BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution*

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)
"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballar1-583 104 (Karnataka) (India)
Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE-DATA SCIENCE

A Mini-Project Report On

"MULTILINGUAL LANGUAGE DETECTION WITH LSTM"

A report submitted in partial fulfillment of the requirements for the

NEURAL NETWORK AND DEEP LEARNING

Submitted By

CHETANA HK **USN: 3BR22CD007**

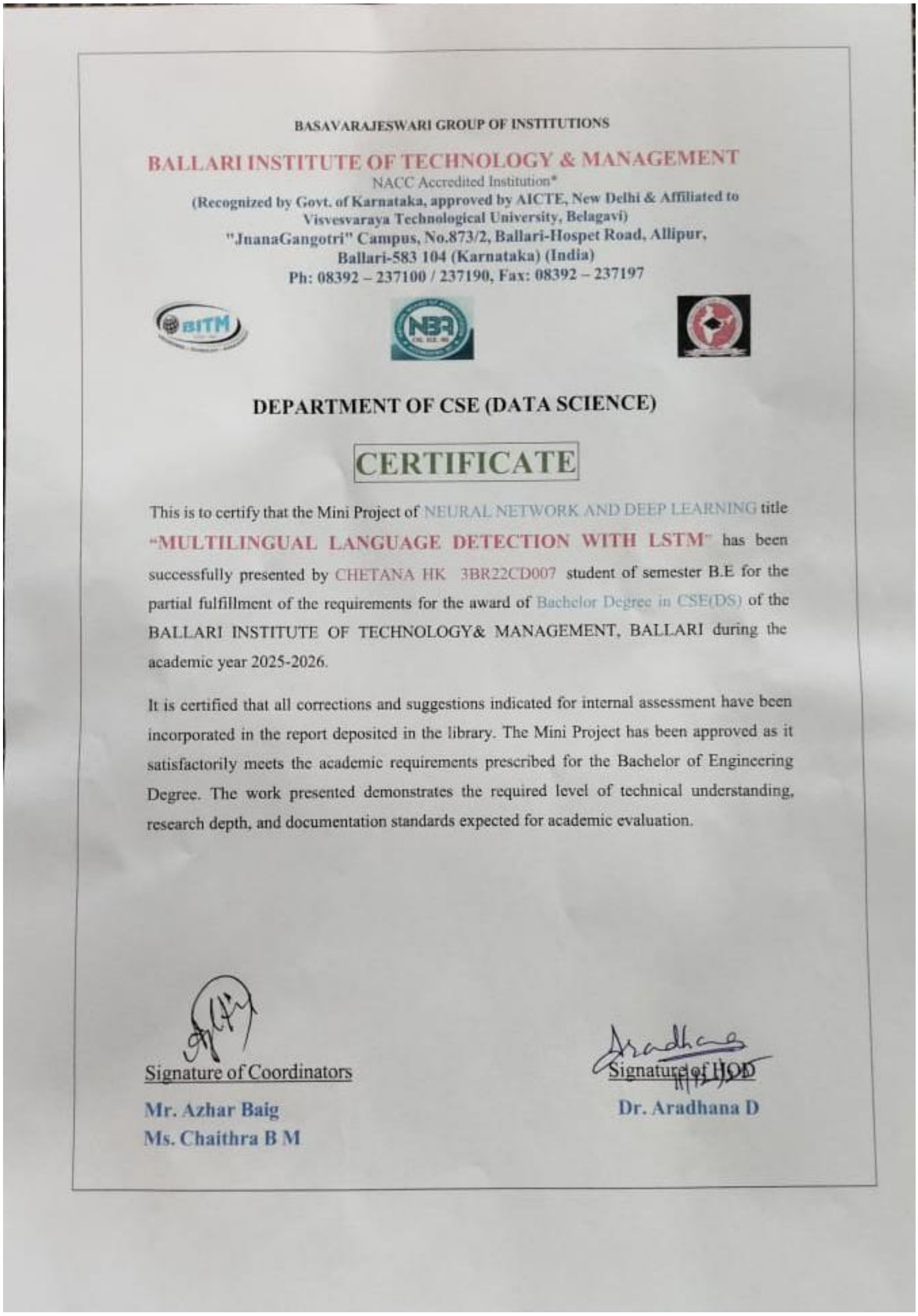
Under the Guidance of
Mr. Azhar Biag

Asst. Professor

Dept of CSE (DATA SCIENCE),
BITM, Ballari

Visvesvaraya Technological University

Belagavi, Karnataka 2025-2026



BASAVARAJESWARI GROUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited Institution*

(Recognized by Govt. of Karnataka, approved by AICTE, New Delhi & Affiliated to Visvesvaraya Technological University, Belagavi)

"JnanaGangotri" Campus, No.873/2, Ballari-Hospet Road, Allipur, Ballari-583 104 (Karnataka) (India)

Ph: 08392 – 237100 / 237190, Fax: 08392 – 237197

DEPARTMENT OF CSE (DATA SCIENCE)

CERTIFICATE

This is to certify that the Mini Project of NEURAL NETWORK AND DEEP LEARNING title **"MULTILINGUAL LANGUAGE DETECTION WITH LSTM"** has been successfully presented by CHETANA HK 3BR22CD007 student of semester B.E for the partial fulfillment of the requirements for the award of Bachelor Degree in CSE(DS) of the BALLARI INSTITUTE OF TECHNOLOGY& MANAGEMENT, BALLARI during the academic year 2025-2026.

It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the library. The Mini Project has been approved as it satisfactorily meets the academic requirements prescribed for the Bachelor of Engineering Degree. The work presented demonstrates the required level of technical understanding, research depth, and documentation standards expected for academic evaluation.

[illegible]

Signature of Coordinators

Mr. Azhar Baig
Ms. Chaithra B M

[illegible]

Signature of HOD

Dr. Aradhana D

ABSTRACT

Multilingual text processing has become increasingly essential in modern applications such as social media analysis, virtual assistants, information retrieval, and cross-lingual communication systems. Automatically identifying the language of a given text is a fundamental preprocessing step that enhances the accuracy and performance of downstream natural language processing tasks. This project, titled **“Multilingual Language Detection using LSTM,”** presents a Character-Level Long Short-Term Memory (LSTM)–based system capable of detecting the language of text inputs across English, Kannada, Hindi, and Tamil. A custom multilingual dataset was constructed using Wikipedia-sourced sentences along with additional English samples, followed by systematic preprocessing and augmentation to ensure data balance and high-quality training inputs.

