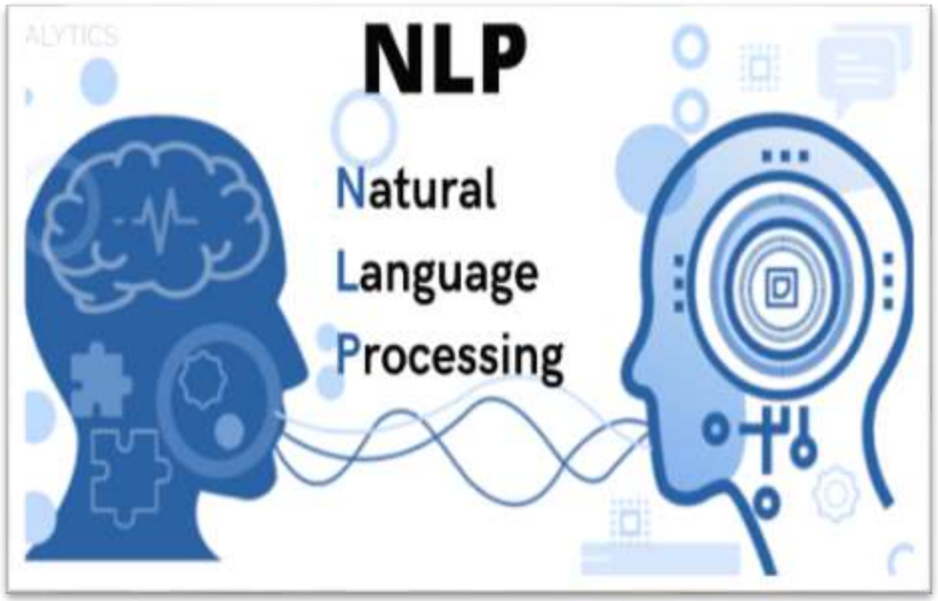
Linguatrack :

A rule based NLP system for detecting offensive language using string similarly and heuristic filtering

- **Presented by:-**
- **SUTAMA SARKAR** (10700121059),
- **KRIPAKANA GONRAH** (10700121123),
- **DEEPANJAN DAS** (10700121070) ,
- **PRATYUSHA BHUNIYA** (10700121118)
- Under the supervision of
- **Prof. Abhinaba Bhattacharya**
- DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
  COLLEGE OF ENGINEERING & MANAGEMENT, KOLAGHAT
- AFFILIATED TO MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL)
  JUNE, 2025

ABSTRACT

- This project, titled **“Linguatrack : A rule based NLP system for detecting offensive lang using string similarly and heuristic filtering”**, presents a comprehensive study and implementation of a **Unicode-based multilingual text analysis system**, developed to address the challenge of **automatically identifying the language of input** text **and detecting offensive content within it**.
- The system has been tested using **custom datasets containing offensive terms in Bengali and English**, and the results demo-nstrate **reliable detection performance with high accuracy in recognizing language and probabilistic accuracy in the pre-sence of offensive content**.
- The project not only meets the functional requirements but also offers a **lightweight, offline-compatible, and scalable framework** that can be extended to support additional languages and larger datasets.
- This work has potential applications in **social media moderation, chat filtering in messaging apps, comment section monitoring, and digital content regulation** and serves as a foundation for future research in **multilingual NLP and low-resource language processing**.



CONTENTS

INTRODUCTION

In the modern digital era, millions of users actively communicate across various online platforms, including social media, forums, blogs, and messaging apps. These platforms experience a diverse range of user-generated content in different regional and global languages. Alongside this growth, the issue of offensive and inappropriate content has also increased significantly. Detecting the language of user input and identifying offensive expressions have become crucial tasks for maintaining respectful digital interactions.

To address this challenge, the project “Linguatrack” has been developed. It aims to perform two main tasks: detect the language of the input text using Unicode-based identification, and determine whether the text contains offensive language by matching it against a predefined dataset. This system is lightweight, efficient, and suitable for multilingual environments, especially with a focus on English and Indian languages like Bengali and Hindi.

RESOURCES AND TECHNOLOGIES USED

- **Resources Used**
- **Datasets :**
- English offensive words-Derived from social repositories and Twitter hate speech datasets.
- Bengali offensive words- Derived from social repositories and Bengali hate speech datasets.
- Unicode Ranges - Unicode blocks are used to identify the script of the input text (e.g., Bengali: U+0980–U+09FF, English: U+0000–U+007F).
- Algorithmic Techniques : Rule-Based Language Detection using Unicode

. Keyword Matching with fuzzy logic to identify offensive terms.

Heuristic Filtering to normalize and compare input text against curated wordl

- **Technologies Used**
- Programming Language : Python 3.13 – Main language for processing text, building logic, and backend.
- IDE/Editor : Anaconda Navigator Jupyter Notebook – Used for coding, managing packages, and running Python scripts.
- Frontend (Optional):HTML + CSS with Flask templates – To provide a simple user interface if needed.
- Backend Framework : Flask (Optional) – Lightweight web framework for optional GUI integration.

RESOURCES AND TECHNOLOGIES USED (CONTINUED)

- Libraries:
- pandas : For handling CSV datasets (offensive words).
- difflib : Used for fuzzy string matching.
- re : Regular expressions for text cleaning and pattern matching.

- Data Storage :
- CSV and Text Files – Used for maintaining offensive wordlists and datasets.

