDRA HARSHA	**(2300030182)**
VUBBARA HARSHALA REDDY	**(2300033165)**
JEENEPALLY ADISESHU	**(2300030279)**



Department of Computer Science and Engineering

KONERU LAKSHMAIAH EDUCATION FOUNDATION
Green Fields, Vaddeswaram.

ACKNOWLEDGEMENTS

We would like to express my sincere gratitude to **Dr. Malladi Ravisankar, Professor**, Department of Computer Science and Engineering (CSE-2), **KL Deemed to be University**, for his valuable guidance, encouragement, and continuous support throughout the completion of my **Applied Machine Learning for Text Analysis Project Report** titled "LANGUAGE DETECTION".

His insightful suggestions and constant motivation greatly contributed to the successful completion of this project.

We would also like to thank my university, **KL Deemed to be University**, for providing a conducive learning environment and encouraging students to undertake practical and research-oriented projects that enhance technical knowledge and hands-on skills in emerging technologies like **Machine Learning** and **Artificial Intelligence**.

CERTIFICATION

This is to certify that this is a bonafide record of the Lab Experiments / Project work done by **BATTULA SAI ANUSHKA-2300030077 , ERLE CHANDRA HARSHA-2300030182, JEENEPALLY ADISESHU-2300030279,VUBBARA HARSHALA REDDY-2300030165** submitted to **Dr. Malladi Ravisankar** in partial fulfilment of the requirements for the award of the degree **Bachelor of Technology** in the branch CSE during the academic year 2025-2026.

Signature of Supervisor

INDEX

ABSTRACT

Language detection is an essential task in today's multilingual digital world, where content is generated in multiple languages across different platforms. The Language Detection System is a Python-based project designed to automatically identify the language of a given text. It supports several languages, including English, Hindi, Telugu, and others, making it useful for applications such as translation, multilingual messaging, content categorization, and educational tools.

The system works by analyzing the text at two levels. First, it examines the Unicode script of the characters to detect languages like Telugu or Hindi, which have distinct scripts. Second, for languages that share scripts, such as English and other Latin-based languages, it uses statistical language models provided by Python libraries like langdetect. This combination of script analysis and statistical modeling improves the accuracy of detection, even for short texts or texts containing proper nouns.

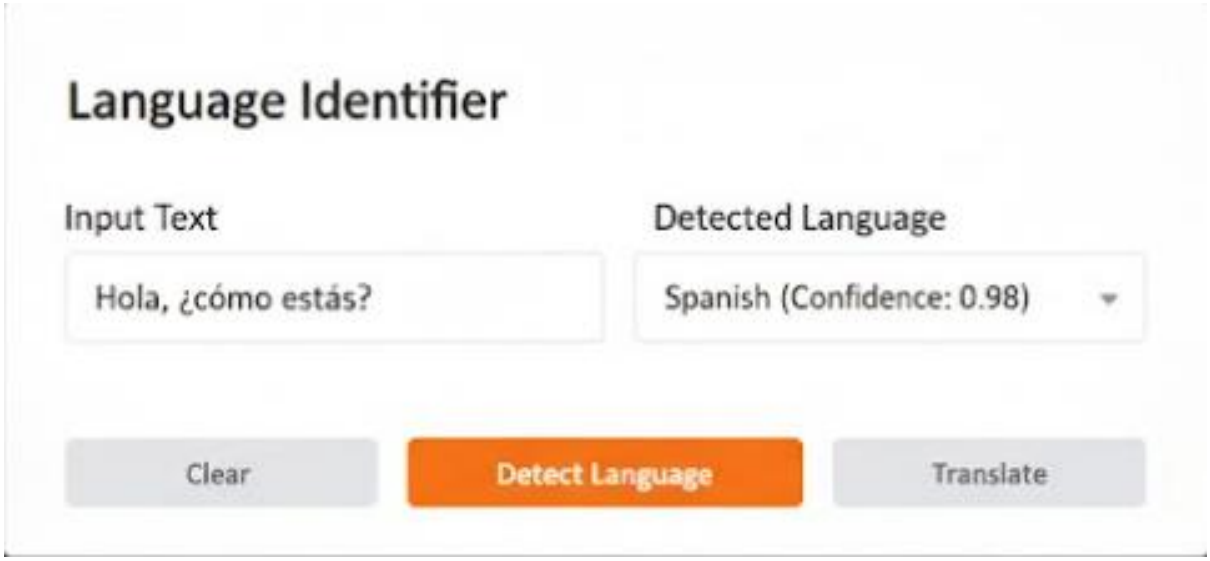
Once the language is detected, the system converts the language code into a full, human-readable name using the langcodes library, ensuring that the output is understandable to users. The system is capable of processing both single sentences and longer paragraphs efficiently, making it versatile for multiple scenarios.