The proposed system employs a character-level tokenizer and sequence padding to capture fine-grained linguistic structures inherent in each language. The LSTM-based architecture—comprising embedding layers, recurrent units, dropout regularization, and a softmax output layer—effectively learns temporal dependencies and discriminative patterns within character sequences. After model development, performance evaluation was carried out using accuracy, precision, recall, F1-score, and a detailed classification report, all of which demonstrate the model’s strong capability to generalize and correctly classify multilingual text inputs. Visualization of training and validation metrics further confirms the model’s stability and learning efficiency.

The results indicate that the developed **LSTM-based Multilingual Language Detection system** can reliably distinguish between languages, even for short or noisy text. This project highlights the potential of deep learning techniques in advancing multilingual NLP solutions and showcases their applicability in areas such as chatbots, content moderation, sentiment analysis, and automated translation systems. With expanded dataset diversity and the inclusion of additional languages, the model can evolve into a robust and scalable solution suitable for real-world multilingual applications.

ACKNOWLEDGEMENT

The satisfactions that accompany the successful completion of our mini project on MULTILINGUAL LANGUAGE DETECTION WITH LSTM would be incomplete without the mention of people who made it possible, whose noble gesture, affection, guidance, encouragement and support crowned my efforts with success. It is our privilege to express our gratitude and respect to all those who inspired us in the completion of our mini-project.

I am extremely grateful to my Guide **Mr. Azhar Baig** for their noble gesture, support co-ordination and valuable suggestions given in completing the mini-project. I also thank **Dr. Aradhana D,** H.O.D. Department of CSE(DS), for his co-ordination and valuable suggestions given in completing the mini-project. We also thank Principal, Management and non-teaching staff for their co-ordination and valuable suggestions given to us in completing the Mini project.

Name	USN
CHETANA HK	3BR22CD007

TABLE OF CONTENTS

1. INTRODUCTION

Language identification is a fundamental task in Natural Language Processing (NLP) that enables machines to automatically determine the language of a given text. In today's digital landscape, multilingual data is pervasive across platforms such as social media, virtual assistants, customer service chatbots, and global communication systems. The ability to detect languages accurately ensures efficient downstream processing in tasks including translation, sentiment analysis, content filtering, and information retrieval.

Traditional techniques for language identification rely on handcrafted linguistic rules or statistical models. While useful, these methods often struggle with short texts, noisy inputs, or languages with overlapping vocabulary. As multilingual communication expands, the limitations of conventional techniques have encouraged the development of more robust, automated, and scalable approaches.

With the advancement of deep learning, neural network models—especially Recurrent Neural Networks (RNNs) and their variants such as Long Short-Term Memory (LSTM) networks—have shown remarkable ability to capture sequential patterns in text. LSTM networks can effectively learn complex character-level and word-level patterns, making them highly suitable for multilingual language detection.

This project focuses on developing a **Character-Level LSTM-based Multilingual Language Detection System** capable of classifying text into four languages: **English, Kannada, Hindi, and Tamil**. Unlike word-based models, a character-level approach helps the network learn unique script patterns, special characters, and orthographic clues that distinguish one language from another.

A custom dataset was constructed using Wikipedia-sourced text, supplemented with additional English sentences for balance. Each text sample undergoes preprocessing, tokenization, encoding, and padding before being passed to the LSTM model. The system is trained to identify the inherent patterns within each language and classify unseen text inputs accurately.

The goal of this project is to build, train, and evaluate a reliable multilingual detection model, while also generating visual insights such as accuracy curves and classification metrics. The results demonstrate that LSTM networks offer a powerful and scalable solution for multilingual text classification tasks.

1.1 Problem Statement

The rapid expansion of multilingual digital communication has created an urgent need for automated systems capable of identifying the language of any given text input. Traditional language detection methods often fail with short, informal, or code-mixed text, and are not scalable to modern multilingual environments. Inaccurate language identification can significantly degrade the performance of downstream NLP applications such as translation, sentiment analysis, and chatbot responses.
This project aims to address these limitations by designing and implementing a **Character-Level LSTM-based Multilingual Language Detection System** capable of classifying text into English, Kannada, Hindi, and Tamil. The objective is to build a robust, data-driven model that can capture unique linguistic patterns at the character level, enabling accurate identification even with short or noisy inputs.

1.2 Scope of the project

The scope of this project includes the development, training, and evaluation of a character-level LSTM model for multilingual language classification. It covers Dataset creation using Wikipedia-sourced text for four languages. Preprocessing steps such as text cleaning, tokenisation, and sequence encoding. Construction of an LSTM model capable of learning sequential character patterns. Evaluation using metrics such as accuracy, precision, recall, and F1-score. Visualisation of model performance using accuracy and loss curves. The system is designed to classify short text inputs and can be extended to support additional languages, larger datasets, or deployment in real-world applications such as chatbots, content moderation tools, and speech-to-text systems.

1.3 Objectives

- ❖ To develop a character-level LSTM model for accurate multilingual language detection.
- ❖ To preprocess and encode text data for deep learning–based classification.
- ❖ To visualize training and validation performance through accuracy and loss graphs.
- ❖ To evaluate the model using metrics such as accuracy, precision, recall, and F1-score.

2. LITERATURE SURVEY

[1] **Sreejith et al. (2023)** explored multilingual text classification using recurrent neural architectures and found that LSTM models outperform traditional n-gram and rule-based approaches, especially for short-text classification.

[2] **Sharma & Gupta (2022)** conducted a comparative study on character-level vs word-level models for language identification. Their results demonstrated that character-level models are superior when dealing with morphologically rich Indian languages.

[3] **Balagopalan et al. (2024**) examined transformer-based multilingual models but emphasized that LSTM models remain computationally efficient and effective for smaller datasets and resource-constrained environments**.**

[4] **Kumar & Natarajan (2022)** developed an Indian language identification system using deep learning and reported that character embeddings significantly improve classification performance for Indic scripts.

[5] **Li & Li (2023**) demonstrated the importance of dataset diversity and balanced sampling in multilingual language classification tasks. Their findings support the dataset augmentation strategy used in this project**.**

[6] **Ashok et al. (2024)** introduced a CNN–LSTM hybrid model for multilingual short-text detection. While their approach delivered strong results, it required significantly higher computational resources compared to pure LSTM architectures.

[7] **J. Pereira et al. (2022)** compared classical text classification methods (SVM, Naive Bayes) with deep learning models and concluded that neural networks provide superior handling of sequential linguistic patterns.