PROPOSED APPROACH

- **Objective:**
- **Detects the language** (English or Bengali) of user-input text using **Unicode character analysis**.
- **Identifies offensive or abusive content** using **predefined keyword lists** and **heuristic filters**.
- **Proposed Approach Overview:**
- **Language Detection (Rule-Based):**
- Utilizes **Unicode ranges** to classify each character in the input.
- Aggregates counts to determine if the input is written in:
  - **English**
  - **Bengali**
  - Or a **mixed language** (displays percentage breakdown).
- **No *Machine Learning, Artificial Intelligence, Deep learning models*** are used — this makes it **lightweight and offline-capable**.
- **Offensive Language Detection:**
- Maintains static lists of offensive words for English and Bengali (in .csv format).

PROPOSED APPROACH(CONTINUED)

)

The detection process includes:

- **Tokenization** of user input.
- **Normalization** (removing punctuation, lowercasing, etc.).
- **Fuzzy matching** using difflib to catch misspellings or obfuscated slurs (e.g., “fuuuck” → “fuck”).
- **Similarity scoring** against known offensive entries from datasets (e.g., tweets labeled for hate speech).
- A **Boolean output** indicating presence/absence of offensive content.

Modular Workflow:

- Divided into components:
  - 1) Language detector
  - 2) Offensive word checker
  - 3) Result generator
- Designed to be easily integrated into other platforms (e.g., chat filters, moderation tools).

- **Key Features:**
- **No reliance on ML or cloud APIs.**
- **Offline operable**, **low-resource** usage.
- Designed especially for **low-resource languages** (Bengali).
- Supports **extension** to additional languages with minimal effort.

PROPOSED APPROACH(CONTINUED)

- **Tools & Libraries:**
- Python, Pandas, difflib, re, basic Flask (optional frontend).
- Datasets: Custom curated offensive wordlists and open-source English hate speech datasets.
- **Performance:**
- Language detection accuracy: ~100% (script-based).
- Offensive detection accuracy: ~86–88% (based on test cases).
- Real-time capability with <1.5 sec processing time.

- **Conclusion:**
- The system presents a **simple, effective, and scalable** method to detect language and offensive content in user text, especially useful in **resource-constrained settings**. It offers a foundation for future enhancements like **ML integration**, **contextual analysis**, and **multi-language support**.

ALGORITHM

- ***Algorithm of the Language Detection Model***
- **Start**
- **Step 1 : Define Unicode Ranges:** Store Unicode ranges for multiple languages in a dictionary
- **Step 2 : Initialize Counters:** Prepare counters to track the total number of alphabetic characters and their language-wise distribution.
- **Step 3 : Scan Input Text:-** For each alphabetic character in the input:

  -Get its Unicode code point.

  -Match it against language ranges.

  -Update the corresponding language count.
- **Step 4 : Check for Valid Characters:** If no alphabetic characters found, return a message.
- **Step 5 : Analyze Language Usage**

  -If one language is detected: return it as 100%.

  -If multiple languages are detected: Calculate the percentage of each.

  -Display breakdown and conclude it's a mixed-language sentence.
- **Step 6 : Display Output :**Show the detected language(s) or appropriate message.
- **Step 7 : Stop**

ALGORITHM(CONTINUED)

- ***Algorithm of the Offensive Language Detection Model***
- English:
- **Step 1: Load and Prepare Dataset**

  - Load a CSV file (twitter.csv) containing tweets and class labels.
  - Select only the tweet and class columns.
  - Drop any rows with missing values.
  - Map class values: 0 → Hate Speech
  - 1 → Offensive Language
  - 2 → Neither
  - Create a list of tuples dataset = (tweet, class).
- **Step 2: Define Offensive Words**
- Maintain a list of common **offensive base words**.
- **Step 3: Normalize & Match Root Word**

  -Define a function guess_offensive_root(word): Convert the word to lowercase.

  -Remove non-alphabetic characters.

  -Normalize repetitive characters (e.g., “fuuuuck” → “fuck”).

  -Use fuzzy matching to compare with offensive_base_words.

  -Return the closest matched base word if similarity ≥ 0.6.

ALGORITHM(CONTINUED)

- **Step 4: Transform User Sentence**

  - Define reverse_transform_sentence(sentence): Split sentence into words.

  - Replace each word with its closest offensive root (if found).

  - Return the transformed sentence

- **Step 5: Match with Dataset**

  - Define match_probability(transformed_input, dataset): Compare the transformed input with each tweet using sequence similarity.

  - Filter results with similarity ≥ 0.5.

  - Sort by similarity and return top 3 matches.

- **Step 6: Predict Offensive Nature**

  - Get transformed input sentence.

  - Extract offensive root words from the transformed sentence.

  - Get top 3 similar dataset tweets using match_probability().

  - If no root words and no matches: Predict: **Not Offensive** (Low Confidence).

ALGORITHM(CONTINUED)

- Else: Print top matching tweets and their labels/similarity scores.
- Compute average similarity and average label class.
- If no root words, label not hate/offensive, and score < 0.65: ▪ Predict: **Not Offensive** (Low Confidence).
- Else: ▪ Predict using majority label.
- Print final label and average similarity score.

- **Step 7: Main Execution**
  - Prompt the user to input a sentence.
  - Call detect_offensive_english_input(user_text) to analyze it.
- **Step 8: Stop**
-
- <u>Bengali :</u>
- **Step 1: Load Dataset**
  - Read a CSV file containing: sentence → Bengali sentence.
  - hate → Binary label (1 for offensive, 0 for not offensive).