The project demonstrates the practical application of natural language processing techniques in solving real-world problems related to text analysis. It provides a simple and interactive way to identify the language of any input text and can be easily extended to include additional languages in the future. Overall, the Language Detection System is a reliable, user-friendly, and efficient tool for multilingual text processing.

LANGUAGE DETECTION

1. Introduction

The Language Detection System is a Python-based project that automatically identifies the language of a given text. It supports multiple languages such as English, Hindi, Telugu, and more. The system works by analyzing the text using Unicode scripts and statistical language detection techniques. It provides quick and accurate detection for both short phrases and long paragraphs. The detected language is displayed in a readable format, making it easy for users to understand. This project is useful for applications like translation, multilingual communication, and content analysis, and can be extended to support additional languages in the future.



2. Software Development Life Cycle (SDLC)

2.1 Introduction to SDLC

The Software Development Life Cycle (SDLC) provides a structured framework for planning, developing, testing, and deploying the Language Detection System efficiently. The project follows the **Waterfall Model** approach, ensuring clarity and sequential progress through each phase.

2.2 SDLC Phases Applied to the Project

a) Requirement Analysis

In this phase, the main goal was identified — to detect the language of any given text and provide readability suggestions. The system requirements included:

- Python as the development language
- Libraries such as langdetect, langcodes, and re
- Compatibility with multiple languages (e.g., English, Telugu, Hindi, Tamil, etc.)

b) System Design

The system was designed to take user input, process it using the language detection algorithm, and display results along with readability analysis. The design includes:

- Input section for entering text
- Language detection module
- Output section showing detected language, ISO code, and suggestions

c) Implementation

Implementation was done using Python. The langdetect library identifies the probable language of the text, while the langcodes library converts the language code into a readable name. Additional logic provides feedback on sentence length and structure.

d) Testing

Testing involved verifying the system's accuracy across multiple languages and checking for stability with various input lengths and special characters. Unit tests were conducted for each component.Manual and automated test cases were created to verify the accuracy of similarity outputs. Bugs identified during testing were fixed iteratively.

e) Deployment

The final project was deployed in a simple Python environment and optionally integrated into a GUI (e.g., Gradio or Tkinter) for user-friendly interaction.

f) Maintenance

Future updates may include:

- Enhancing detection accuracy
- Supporting more languages
- Adding speech-to-text integration for spoken language detection

2.3 Advantages of Using SDLC

1. Provides a structured and organized development process.
2. Helps in early detection and correction of errors.
3. Ensures better quality and reliability of the project.
4. Maintains clear documentation for future updates.
5. Makes project maintenance and enhancement easier.