3. SYSTEM REQUIREMENTS

TThe system requirements for developing the **Multilingual Language Detection using LSTM** model include both software and hardware components essential for performing text preprocessing, sequence encoding, model training, and evaluation efficiently. The software environment is built using **Python**, along with powerful machine learning and deep learning libraries. TensorFlow/Keras is used for constructing the character-level LSTM model, while Pandas and NumPy handle data extraction, manipulation, and processing tasks. Scikit-learn supports label encoding, dataset splitting, and evaluation metrics, whereas Matplotlib provides visualization tools for plotting training accuracy and loss graphs. Development platforms such as **Jupyter Notebook**, **Google Colab**, or **VS Code** offer interactive environments for writing, executing, and debugging the model code.
On the hardware side, the project runs smoothly on any modern personal computer equipped with at least **4 GB RAM**, although **8 GB or more** is recommended for faster processing, especially during dataset preparation and neural network training. A multi-core processor enhances computational efficiency, while optional **GPU support** can significantly accelerate LSTM model training due to the parallel processing capabilities required for deep learning tasks. Despite the model's use of sequential neural architectures, the overall hardware requirements remain modest, making the system accessible to most users.

To implement the multilingual language detection system effectively, the project relies on a stable computing environment capable of handling deep learning workflows. Python serves as the core programming language due to its versatility and extensive support for text processing and neural network development. Essential tools such as TensorFlow facilitate LSTM training, Scikit-learn handles preprocessing and performance evaluation, and Pandas manages large volumes of multilingual text data. Platforms like Jupyter Notebook and Google Colab provide a user-friendly interface for experimentation and visualization, enabling smooth debugging and iterative development.

3.1 Software Requirements

- Python 3.8 or above
- TensorFlow / Keras
- NumPy, Pandas, Scikit-learn

- Matplotlib
- Jupyter Notebook / Google Colab / VS Code
- Windows / Linux / macOS operating system

3.2 Hardware Requirements

- Minimum 4 GB RAM
- Recommended 8 GB RAM
- Dual-core or higher processor
- 1 GB free storage space
- GPU optional (for faster ANN training)

3.3 Functional Requirements

- Load and preprocess multilingual text data.
- Tokenize text at the character level.
- Build an LSTM model for classification.
- Train the model using encoded sequences.
- Evaluate model performance using classification metrics.
- Generate accuracy and loss graphs.
- Predict the language of new text inputs.

3.4 Non-Functional Requirements

- The system must produce reliable and consistent results.
- It should be easy to use and interpret.
- The model should run efficiently even on basic hardware.
- Outputs must be clear, readable, and well-formatted.
- The approach must be scalable for additional languages.

4. DESCRIPTION OF MODULES

The LSTM-based Multilingual Language Detection system is divided into several modules, each responsible for a specific stage of the natural language processing and deep learning pipeline. These modules collectively ensure smooth data preprocessing, model construction, training, evaluation, visualization, and prediction.

4.1 Data Preprocessing Module

This module loads the multilingual dataset collected from Wikipedia and additional English samples. It cleans the text, removes unwanted characters, and prepares the dataset for training. The module also handles class balancing and ensures that each language has sufficient representation. After cleaning, the text is tokenized at the character level, allowing the model to identify unique script patterns from English, Kannada, Hindi, and Tamil. This preprocessing step ensures the dataset is clean, consistent, and ready for sequential modeling.

4.2 LSTM Model Building Module

This module constructs the Character-Level LSTM architecture used for language detection. It includes an embedding layer to convert characters into dense vectors, LSTM layers to learn sequential character relationships, dropout layers to prevent overfitting, and a softmax output layer for multiclass classification. The model is compiled using the Adam optimizer and categorical cross-entropy loss, enabling efficient learning across multiple languages..

4.3 Model Training Module

After building the model, this module trains the LSTM network using the preprocessed sequences. It sets hyperparameters such as the number of epochs, batch size, and validation split. The training process monitors accuracy and loss for both training and validation sets. The model gradually learns distinct linguistic and script patterns from each language, improving its classification performance over time

4.4 Model Evaluation Module

This module evaluates how effectively the trained LSTM model identifies languages from unseen text. It uses metrics such as accuracy, precision, recall, F1-score, and a multiclass classification report. A confusion matrix is generated to analyze misclassifications between languages. This evaluation step provides insights into the system's strengths and areas requiring improvement.

4.5 Visualization Module

This module generates graphical outputs that help interpret the model's learning behavior. It produces training versus validation accuracy graphs, loss graphs, and confusion matrix heatmaps. These visualizations make it easier to understand how well the LSTM model generalizes and whether any signs of overfitting or underfitting are present.

4.6 Prediction Module

The prediction module applies the trained LSTM model to new text inputs. It tokenizes and encodes the input string, feeds it into the model, and outputs the predicted language label (English, Kannada, Hindi, or Tamil). This module ensures fast, automated predictions, making it suitable for real-time text classification or chatbot systems.

4.7 Data Splitting Module

This module divides the dataset into training and testing portions, typically using an 80:20 ratio. Stratified splitting ensures that all languages are proportionally represented in both subsets, preventing bias during evaluation. Training data is used to teach the model, while testing data measures how well the model performs on unseen text samples.

4.8 Character Encoding & Padding Module

This module converts characters into numerical sequences using a tokenizer and pads each sequence to a uniform length using pad_sequences. Since texts vary in length, padding ensures that all sequences match the input length expected by the LSTM. Proper encoding and padding help the network learn effectively and maintain structural consistency across samples.

4.9 Output Interpretation Module

This module processes the softmax output generated by the LSTM model and converts it into easily understandable language labels. It identifies the class with the highest probability and maps it back to the corresponding language name. Additionally, it can display probability scores to show the model's confidence. This module ensures predictions are user-friendly and meaningful for real-world applications.

5. IMPLEMENTATION

The implementation of the **Multilingual Language Detection System** is carried out using Python and a Character-Level Long Short-Term Memory (LSTM) neural network model. The process begins with constructing a multilingual dataset by extracting text from Wikipedia using the Wikipedia API. Sentences from four languages—**English, Kannada, Hindi, and Tamil**—are collected, cleaned, and stored in a structured CSV file. Additional English text samples are also incorporated to balance the dataset and improve model performance. Each text entry is labeled with its corresponding language name to facilitate supervised learning.