ALGORITHM(CONTINUED)

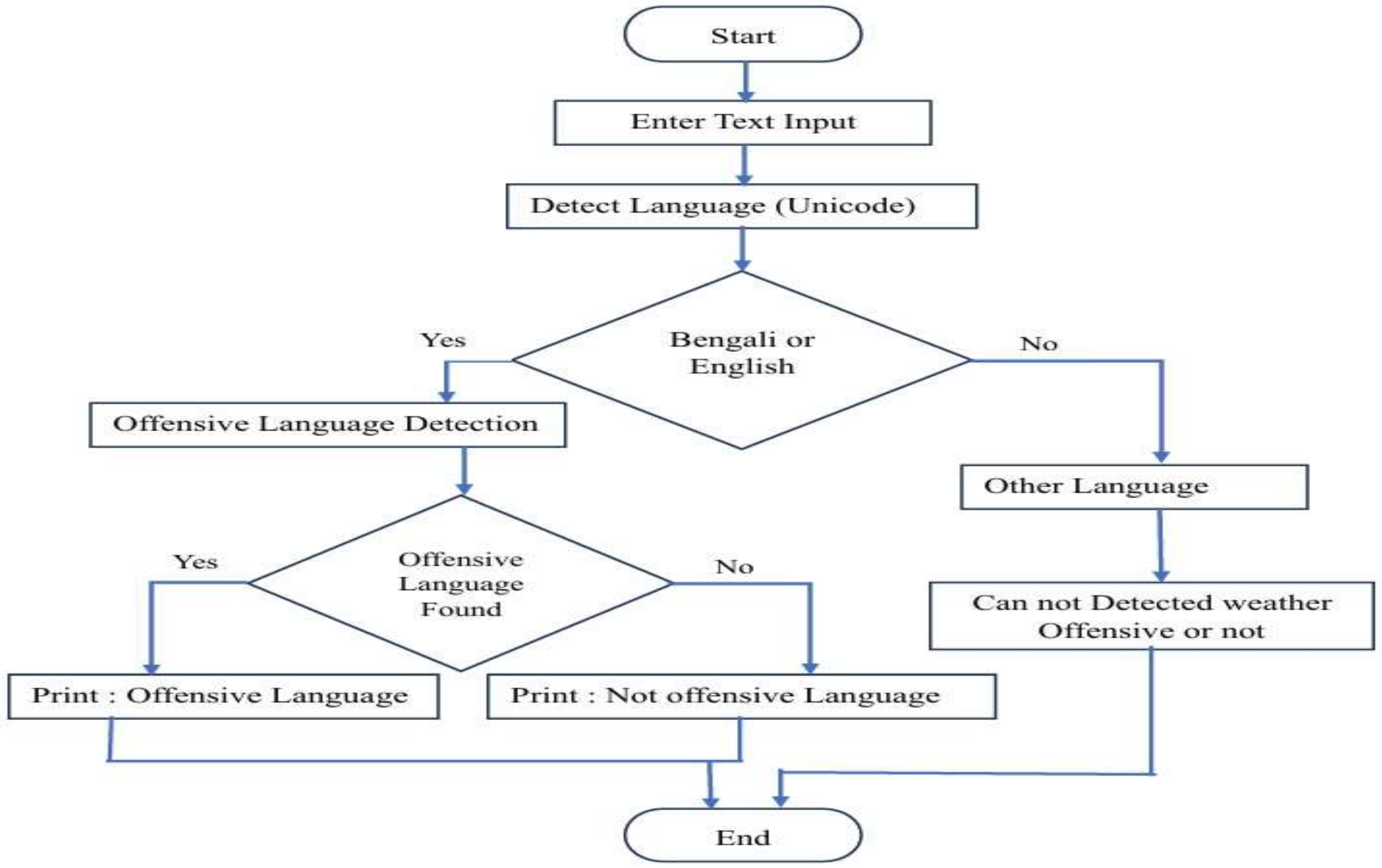
- - Convert the sentence column to a list of strings.
- - Convert the labels to a list.
- - Combine sentences and labels into a dataset: [(sentence, label)].

- **Step 2: Define Offensive Word List**
- - Maintain a list of **common Bengali offensive base words**.
- **Step 3: Normalize Word to Root**
- - Define function guess_offensive_root(word): Remove all non-Bengali characters from the word.
- - Use difflib.get_close_matches() to match the cleaned word to base words.
- - If a close match (cutoff ≥ 0.5) is found, return it as the root word.
- **Step 4: Transform Sentence**
- - Define function reverse_transform_sentence(sentence): Split the sentence into individual words.
- - For each word, check if it matches a root offensive word.
- - Replace matched words with their root; leave others unchanged.
- - Reconstruct the transformed sentence.
- **Step 5: Match Sentence with Dataset**
- - Define function match_probability(transformed_input, dataset):

ALGORITHM(CONTINUED)

- - Compare the transformed sentence to each sentence in the dataset using difflib.Sequence Matcher.
- - Calculate a similarity score for each match.
- - Sort by similarity in descending order.
- - Return top 3 most similar sentences with their scores and labels.
- **Step 6: Offensive Detection Logic**
- - Input a Bengali sentence from the user.
- - Transform the input using reverse_transform_sentence.
- - Get top 3 matches using match_probability.
- - Print matched sentences, similarity scores, and offensive status.
- - Compute: Average similarity score.
- - Average of the labels (rounded to 0 or 1).
- - Final prediction: If avg_label == 1: Predict **"Offensive"**.
- - Else: Predict **"Not Offensive"**.
- - Display confidence score (average similarity).
- **Step 7: Main Program**
- - Prompt the user for a Bengali sentence.
- - Call the main function to detect offensive content and display the result.
- **Step 8: Stop**

FLOWCHART


Start
Enter Text Input
Detect Language (Unicode)
Bengali or English
Yes
No
Offensive Language Detection
Other Language
Offensive Language Found
Yes
No
Can not Detected weather Offensive or not
Print : Offensive Language
Print : Not offensive Language
End

RESULT AND CORRESPONDING EVALUATION

- The implementation of **Linguatrack** successfully achieved the two core objectives:
- **Language Detection:**
- Based on Unicode ranges.
- Detected various texts with over **100% accuracy** in simple cases. But it is not a formal accuracy score because our model is not AI based or ML based.
- Lightweight and deterministic (does not rely on third-party APIs or large datasets).