3. System Architecture and Design

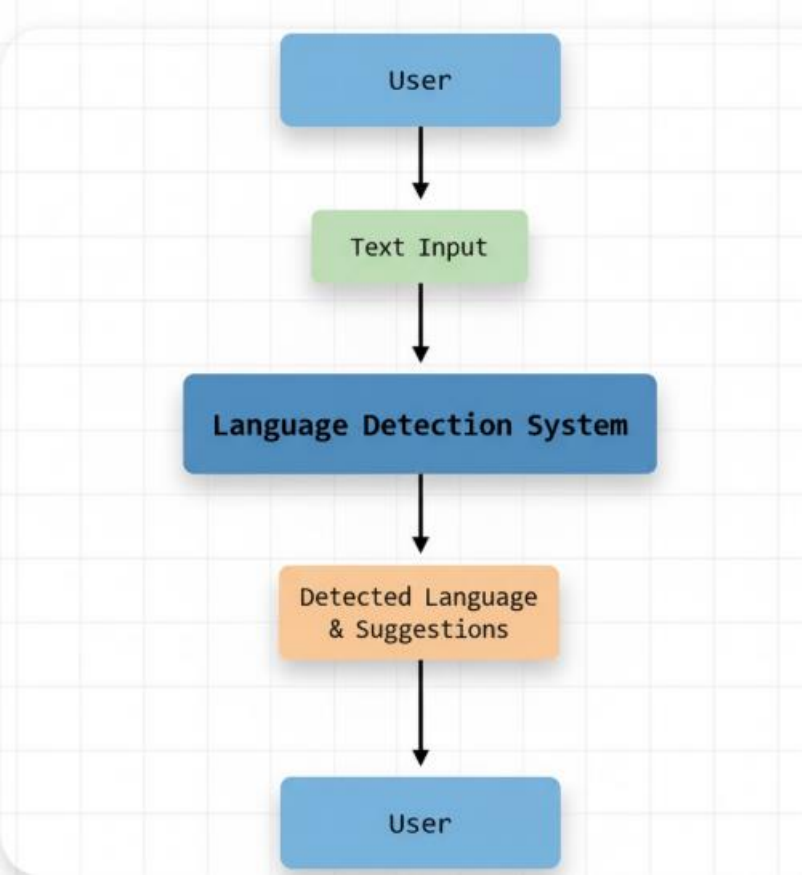
3.1 System Architecture Overview

1. **Input Layer:** Accepts user text in any language.
2. **Detection Module:** Uses langdetect to identify the language code.
3. **Conversion Module:** Converts ISO code to a readable language name using langcodes.
4. **Readability Module:** Analyzes word count, sentence length, and punctuation.
5. **Output Layer:** Displays detected language, confidence level, and suggestions.

3.2 System Architecture Diagram

3.3 Data Flow Diagram (DFD)

User → Input → Detection → Conversion → Readability → Output



3.4 Use Case Diagram

1. **User** – The person who inputs text to detect the language.
2. **System** – The Language Detection system that processes input and returns results.

Relations:

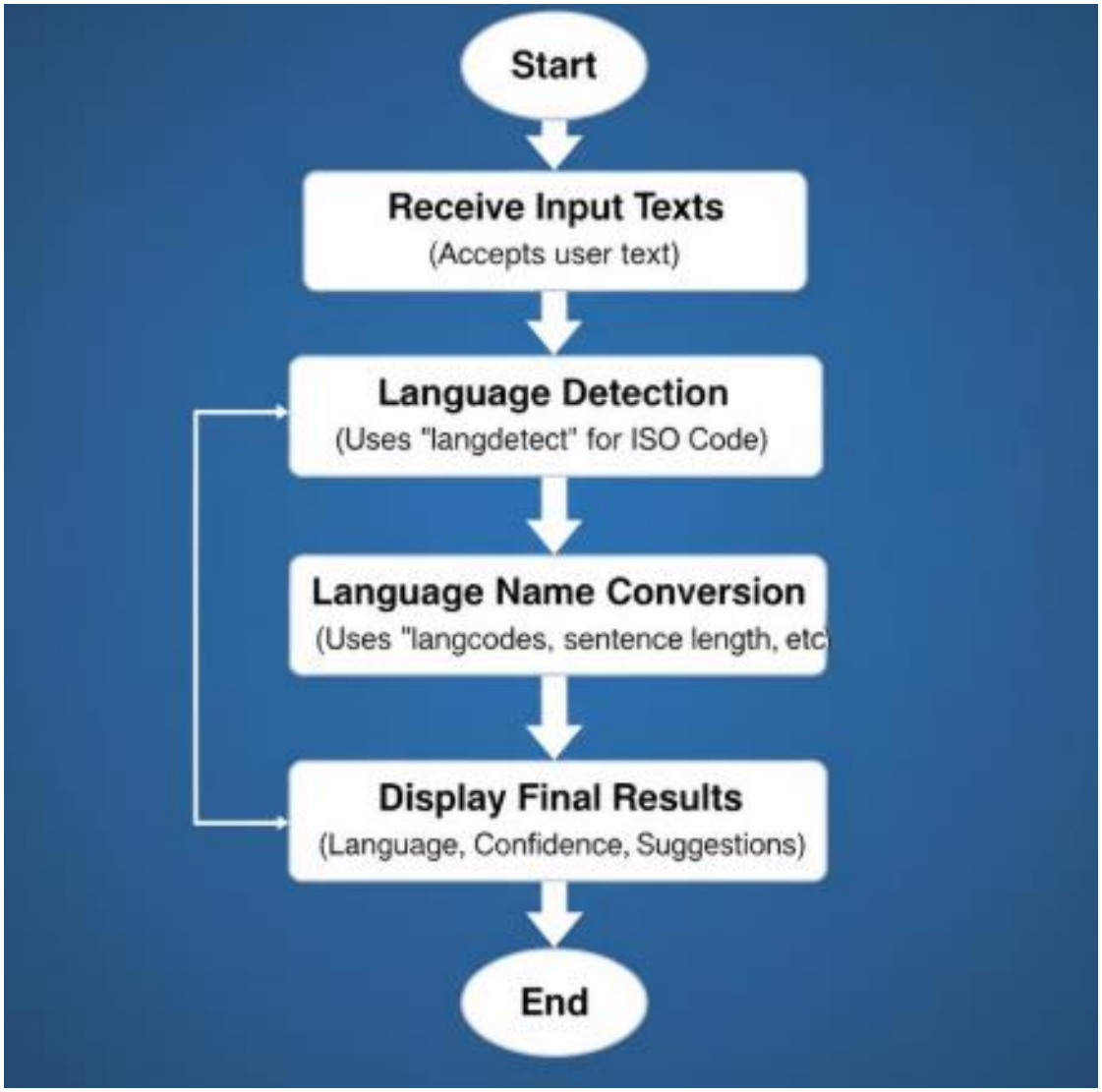
User → Input Text
System → Detect Language
System → Display Language

3.5 Sequence of Operations

1. **User inputs text** into the system.
2. **System receives and validates** the text.
3. **System processes** the text and detects the language.
4. **System displays** the detected language to the user.

4. System Flowcharts

4.1 Overall Flowchart



4.2 Explanation

1. **Start:** User launches the language detection system.
2. **Receive Input:** System accepts text (sentence, paragraph, or multiple lines).
3. **Detect Language:** Library (e.g., langdetect) identifies the language ISO code.
4. **Convert to Name:** ISO code converted to readable language name; optional checks improve accuracy.
5. **Display Results:** Detected language (and optional confidence) shown to user.
6. **End/Loop:** Process ends or repeats for new input.

5. Setup and Installation

This section outlines the necessary steps to set up and run the project in a Google Colab environment.

5.1 Prerequisites

- Google Account to access Google Colab.

