Sentence and Word Level Language Identification for Native and Romanized Script Code-Mixed Indian Languages

B. Tech Project Report Submitted
in Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Devika Singh
(210101036)

under the guidance of

Dr. Sanasam Ranbir Singh

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Sentence and Word Level Language Identification for Native and Romanized Script Code-Mixed Indian Languages**" is a bonafide work of **Devika Singh (Roll No. 210101036)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Sanasam Ranbir Singh**

Professor,

Apr, 2026 — Department of Computer Science & Engineering,

Guwahati. — Indian Institute of Technology Guwahati, Assam.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, **Professor Dr. Sanasam Ranbir Singh**, and my mentor, **Saurabh Kumar**, for their invaluable support and continuous guidance throughout this project. Their expertise and encouragement have contributed significantly to my learning and the successful completion of my BTP work. I am truly thankful for their valuable time and the constant efforts they invested in assisting me during this journey.

Abstract

This research addresses the evolving challenges of language identification in Indian languages, focusing on both native and Romanized scripts at the sentence level and word level. With the rise of multilingual content on digital platforms, especially social media, conventional models often struggle due to transliteration, script variation, and limited resources. To tackle this, we present a dataset comprising text in both native and transliterated Latin scripts, annotated with 30 unique language-script label pairs across Indian languages. We introduce a pipeline with sub-models designed for sentence-level language identification for single-language, single-script inputs, capable of handling both native and Romanized text. Building on this, we extend our work to word-level tagging for Romanized code-mixed Indian languages, which present unique issues like script mixing, borrowed vocabulary, and homographs. This second model focuses on accurate word-level tagging in Latin-script, multi-language, code-mixed sentences. This BTP contributes toward robust language identification in diverse and low-resource multilingual environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Language identification for both native scripts and transliterated (Romanized) text is a foundational task in multilingual natural language processing (NLP), especially in linguistically diverse regions like India. With 22 scheduled languages and over 735 million internet users, digital communication in India frequently features not only multiple languages but also a heavy use of code-mixing and transliteration. Social media, messaging platforms, and other informal digital spaces are rich with content where users blend languages—often using a single script, such as the Latin script—creating complex, code-mixed text. These patterns pose unique challenges for language technology systems.

In this work, we explore sentence-level and word-level language identification in parallel, recognizing that both are essential and tightly coupled. Accurate sentence-level language identification serves as a foundation for more granular analysis. Identifying the dominant language of a sentence helps in segmenting multilingual content and enabling context-aware downstream NLP tasks such as machine translation, sentiment analysis, and information retrieval. However, sentence-level detection alone often falls short in handling highly code-mixed or transliterated text, where multiple languages co-exist within a single sentence.

To capture the full linguistic nuance of such content, we progressively move towards word-level language identification, which is essential for parsing and understanding code-mixed and transliterated inputs. This finer granularity allows models to discern subtle language shifts, enabling more precise semantic interpretation and supporting robust applications like entity recognition and intent detection.

Despite the clear need, current data resources remain sparse and uneven, for both native and transliterated text, particularly for low-resource Indian languages. Many existing datasets are outdated, lack diversity, or are skewed toward a small subset of languages. This data scarcity is even more pronounced for Romanized variants, which dominate user-generated content but are underrepresented in research corpora. Furthermore, state-of-the-art language identification models often struggle with real-world code-mixed data, limiting their effectiveness in multilingual environments.

1.1 Challenges

1.1.1 Dataset Creation - Native and Latin Transliterated

- **Dataset Size and Coverage:** Ensuring a minimum of 10,000 sentences per language posed a significant challenge, particularly for low-resource languages where digital text availability is limited.

- **Sentence Length Constraints:** To improve model performance, only sentences containing at least three words were included. While this helped reduce noise and improve accuracy of model trained on the dataset, it also led to the exclusion of many shorter yet valid sentences. As a result, the already scarce publicly available datasets became even smaller, limiting linguistic diversity and reducing coverage of natural language variability.

- **Validation and Quality Assurance:** Although partial human validation was performed via random sampling, followed by native speaker verification, comprehensive manual validation across the entire dataset is still pending. This leaves room for potential errors or inconsistencies, especially in languages with complex orthography.

- **Scalability of Human Involvement:** The involvement of native speakers in the verification process, while valuable, is not easily scalable and highlights the need for more automated yet reliable validation methods in future iterations.
- **Consistency Across Languages:** Maintaining consistency in data formatting and transliteration rules across multiple languages with different scripts and phonetic systems added another layer of complexity to the data preparation process.