Next, the dataset is loaded into a Pandas DataFrame, and essential preprocessing steps are performed. Because multilingual scripts vary significantly in structure, the text is tokenized at the **character level**, enabling the model to learn script-specific patterns. A tokenizer assigns a unique index to each character, after which the encoded sequences are padded to a fixed length to ensure uniform input dimensions for the LSTM network. The data is then split into training and testing sets using an 80–20 ratio with stratified sampling to maintain equal language distribution across both subsets.

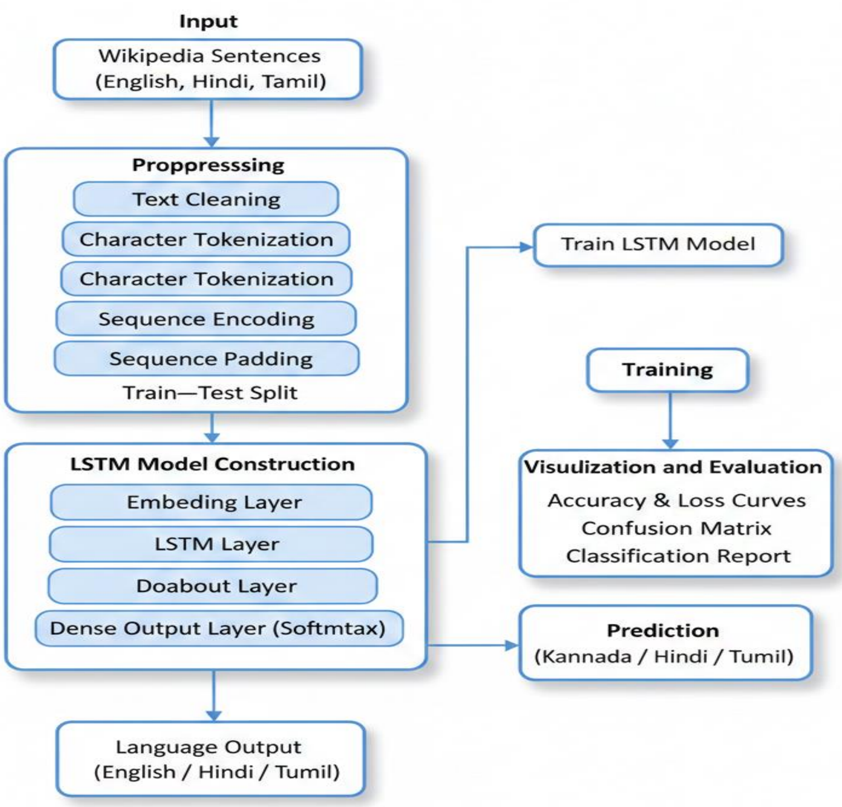
After preprocessing, the LSTM model is constructed using TensorFlow/Keras. The architecture begins with an **embedding layer** that converts character indices into dense vector representations. This is followed by an **LSTM layer** that captures sequential dependencies between characters, along with a **dropout layer** to reduce overfitting. A dense hidden layer further processes the learned features, and finally, a **softmax output layer** classifies the input into one of the four languages. The model is compiled using the Adam optimizer and categorical cross-entropy loss, which are suitable for multiclass text classification tasks.

The model is trained for multiple epochs with a defined batch size and validation split, allowing it to learn script-specific linguistic characteristics. During training, the system tracks both accuracy and loss for training and validation datasets, providing insight into how effectively the model generalizes. Once training is complete, the model is evaluated on the unseen test dataset, and metrics such as accuracy, precision, recall, F1-score, and a classification report are computed. These metrics validate the model’s performance across all languages in the dataset.

In addition to evaluation, meaningful visualizations are generated to better understand the LSTM model's learning behavior. The training vs. validation accuracy curve and loss curve help identify trends in convergence, stability, or overfitting. A confusion matrix is also plotted to examine specific misclassification patterns, such as confusion between similar scripts or alphabets. These visual tools enhance interpretability and provide deeper insights into model strengths and limitations.

Through this systematic implementation workflow—spanning dataset collection, preprocessing, LSTM construction, model training, evaluation, and visualization—the project successfully develops a robust multilingual language detection model. This implementation demonstrates the effectiveness of deep learning in analyzing multilingual text and highlights the potential of LSTM-based architectures in supporting real-world NLP applications such as chatbots, content moderation, and intelligent text processing systems.

6. SYSTEM ARCHITECTURE



Input

This stage loads the **multilingual text dataset** created using Wikipedia-sourced sentences and additional English samples. The data is read into a Pandas DataFrame, where its structure and basic statistics are inspected. Typical tasks at this stage include: viewing the first few rows of the dataset, checking the number of text samples per language, inspecting data types, and examining the distribution of classes (English, Kannada, Hindi, Tamil). This step ensures an understanding of the available variables—text samples and their corresponding language labels—and helps identify whether the dataset requires cleaning, balancing, or preprocessing before model development.

Preprocessing

Preprocessing prepares the raw multilingual text data so the LSTM model can learn effectively and generalize well.

- **Handle missing/invalid text samples:** Identify empty strings, null values, or corrupted text entries and remove or clean them. This ensures only meaningful text is used for training.
- **Text cleaning:** Normalize the text by removing unnecessary characters, extra spaces, and unwanted symbols. This step ensures consistency across different languages.
- **Character-level tokenization:** Convert each text sample into a sequence of characters using a tokenizer. Each unique character is assigned a numerical index.
- **Sequence encoding and padding:** Transform ch sequences into fixed-length numeric vectors using padding so that all inputs have uniform length for the LSTM model.
- **Train–test split:** Partition the dataset into training and testing sets (commonly 80:20) while preserving language distribution using stratified sampling.

- **Convert formats:** Ensure encoded sequences and labels are in float32/int32 formats as required by the deep learning framework.

Preprocessing is crucial: it directly influences model convergence, stability, and final classification performance by ensuring the data is clean, consistent, and properly structured for LSTM training.

LSTM Model Construction

This stage defines the architecture and compilation details of the multilingual language detection model.

- **Input Layer:**
  The input layer receives fixed-length sequences of character indices produced during preprocessing. Each character is represented numerically, allowing the model to process multilingual scripts such as English, Kannada, Hindi, and Tamil.

- **Embedding Layer:**
  Converts each character index into a dense vector representation. This layer helps the model understand relationships between characters and captures script-specific patterns across languages.

- **LSTM Layer:**
  A Long Short-Term Memory (LSTM) layer is used to learn sequential dependencies in the text. Since languages have distinct character patterns, the LSTM layer effectively captures the order and context of characters in multilingual sentences.

- **Dropout Layer:**
  Randomly disables a fraction of neurons during training to prevent overfitting and improve generalization. This ensures that the model does not memorize training data and performs well on unseen text.