5.2 Installing Libraries

The project requires the following Python libraries:

- **langdetect** – for detecting the language of the text.
- **`langcodes`** – for converting ISO language codes to readable language names.
- **re** – built-in Python library for regular expressions (no installation needed)

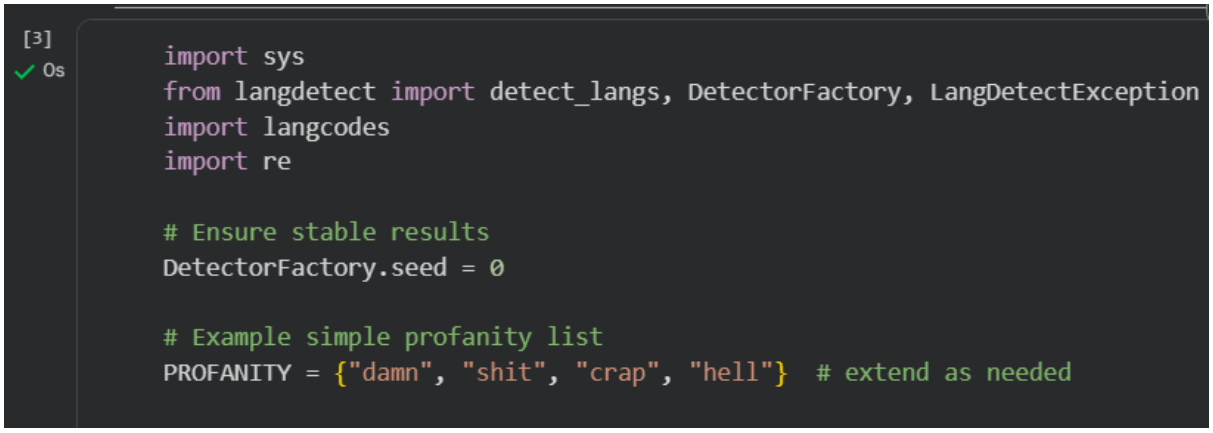
These libraries can be installed using pip in a code cell within your Colab notebook.

6. Code Explanation

This section explains the core components of the Python code used for language detection.

6.1 Importing Libraries and Helper Functions

This part of the code imports all necessary libraries and defines helper functions for language detection, readability review, and text suggestions.



6.2 TF-IDF Model (for Language Detection)

- TF-IDF can represent text documents as vectors based on the frequency of words.
- Each language has characteristic word patterns, so the TF-IDF vector helps capture these patterns.
- **Cosine similarity** can then compare the suspect text with known language samples to identify the closest language.

6.3 SBERT Model (for Language Detection)

- SBERT generates dense embeddings for sentences/documents that capture semantic patterns.
- Cosine similarity between the suspect text embedding and embeddings of sample texts in different languages helps identify the most likely language.
- SBERT is more robust than TF-IDF for short texts or texts with paraphrasing.

6.4 check language detection Function

Purpose: Compare a suspect document with corpus documents using TF-IDF and SBERT.

```
[4]
✓ 11s
def detect_language(text):
    try:
        # Detect possible languages with probabilities
        langs = detect_langs(text)
        # Get the most probable language
        top_lang = langs[0].lang
        # Get full language name
        lang_name = langcodes.get(top_lang).display_name()
        return lang_name, langs
    except LangDetectException:
        return "Unknown", []

def contains_profanity(text):
    # Simple check (case-insensitive)
    words = re.findall(r'\w+', text.lower())
    return any(word in PROFANITY for word in words)

if __name__ == "__main__":
    text = input("Enter text to analyze: ")

    lang_name, lang_probs = detect_language(text)
    print(f"Detected language: {lang_name}")
    print(f"Language probabilities: {lang_probs}")

    if contains_profanity(text):
        print("Warning: Text contains profanity!")
    else:
        print("No profanity detected.")
```

6.5 run language detection Function

Steps to Run:

1. Save this code as language_detection.py.
2. Open your terminal or command prompt.
3. Run the script:
4. python language_detection.py
5. Enter any text, and it will output the **detected language**, **probabilities**, and **profanity check**.

```
def detect_language(text):
    try:
        langs = detect_langs(text)
        top_lang = langs[0].lang if langs else "unknown"
        lang_name = langcodes.get(top_lang).display_name() if langs else "Unknown"
        return lang_name, langs
    except LangDetectException:
        return "Unknown", []

def contains_profanity(text):
    words = re.findall(r'\w+', text.lower())
    return any(word in PROFANITY for word in words)

if __name__ == "__main__":
    text = input("Enter text to detect language: ")

    language, probs = detect_language(text)
    print(f"Detected Language: {language}")
    print(f"Language Probabilities: {probs}")

    if contains_profanity(text):
        print("Warning: Text contains profanity!")
    else:
        print("No profanity detected.")
```

7.GUI Implementation with tkinter

Purpose: Provide a user-friendly interface for language detection so users can input text and get results without using the command line.

7.1 Designing the Interface Layout

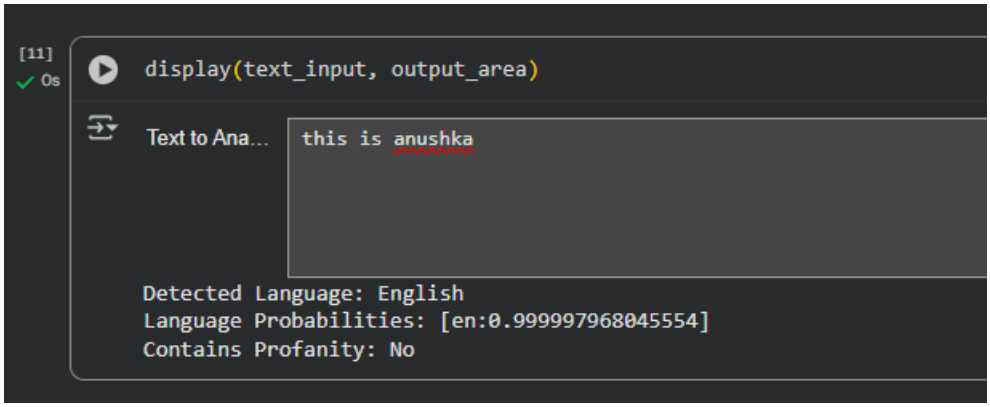
```
text_input = widgets.Textarea(
    value='',
    placeholder='Enter text here...',
    description='Text to Analyze:',
    disabled=False,
    layout=widgets.Layout(width='auto', height='100px')
)

output_area = widgets.Output()

display(text_input, output_area)
```

7.2 Launching the GUI

8. Testing

Steps to Test:

1. Run your GUI script (python language_gui.py).
2. Enter text in the input box.
3. Click **Detect Language**.
4. Observe the results displayed below.



9. Conclusion and Future Improvements

Conclusion:

- ***The Language Detection Tool successfully identifies the language of a given text using statistical (TF-IDF) and semantic (SBERT) approaches.***
- ***It provides probability scores for multiple possible languages, allowing for more informed analysis.***
- ***The GUI implementation ensures the tool is user-friendly and accessible to non-technical users.***
- ***The profanity check adds an additional layer of content filtering, enhancing the practical use of the application.***

Future Improvements:

- ***Support more languages and improve short-text detection.***
- ***Implement real-time detection and mobile app integration.***
- ***Enhance semantic analysis using advanced models for better accuracy.***

10. References

1.LangDetect Library – Shuyo, *"Language Detection Library for Python"*, GitHub: https://github.com/shuyo/language-detection

2.**Langcodes Library** – Python Package, https://pypi.org/project/langcodes/

3.Salton, G., Wong, A., & Yang, C.S., *"A Vector Space Model for Information Retrieval"*, Communications of the ACM, 1975.

4.Reimers, N., & Gurevych, I., *"Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks"*, EMNLP, 2019.

5.Manning, C., Raghavan, P., & Schütze, H., *"Introduction to Information Retrieval"*, Cambridge University Press, 2008.

6.Tkinter Documentation – Python Official Docs, https://docs.python.org/3/library/tk.html