1.1.1.1 Latin Transliterated Dataset

- **Script Ambiguity:** Many native scripts have characters that map to multiple possible Roman representations, especially when phonemes do not have direct equivalents in the Roman alphabet. For example, in Hindi, the same native word translating to 'yes' may be transliterated in Latin script as *han*, *hn*, or *haa*, depending on personal preference or phonetic interpretation. This variability makes it challenging to create consistent transliteration mappings and complicates both model training and evaluation.
- **Limited Resources:** There is a general scarcity of publicly available transliterated datasets for Indian languages in Latin script, particularly for low-resource languages. This made the task of data collection and validation both time-consuming and resource-intensive.
- **Noisy User-Generated Data:** When using data from informal sources like social media, blogs, or forums, Romanized text is often inconsistent, non-standard, and filled with typos or code-mixing, increasing preprocessing complexity.
- **Data Generation for Low-Resource Languages:** For many low-resource languages, publicly available transliterated datasets in Latin script are either

very limited or nonexistent. As a result, we had to rely on native script datasets and generate transliterations. This introduces potential errors, especially when there is no established transliteration standard or sufficient linguistic resources for validation.

1.1.2 Code-mixed dataset

- **Tool Limitations in Code-Mixed NLP:** Existing NLP tools often struggle to handle the complexities of code-mixing, especially in downstream applications [SS21].
- **Data Scarcity in Formal Sources:** Despite the widespread use of code-mixed language, there is a noticeable gap in the data available for studying it further. This is primarily due to the scarcity of code-mixed content in formal texts found online, as these are typically more standardized and less reflective of the mixed-language usage seen in casual or informal settings.
- **Privacy Constraints on Data Collection:** A significant challenge in generating code-mixed datasets is the privacy restrictions imposed by many online platforms, which prevent data scraping or usage. This limitation forces reliance on platforms like Twitter, where code-mixing is more prevalent among bilingual and multilingual users, and the data is publicly available for analysis.
- **Limitations of Synthetic Datasets:** Although studies show that training on code-mixed sentences leads to better results compared to using separate monolingual corpora, most existing code-mixed datasets are synthetically generated. This synthetic nature may limit their ability to fully capture the complexities of natural code-mixing.
- **Spelling Variability in Mixed-Script Text:** In mixed-script code-mixing, spelling inconsistencies often arise when languages are written in non-native scripts using phonetic transliterations, leading to variations that complicate accurate language processing.

- **Bias Toward High-Resource Languages:** While pre-trained multilingual models perform well for high-resource languages, they struggle with low-resource languages. This is due to the limited availability of pre-training data for these languages in large multilingual corpora, which is often dwarfed by the volume of data available for high-resource languages like English. Moreover, evaluation benchmarks often underrepresent low-resource languages, exacerbating the problem.

1.1.3 Borrowed words

In mixed-script, code-mixed text, borrowed terms introduce a distinct challenge. These are words that are taken from one language and incorporated into another, often due to their cultural or contextual significance. For example, the term कंप्यूटर (*Translation: computer*) is borrowed from English into Hindi. The presence of borrowed words complicates language identification systems, as it becomes difficult to determine whether these words should be categorized under the "borrowing" language or the "native" language from which the word originates.

1.1.4 Homographs

Code-mixed text can also feature words that are spelled identically across different languages but have divergent meanings. This can lead to confusion for word-level language identification tools, which must rely on context to correctly assign the word to the appropriate language.

The car is parked outside.

Ham ja rahe the. (*Translation: "We were going."*)

Figure 1.1: Example of homographs in code-mixed text, showing the as an English article and the in Hindi meaning "were".

1.1.5 Sentence-level Language Identification

- **Multiple Native Scripts per Language:** Some languages, such as Manipuri, Kashmiri, and Sindhi, are used in more than one native script. Our dataset includes multiple script variants for these languages to reflect real-world usage.

- **Script Ambiguity:** Some scripts (e.g., Devanagari, Bengali) are visually similar and share characters, increasing the risk of misclassification. To mitigate this, we rely on Unicode ranges rather than visual features for script identification.

- **Multi-script Sentences:** A single sentence may contain words in multiple scripts (e.g., Devanagari and Latin). To handle this, we identify the dominant script using a percentage threshold based on character count and perform language identification based on dominant script.

1.1.6 Word-Level Language Identification

- **Code-Mixing with Multiple Languages:** Code-mixed text often contains words from more than one language, making it essential to identify the language at the word-level rather than assuming a single language for the entire sentence.

- **Proper Nouns and Universal Tokens:** The presence of proper nouns, hashtags, URLs, emojis, and other universal tokens introduces ambiguity as they are not tied to any specific language. These tokens must be correctly identified and tagged with a special label (e.g., `univ`) to avoid misclassification to a language.

- **Transliteration Variability:** Romanized words often lack consistent spelling due to phonetic typing, which makes it difficult to match them to standardized language dictionaries or embeddings.

- **Ambiguous Words:** Homographs belonging to multiple languages, require context-aware models.

- **Lack of Annotated Data:** Word-level annotated datasets for code-mixed languages are scarce, especially for low-resource languages, which limits the ability to train supervised models effectively.
- **Unseen Vocabulary:** Due to the informal and creative nature of code-mixed communication, models must handle out-of-vocabulary words and non-standard spellings gracefully, possibly using subword units or character-level representations.