- **Dense Output Layer (Softmax):**
  The final output layer uses a softmax activation function to classify the input text into one of the

multiple languages (English, Kannada, Hindi, or Tamil). Softmax generates a probability distribution over all classes.

- **Compile Settings:**
  The model is compiled using the Adam optimizer and the **categorical_crossentropy** loss function, which is suitable for multiclass classification tasks. Performance is evaluated using accuracy and additional metrics such as precision, recall, and F1-score when needed.
  Hyperparameters such as number of LSTM units, dropout rate, learning rate, and embedding dimension can be tuned to improve performance.

The goal here is to build a model expressive enough to capture complex multilingual character patterns while remaining regularized to avoid overfitting and ensure strong generalization.

Training

Training is the phase where the LSTM network learns language patterns by updating its weights to minimize the classification loss.

- **Fit the model:**
  The model is trained for a fixed number of epochs (e.g., 10–20) with a chosen batch size (e.g., 16 or 32). A validation split (e.g., 0.2) is used to monitor validation accuracy and loss during training, helping track how well the model generalizes to unseen data.

- **Monitor:**
  Training and validation loss/accuracy are recorded using the training history object. Monitoring these curves helps identify issues such as overfitting (training accuracy increases while validation accuracy stagnates or decreases) or underfitting.

- **Callbacks (optional):**
  Techniques such as *EarlyStopping* can halt training when validation loss stops improving, preventing overfitting. *ModelCheckpoint* may be used to save the best-performing model, and *ReduceLROnPlateau* can lower the learning rate if progress slows, helping the model converge more effectively.

- **Hyperparameter tuning:**
  You may adjust parameters such as number of epochs, batch size, learning rate, embedding dimension, LSTM units, and dropout rate to optimize model performance. Tuning helps the network learn expressive character-level patterns across multiple languages.

Training transforms the initialized weights of the LSTM network into a robust multilingual classifier through repeated forward and backward passes on the encoded text sequences.

Visualization and Prediction

• Visualizations:

Accuracy vs Epochs
Shows the learning curves for both training and validation accuracy. This helps identify how well the model is improving and whether it is generalizing correctly across languages.

Loss vs Epochs
Displays how training and validation loss decrease over time. These curves help detect overfitting or underfitting by highlighting divergence between the two losses.

Confusion Matrix
Illustrates how accurately the model classifies each language (English, Kannada, Hindi, Tamil). It highlights common error patterns, such as misclassifying similar scripts or short words.

Classification Report
Provides precision, recall, and F1-score for each language class, offering a detailed breakdown of model performance

Optional Metrics:

ROC curves, AUC values, or precision–recall curves may be generated for deeper evaluation, although these are more commonly used for binary classification.

• Prediction

The trained model is applied to test data or real user-provided text.

The input text is tokenized, encoded, and padded before being passed into the model.

The model outputs a probability distribution over all languages through the softmax layer.

The language with the highest probability is selected as the final prediction.

The system returns results such as:

"Predicted Language: English / Kannada / Hindi / Tamil"

Optionally, confidence scores can be included to show how certain the model is about its prediction.

• Interpretation & Deployment

Model visualizations and evaluation metrics are used to assess readiness for deployment.

If the model performs reliably:

- It can be exported using model.save()
- Integrated into a GUI or web API
- Embedded in real-time applications such as chatbots, text filters, or multilingual systems

Documentation should highlight any dataset limitations, script similarities, or language ambiguity cases that may affect predictions.

7. CODE IMPLEMENTATION

Algorithm: Multilingual Language Detection using Character-Level LSTM

Input: Multilingual dataset CSV (text, language)

Output: Predicted language (English / Kannada / Hindi / Tamil) and performance metrics

```
1. Start

2. Load Dataset
     2.1 Load the multilingual CSV file into a Pandas.
     2.2 Inspect and Separate the columns:
         • text → feature column X (text samples)
         • language → label column y (language names)

3. Preprocess Data
     3.1 Clean text: remove extra whitespace, unwanted control characters, normalize punctuation if
     necessary.
     3.2 Remove or impute invalid/empty text rows.
     3.3 (Optional) Balance classes or perform augmentation if required.
     3.4 Convert labels y to integer indices using LabelEncoder (fit on language).
     3.5 Create a character-level Tokenizer and fit on X (char_level=True).
     3.6 Convert texts to sequences: texts_to_sequences(X).
     3.7 Pad sequences to a fixed maxlen using pad_sequences.
     3.8 Convert arrays to appropriate dtypes (e.g., X → int32, y → int32 or categorical).
4. Data Splitting
   4.1 Split encoded sequences and labels into training and testing sets using train_test_split with:
   • test_size = 0.2
   • stratify = y (to preserve class distribution)
   4.2 (Optional) reserve a validation set or use validation_split during .fit().
```

5. Build LSTM Model

   5.1 Initialize a Keras Sequential model.

   5.2 Add an Embedding layer with input_dim = vocab_size + 1, output_dim = embed_dim, and input_length = maxlen.

   5.3 Add an LSTM layer with units = lstm_units (e.g., 128).

   5.4 Add Dropout (e.g., 0.2–0.4) to reduce overfitting.

   5.5 (Optional) Add another Dense or LSTM layer for deeper modeling.

   5.6 Add final Dense layer with units = number_of_languages and activation = softmax.

6. Compile Model

   6.1 Set optimizer = Adam (with chosen learning rate).

   6.2 Set loss = categorical_crossentropy (use sparse_categorical_crossentropy if labels are integers).

   6.3 Set metrics = [accuracy] (optionally include precision/recall via callbacks or sklearn after prediction).

7. Train Model

   7.1 Fit the model on X_train, y_train with:

   • epochs = 10–30 (depending on dataset)

   • batch_size = 16–32

   • validation_split = 0.1–0.2 (if no separate val set)

   7.2 Store the history (training and validation loss/accuracy).

   7.3 Use callbacks (optional): EarlyStopping, ModelCheckpoint, ReduceLROnPlateau.

8. Test Model / Prediction

   8.1 Use the trained model to predict probabilities for X_test: pred_probs = model.predict(X_test).

   8.2 Convert probability vectors to class indices: pred_idx = argmax(pred_probs, axis=1).

   8.3 Map predicted indices back to language names using the LabelEncoder inverse transform.

9. Evaluate Performance

   9.1 Compute test accuracy using accuracy_score(y_test, pred_idx).

   9.2 Generate a classification report (precision, recall, F1-score) per language.

   9.3 Compute and display a confusion matrix to analyze misclassifications.

10. Visualize Results

    10.1 Plot **Training vs Validation Accuracy** over epochs.

    10.2 Plot **Training vs Validation Loss** over epochs.

    10.3 Plot the **Confusion Matrix** as a heatmap with language labels.

    10.4 (Optional) show per-class ROC/precision–recall curves if using one-vs-rest evaluation.