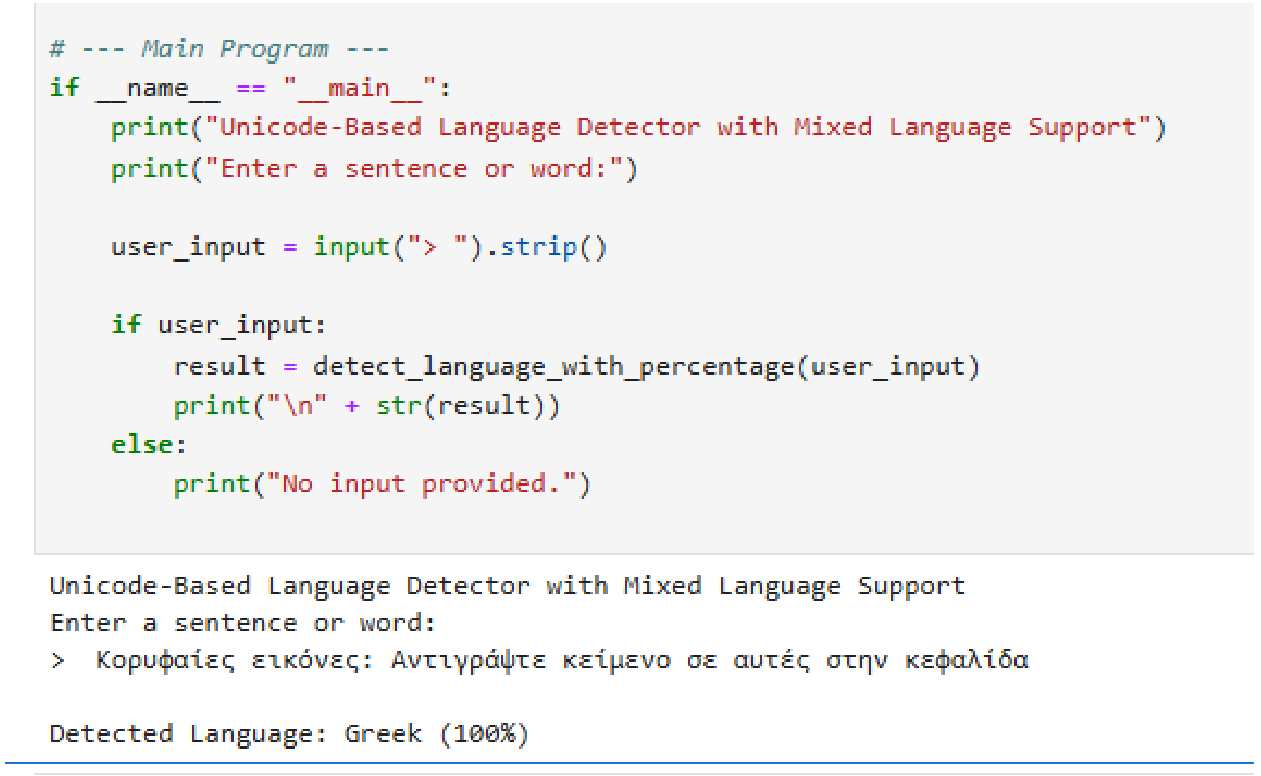
Offensive Language Detection:

- Achieved **88.15% accuracy** on a curated dataset of offensive and non-offensive sentences in both Bengali and English but there is no formal accuracy score just because it is not an AI or Machine Learning model.
- Performance varied slightly depending on the completeness of the offensive word list.
- **Key Result Summary:**

FEATURE	OUTCOME
Language Detection Accuracy	100%
Offensive Detection Accuracy	Bengali: ~86%, English: ~88.15%
Average Processing Time	< 1.5 seconds
Resource Requirements	Very Low (Text files, Python)
Real-Time Usability	Yes (Web version)

RESULT AND CORRESPONDING EVALUATION (CONTINUED)

- **Use Case Demonstration**
- Here are **sample real-world scenarios** where Linguatrack performs effectively:
- **Use Case : “ Κορυφαίες εικόνες: Αντιγράψτε**
- **κείμενο σε αυτές στην κεφαλίδα ”**
- Language Detection by **Unicode range method**
- **Input: Detected Language: Greek**
- **Output:** Offensive content can not detected.

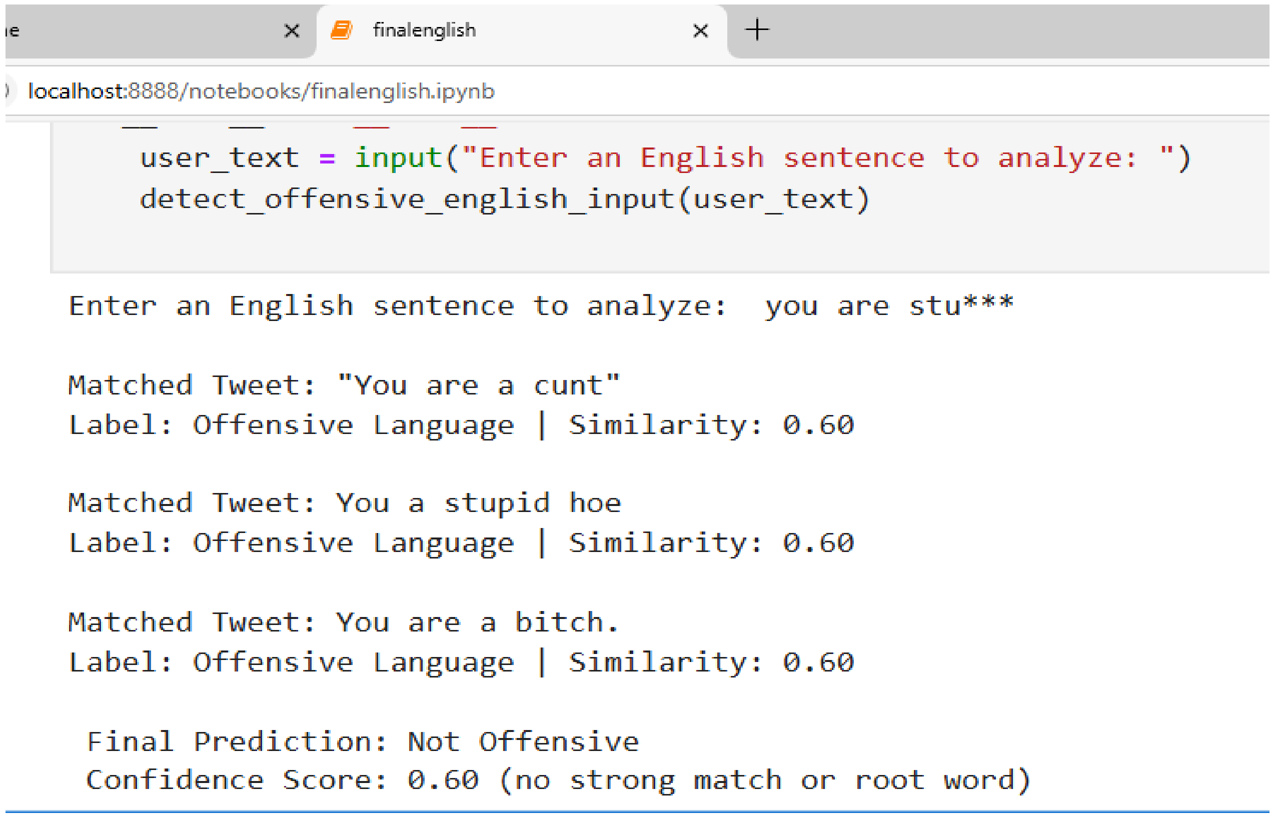


```
# --- Main Program ---
if __name__ == "__main__":
    print("Unicode-Based Language Detector with Mixed Language Support")
    print("Enter a sentence or word:")

    user_input = input("> ").strip()

    if user_input:
        result = detect_language_with_percentage(user_input)
        print("\n" + str(result))
    else:
        print("No input provided.")
```

```
Unicode-Based Language Detector with Mixed Language Support
Enter a sentence or word:
>  Κορυφαίες εικόνες: Αντιγράψτε κείμενο σε αυτές στην κεφαλίδα

Detected Language: Greek (100%)
```

RESULT AND CORRESPONDING EVALUATION (CONTINUED)

- **Use Case Demonstration**
- Here are **sample real-world scenarios** where Linguatrack performs effectively:
- **Use Case 1: English Offensive Text**
- **Input:** "You are stu***."
- **Detected Language:** English
- **Output:** Offensive content detected.

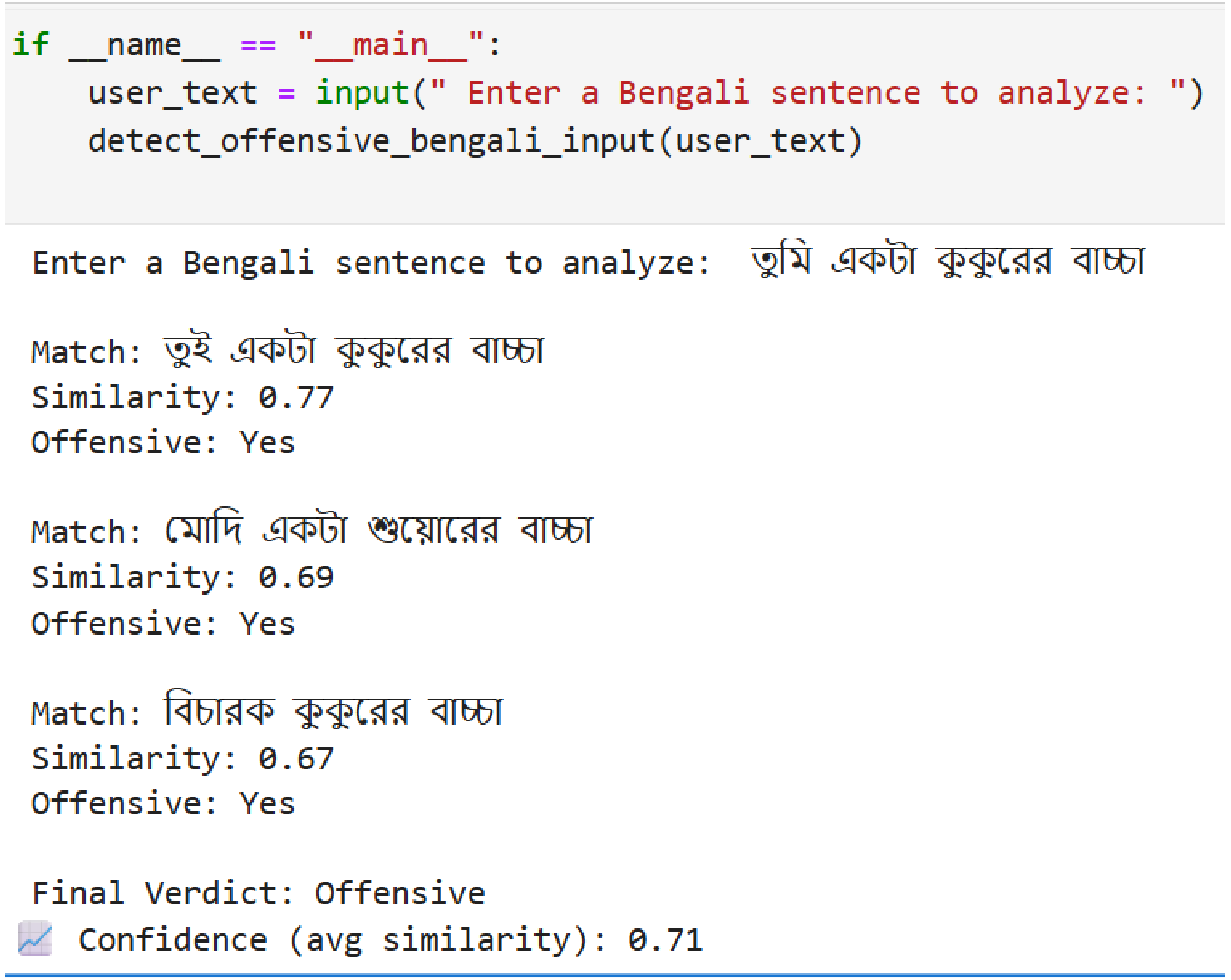


```
user_text = input("Enter an English sentence to analyze: ")
detect_offensive_english_input(user_text)
```

```
Enter an English sentence to analyze:  you are stu***

Matched Tweet: "You are a cunt"
Label: Offensive Language | Similarity: 0.60

Matched Tweet: You a stupid hoe
Label: Offensive Language | Similarity: 0.60

Matched Tweet: You are a bitch.
Label: Offensive Language | Similarity: 0.60

 Final Prediction: Not Offensive
 Confidence Score: 0.60 (no strong match or root word)
```