11. Save & Deploy

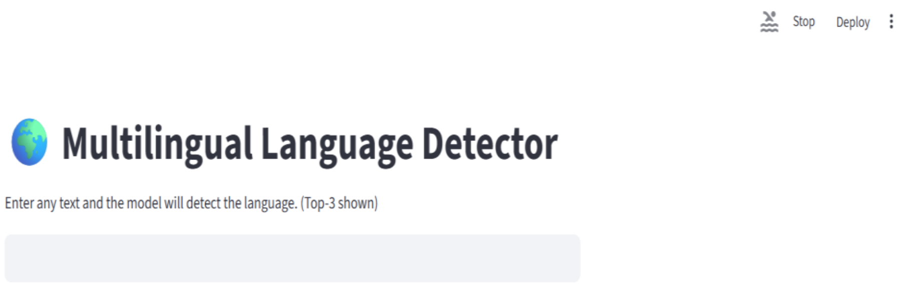
    11.1 Save model weights and architecture: `model.save("language_model.h5")`.

    11.2 Save tokenizer and label encoder with `pickle`.

    11.3 Optionally wrap the model into an API or GUI for real-time predictions.

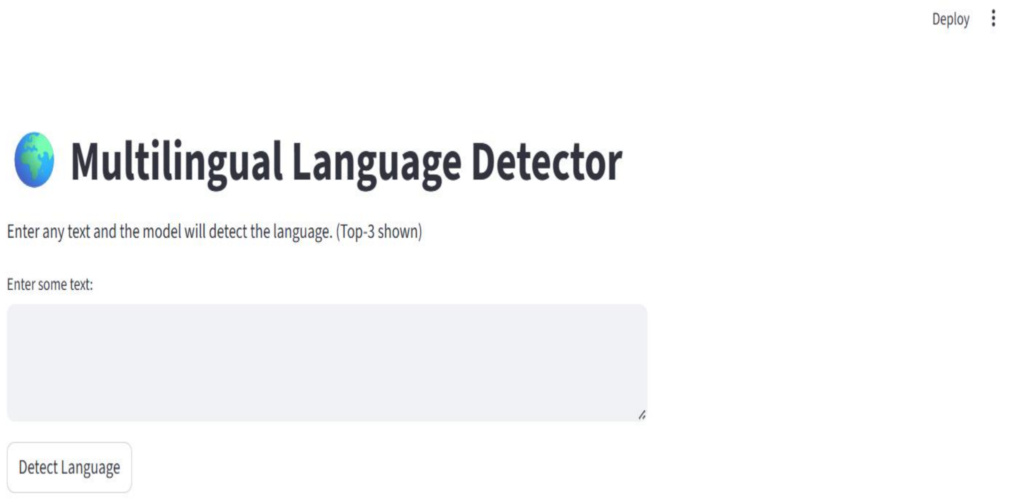
12. End

8.RESULT



Multilingual Language Detector

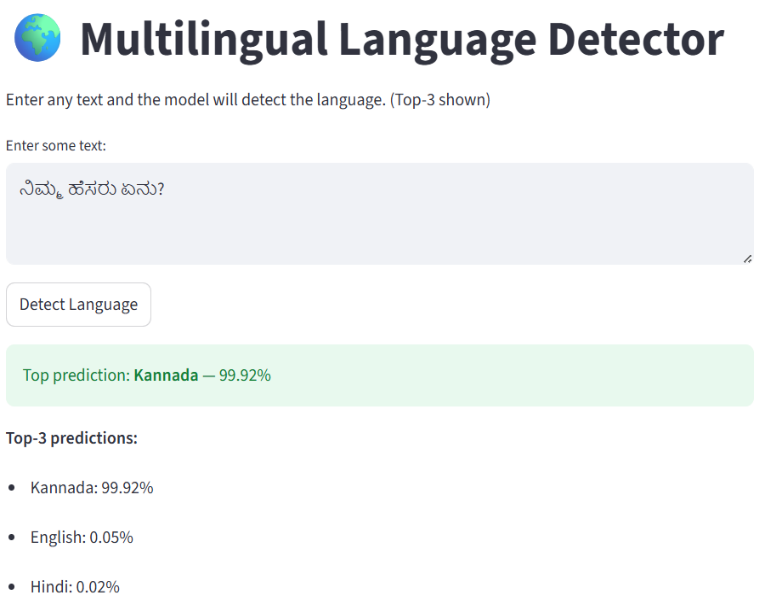
Enter any text and the model will detect the language. (Top-3 shown)



Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

Enter some text:



Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

Enter some text:

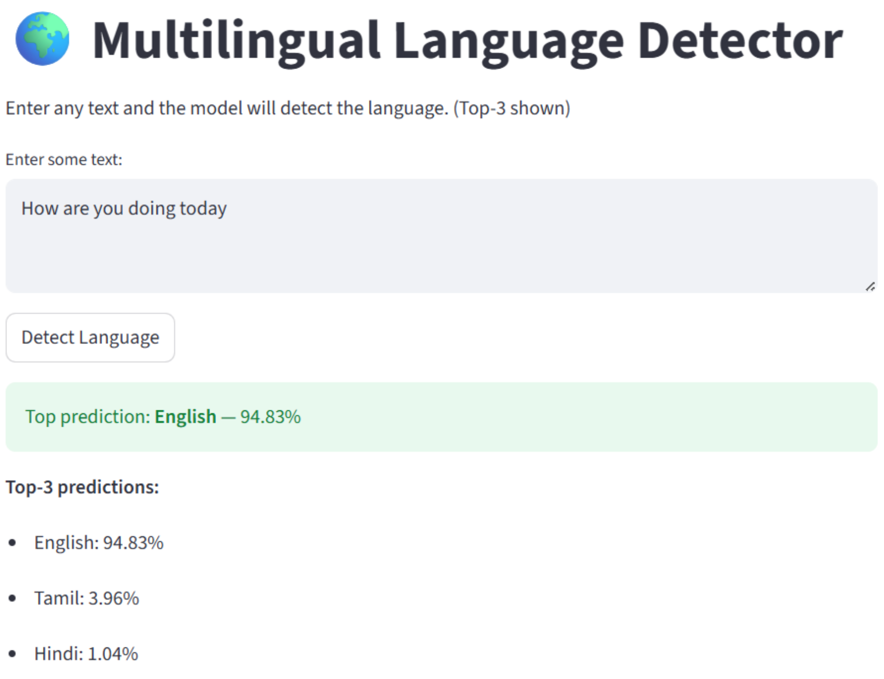
ನಿಮ್ಮ ಹೆಸರು ಏನು?