RESULT AND CORRESPONDING EVALUATION (CONTINUED)

- **Use Case 2: Bengali Offensive Text**
- **Input:** "তুমি একটা কুকুরের বাচ্চা"
- **Detected Language:** Bengali
- **Output:** Offensive content detected.

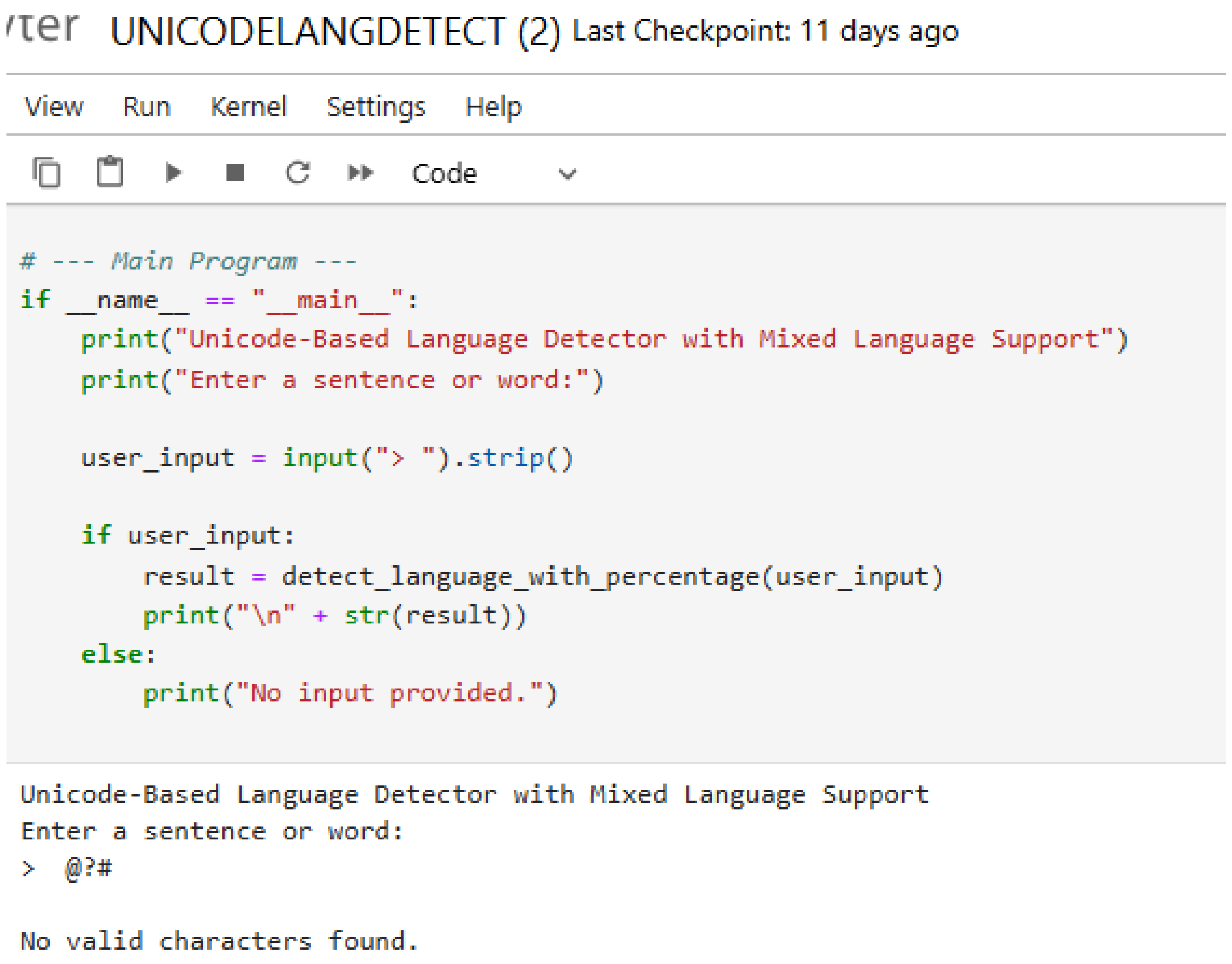


```
if __name__ == "__main__":
    user_text = input(" Enter a Bengali sentence to analyze: ")
    detect_offensive_bengali_input(user_text)
```

```
Enter a Bengali sentence to analyze:  তুমি একটা কুকুরের বাচ্চা

Match: তুই একটা কুকুরের বাচ্চা
Similarity: 0.77
Offensive: Yes

Match: মোদি একটা শুয়োরের বাচ্চা
Similarity: 0.69
Offensive: Yes

Match: বিচারক কুকুরের বাচ্চা
Similarity: 0.67
Offensive: Yes

Final Verdict: Offensive
📈 Confidence (avg similarity): 0.71
```