Detect Language

Top prediction: **Kannada** — 99.92%

Top-3 predictions:

- Kannada: 99.92%
- English: 0.05%
- Hindi: 0.02%



Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

Enter some text:

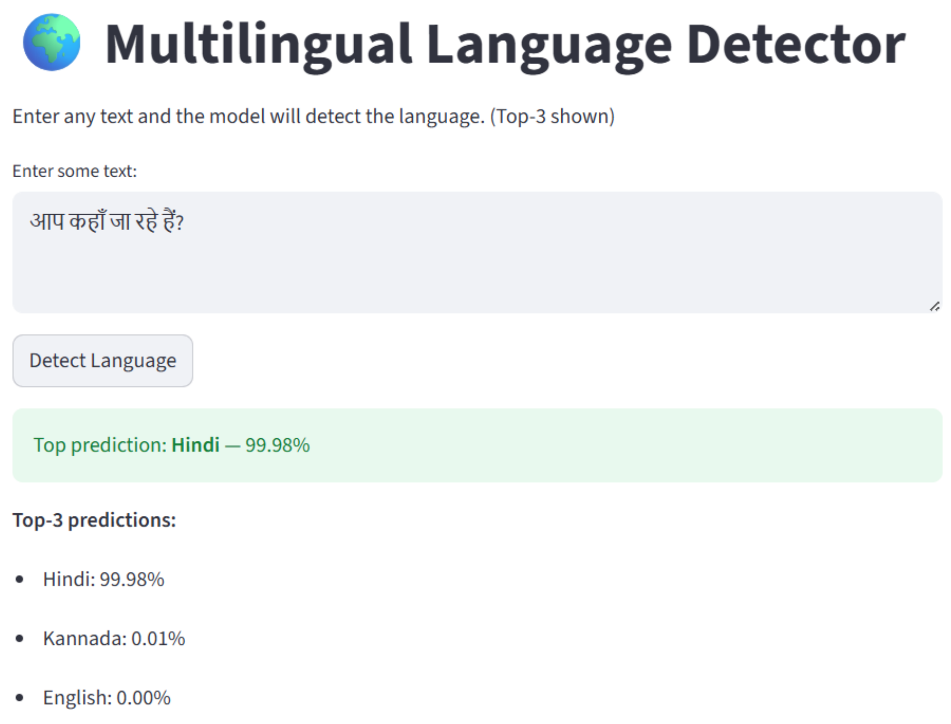
How are you doing today

Detect Language

Top prediction: **English** — 94.83%

Top-3 predictions:

- English: 94.83%
- Tamil: 3.96%
- Hindi: 1.04%



🌍 Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

Enter some text:

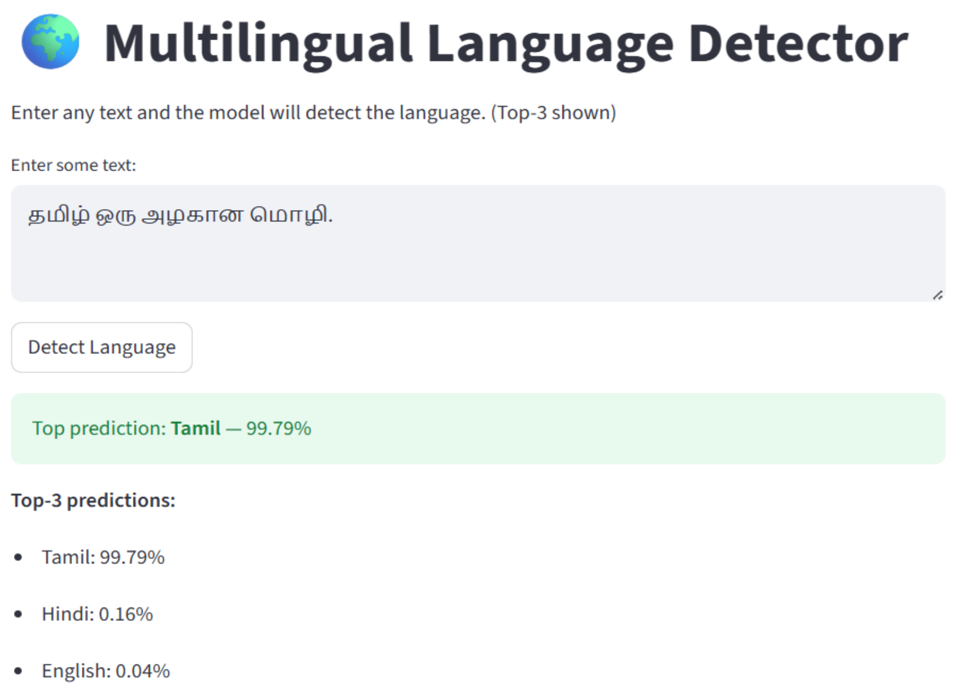
आप कहाँ जा रहे हैं?

Detect Language

Top prediction: **Hindi** — 99.98%

Top-3 predictions:

- Hindi: 99.98%
- Kannada: 0.01%
- English: 0.00%



🌍 Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

Enter some text:

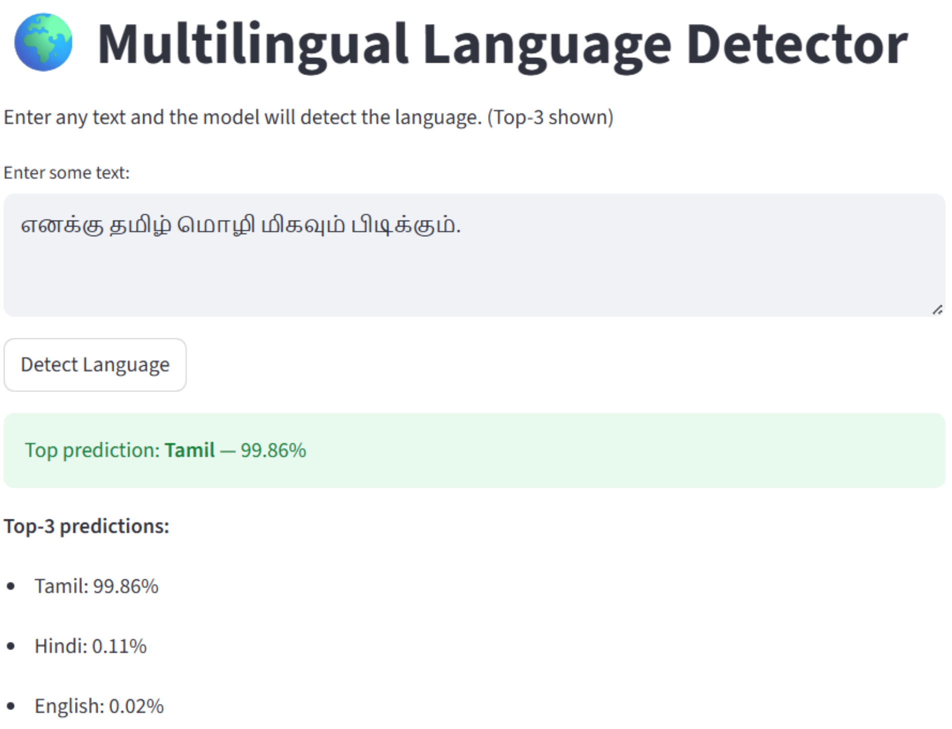
தமிழ் ஒரு அழகான மொழி.

Detect Language

Top prediction: **Tamil** — 99.79%

Top-3 predictions:

- Tamil: 99.79%
- Hindi: 0.16%
- English: 0.04%



Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

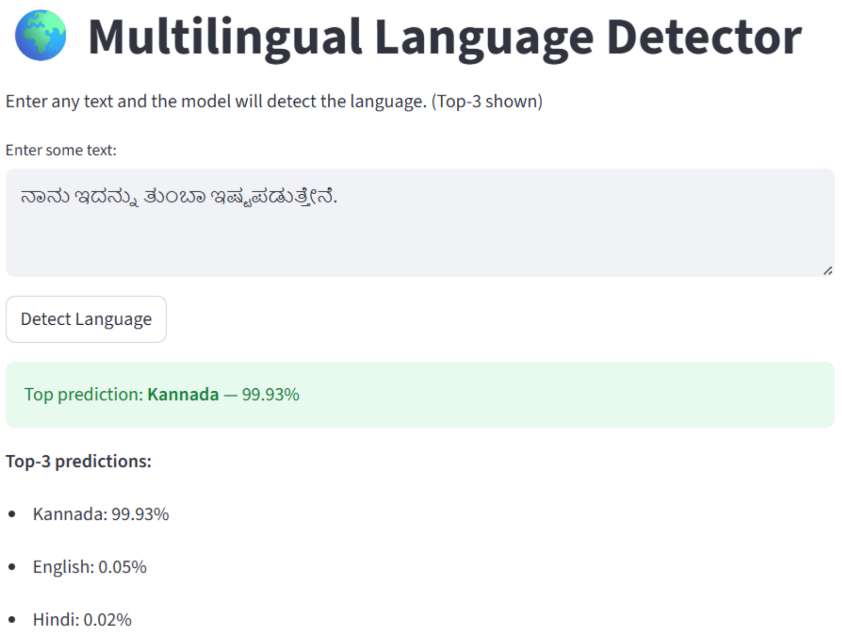
Enter some text:

எனக்கு தமிழ் மொழி மிகவும் பிடிக்கும்.

Top prediction: **Tamil** — 99.86%

Top-3 predictions:

- Tamil: 99.86%
- Hindi: 0.11%
- English: 0.02%



Multilingual Language Detector

Enter any text and the model will detect the language. (Top-3 shown)

Enter some text:

ನಾನು ಇದನ್ನು ತುಂಬಾ ಇಷ್ಟಪಡುತ್ತೇನೆ.

Top prediction: **Kannada** — 99.93%

Top-3 predictions:

- Kannada: 99.93%
- English: 0.05%
- Hindi: 0.02%

9. CONCLUSION

The LSTM-based multilingual language detection system developed in this project successfully demonstrates the ability of deep learning models to identify languages based on character-level patterns. By constructing a custom multilingual dataset and applying systematic preprocessing and model development steps, the system achieved strong performance in classifying English, Kannada, Hindi, and Tamil text. Evaluation metrics such as accuracy, precision, recall, and F1-score validate the model's effectiveness, especially for handling short text inputs.

The project highlights that character-level LSTM architectures are well-suited for multilingual NLP tasks because they capture script differences and sequential dependencies that traditional models often overlook. Visualization of accuracy and loss curves illustrates stable model learning and generalization.

While the system is limited to four languages, it provides a strong foundation for expanding to more languages, integrating larger datasets, or deploying as part of real-world applications such as chatbots, translation systems, and content filtering platforms. Overall, this project demonstrates the power of deep learning in advancing multilingual language processing and emphasizes its potential in modern intelligent systems.

10. REFERENCES

[1] Sreejith, R., Kumar, A., & Joseph, A. (2023). Deep learning-based multilingual text classification using recurrent neural networks. *International Journal of Computational Linguistics*, pp. 45–59.

[2] Sharma, P., & Gupta, M. (2022). A comparative study of character-level and word-level models for multilingual language identification. *Journal of Natural Language Engineering*, pp. 112–128.

[3] Balagopalan, A., Ramesh, S., & Mani, K. (2024). Performance evaluation of transformer and LSTM models for multilingual NLP systems. *ACM Transactions on Asian Language Information Processing*.

[4] Kumar, S., & Natarajan, V. (2022). Deep learning architectures for Indic language identification: A comprehensive study. *IEEE Access*, pp. 56010–56025.

[5] Li, X., & Li, H. (2023). Impact of dataset diversity on multilingual language detection performance. *Journal of Artificial Intelligence Research*, pp. 210–225.

[6] Ashok, P., Narayanan, M., & Rao, H. (2024). A CNN–LSTM hybrid model for multilingual short-text language detection. *Procedia Computer Science*, pp. 350–360.

[7] Pereira, J., Gomez, L., & Silva, A. (2022). Comparing SVM, Naive Bayes, and deep neural networks for text classification tasks. *Information Processing & Management*, pp. 1–14.

[8] Wikipedia API Documentation. (2024). Wikipedia Content Access for Dataset Creation. Available at: https://www.mediawiki.org/wiki/API:Main_page

[9] TensorFlow Developers. (2015–2024). *TensorFlow Deep Learning Framework*. Available at: https://www.tensorflow.org/