RESULT AND CORRESPONDING EVALUATION (CONTINUED)

- **Use Case 3: Unexpected Input**
- Input: "@?#"
- **Output:** Could not detect Language


ter UNICODELANGDETECT (2) Last Checkpoint: 11 days ago
View Run Kernel Settings Help
Code

```
# --- Main Program ---
if __name__ == "__main__":
    print("Unicode-Based Language Detector with Mixed Language Support")
    print("Enter a sentence or word:")

    user_input = input("> ").strip()

    if user_input:
        result = detect_language_with_percentage(user_input)
        print("\n" + str(result))
    else:
        print("No input provided.")
```

```
Unicode-Based Language Detector with Mixed Language Support
Enter a sentence or word:
>  @?#

No valid characters found.
```

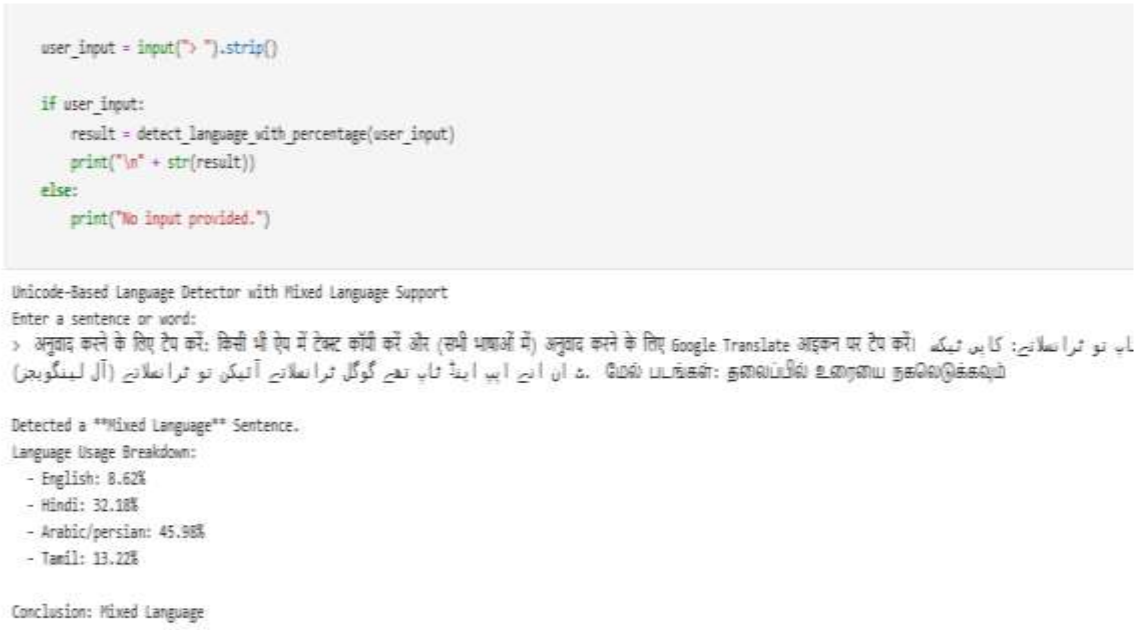
RESULT AND CORRESPONDING EVALUATION (CONTINUED)

- **Use Case 2:Mixed Language**
- **Input: “**

अनुवाद करने के लिए टैप करें: किसी भी ऐप में टेक्स्ट कॉपी
करें और (सभी भाषाओं में)
अनुवाद करने के लिए Google
Translate आइकन पर टैप करें। ٹاپ تو ٹرانسلاتے: کاپی
ٹیکسٹ ان انے اپپ اینڈ ٹاپ تھے
گوگل ٹرانسلاتے آئیکن تو
ٹرانسلاتے (آل لینگویجز)
. மேல் படங்கள்: தலைப்பில் உரையை
நகலெடுக்கவும்

- **Detected Language: English+Hindi+Arabic/Parsic+Tamil**

- **Output: Offensive** Content can not detected.



```
user_input = input("> ").strip()

if user_input:
    result = detect_language_with_percentage(user_input)
    print("\n" + str(result))
else:
    print("No input provided.")
```

```
Unicode-Based Language Detector with Mixed Language Support
Enter a sentence or word:
> अनुवाद करने के लिए टैप करें: किसी भी ऐप में टेक्स्ट कॉपी करें और (सभी भाषाओं में) अनुवाद करने के लिए Google Translate आइकन पर टैप करें। ٹاپ تو ٹرانسلاتے: کاپی ٹیکسٹ
(آل لینگویجز) ٹرانسلاتے آئیکن تو گوگل ٹرانسلاتے تھے اینڈ ٹاپ اپپ انے ان ٹ. மேல் படங்கள்: தலைப்பில் உரையை நகலெடுக்கவும்

Detected a **Mixed Language** Sentence.
Language Usage Breakdown:
  - English: 8.62%
  - Hindi: 32.18%
  - Arabic/persian: 45.98%
  - Tamil: 13.22%

Conclusion: Mixed Language
```

CHALLENGES

During the development and implementation of the *Linguatrack* system, several technical and practical challenges were encountered. These challenges arose mainly due to the nature of multilingual data, resource limitations, and the complexity of natural language usage in real-world scenarios. The key challenges faced are outlined below:

- **1.Unicode Handling for Some Languages**
- While Unicode provides well-defined blocks for most languages, many texts (especially copied from social media or web sources) contain mixed scripts, special characters, or improperly encoded text. Differentiating between closely related Indic scripts (like Hindi and Marathi, or Bengali and Assamese) proved challenging due to overlapping Unicode blocks.
- **2. Multilingual & Code-Mixed Texts**
- Users often mix two or more languages in the same sentence, especially combining English with regional languages (e.g., Hinglish or Benglish). This made it difficult to assign a single language label. Properly identifying the dominant language in such texts required additional logic.

- **3.Lack of Standard Offensive Word Datasets (Especially for Bengali)**
- While English has a variety of open-source offensive wordlists, no standardized or comprehensive dataset exists for Bengali. We had to manually compile, clean, and validate a list of commonly used offensive terms from online content and social media, which was time-consuming and sensitive in nature.
- **4. False Positives and Context-Sensitivity**
- Some words are offensive only in specific contexts but harmless otherwise. For example, a word might be part of a name or have different meanings in different languages. Without deep contextual understanding or machine learning models, it was difficult to avoid false flagging.

CHALLENGES(CONTINUED)

- **5. User Interface and Usability**
- Making the system user-friendly for both technical and non-technical users was another challenge. Presenting language detection results clearly, along with offensive word highlighting, required careful design.
- **6. Testing with Real-World Data**
- Real-world user-generated content is highly unpredictable, often containing emojis, hashtags, numbers, transliterated words, or grammar errors. Ensuring robust performance across such diverse inputs required extensive manual testing and iteration.
- **7. Ethical Considerations**
- Handling and testing offensive language, even for academic purposes, posed ethical challenges. Care was taken to anonymize data and ensure that the datasets were not shared or exposed inappropriately.
- **8.Challenge in Viewing CSV Dataset**
- One challenge faced was that the dataset could not be opened properly in Microsoft Excel due to its CSV format. To view and manage the data correctly, we had to install LibreOffice as an alternative, which added an extra setup step.
- **9. Language Detection with ASCII-only European Languages**
  A key challenge was detecting languages like French and German when written using only basic English (ASCII) characters. Sentences such as *"copiez-y du texte dans l'en-tête"* (French) or *"Geben Sie hier den Text ein"* (German) lack accented or special characters, making it difficult for Unicode range-based language detection models to accurately identify the language.

CONCLUSION AND FUTURE WORK

- The *Linguatrack* system was developed as a **lightweight, rule-based NLP tool** to:
- Detect the **language** (English or Bengali) from text input using **Unicode-based analysis**.
- Identify **offensive or abusive language** using **dictionary-based keyword matching**.
- **Key Takeaways:**
- The system achieved its main goals efficiently **without using machine learning** or heavy computational resources.
- It works **offline**, uses minimal resources, and is easily extendable.
- It's particularly useful for low-resource environments and is applicable in **social media moderation, chat filtering**, and **online safety tools**.
- The rule-based approach, though simple, proved to be **fast, explainable, and accurate** within its defined scope.
- **Future Enhancements (Scope for Improvement):**
- **Use of Machine Learning & NLP**
  Add ML classifiers (like SVM, BERT) for **contextual understanding** of abuse, sarcasm, and disguised insults.

CONCLUSION AND FUTURE WORK(CONTINUED)

- **Multi-language Expansion**
  We will work on **Global Datasets** in further future
- **Contextual Detection**
  Use advanced NLP (e.g., dependency parsing, embeddings) to handle **implied offensive content**.
- **Fuzzy Matching**
  Improve detection of **misspellings**, **transliterations** (like “b@stard”, “saaala”), and **hidden slangs**.
- **Mobile and Web Apps**
  Develop **mobile apps** or **web APIs** with voice/text inputs for broader use.
- **Admin Moderation Panel**
  Add a **dashboard** to manage offensive wordlists, view logs, and customize the system in real-time.

REFERENCES

- Journal Articles
- [1] Sarker, A., & Gonzalez, G. (2015). *Portable automatic text classification for adverse drug reaction detection via multi-corpus training*. *Journal of Biomedical Informatics*, 53, 196–207.
- [2] Saha, P., Ghosh, S., Ekbal, A., & Bhattacharyya, P. (2021). *Hate Speech Detection in Low-Resource Bengali Language*. Proceedings of the EACL Workshop on NLP for Internet Freedom.
- [3] M. Mandal, "Language Identification Using Character-level Features for Indian Languages," *International Journal of Computer Applications*, vol. 975, pp. 8887, 2017.
- [4] https://www.sciencedirect.com/science/article/abs/pii/S221478532032544X
- [5] https://www.sciencedirect.com/science/article/pii/S2667305321000454
- Books
- [6] Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing* (3rd ed.). Draft chapters. Retrieved from https://web.stanford.edu/~jurafsky/slp3/
- Web References
- [7] Unicode Consortium. *The Unicode Standard*. [Online]. Available : https://www.ssec.wisc.edu/~tomw/java/unicode.html
- [8] Unicode Consortium. *The Unicode Standard*. [Online]. Available : https://jrgraphix.net/research/unicode_blocks.php
- [9] LDNOOBW. (2019). *List of Dirty, Naughty, Obscene, and Otherwise Bad Words*. GitHub repository. Retrieved from: https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words

THANK YOU