1.2 Building a Pipeline for Sentence-Level Language Identification

We designed a pipeline that identifies the language of a sentence by first detecting the dominant script using Unicode-based frequency analysis. Based on the detected script, the model selects a corresponding fastText-based sub-model to predict the language. This approach supports 30 native script–language pairs and 22 Romanized language versions. While it does not handle non-native script–language combinations, it performs well for valid script–language pairs and enables efficient sentence-level language identification.

1.3 Building a General-Purpose Model for Word-Level Language Identification

For developing and optimizing a general-purpose model for word-level language identification, we begin by training on publicly available datasets for Hindi [BJPD22] and Telugu [BC20]. Our approach combines sentence-level and word-level models: we first apply the sentence-level model to detect the dominant language of a code-mixed sentence. Based on this prediction, we route the input to a corresponding word-level sub-model trained specifically for English mixed with that identified language. This modular design allows us to reuse the sentence-level prediction to guide fine-grained word-level tagging. Our goal is to eventually generalize this architecture for other language pairs as more annotated data becomes available.

1.4 Our Contributions

We construct a comprehensive dataset for native-script and Romanized version of Indian languages by aggregating and augmenting existing public datasets. This results in a larger corpus than any currently available resource for sentence-level language identification. We develop a sentence-level language identification pipeline. We build a word-level language identification model designed for code-mixed text, particularly for English and Indian language combinations written in Roman script.Our work contributes toward improving multilingual language identification in low-resource, code-mixed, and informal settings—common across Indian social media and online communication.

मुझे कॉफी पसंद है | **Mujhe coffee pasand hai**

(a) (b)

Output: Hindi and Devanagari | **Output:** Hindi and Latin

(c) (d)

$$\begin{bmatrix} (\text{Mujhe}, \text{Latin\_Hindi}), & (\text{coffee}, \text{Latin\_English}), \\ (\text{pasand}, \text{Latin\_Hindi}), & (\text{hai}, \text{Latin\_Hindi}) \end{bmatrix}$$

(e)

Figure 1.2: Example of sentence-level and word-level language identification for both native script Hindi and Romanized Hindi sentences. (a) Native script sentence; (b) Romanized transliterated sentence; (c) Language and script output for native sentence; (d) Language output for transliterated sentence; (e) Word-level Tagged output with language and script labels.

1.5 Organization of The Report

This report is organized as follows. Chapter 1 introduces the motivation for our work, outlining the key challenges in language identification across native-script, Romanized, and code-mixed text in the Indian context. It discusses the creation of our datasets, the approaches taken for sentence-level and word-level language identification, and summarizes our contributions. Chapter 2 provides background on code-mixed language usage, language identification, and tagging techniques relevant to our study. Chapter 3 surveys related work, including existing datasets such as Aksharantar, Dakshina, and IndicXlit, and tools like fastText, along with efforts by initiatives like AI4Bharat. Chapter 4 presents our proposed approach in detail, covering dataset construction and the development of models for native script, transliterated text, and code-mixed scenarios. Chapter 5 describes the experimental setup, including model parameters, training details, and evaluation strategies, as well as ablation studies. Chapter 6 reports and analyzes the results obtained for all three dataset types. Finally, Chapter 7 concludes the report and discusses possible directions for future work.

Chapter 2

Background

2.1 Data

2.1.1 Code - Mixed data

Code-switching or Code-mixing is the blending of linguistic elements from multiple languages within a single conversation, and often even within a single sentence. While initially observed in casual, spoken interactions, the rise of social media has extended this phenomenon to digital spaces, particularly on forums and messaging platforms. This increased visibility has led to a growing body of research in code-mixed natural language processing (NLP), as the prevalence of mixed-language data on social media and chat applications continues to grow.

2.1.1.1 Single-Script Code-Mixing

At sentence level, this approach to code-mixing involves using a single script, often with elements from different languages.

Mujhe coffee bohot pasand hai.

Translation: "I really like coffee."

Figure 2.1: An example of single-script code-mixing in Romanized Hindi-English (Hinglish), where both Hindi and English words use the Roman(Latin) alphabet.

2.1.1.2 Mixed-Script Code-Mixing

In mixed-script code-mixing, languages with different native scripts are combined in a sentence. This phenomenon, known as Intra-sentential script-mixing, reveals patterns in how scripts and languages interact in multilingual communities.

मुझे coffee बहुत पसंद है।

Translation: "I really like coffee."

Figure 2.2: An example of mixed-script code-mixing with Hindi in Devanagari and English in Roman(Latin) script.

This type of mixing is common in code-mixed text where speakers blend languages fluidly within a single thought or phrase. All permutations of scripts and languages can be observed in mixed-script code-mixing sentences.

2.2 Language identification

Language identification is the task of identifying the language of a given input at the sentence or word levels. At the sentence level, the sentence is classified into a single language, whereas, at the word level, each word in a sentence is tagged with its respective language.

Sentence-level:

"I love coffee." [*English*]

Word-level:

Mujhe[*Hindi*] coffee[*English*] pasand[*Hindi*] hai[*Hindi*]

Figure 2.3: Examples of language identification at sentence-level and word-level.

2.3 Language Tagging

Language tagging in NLP is the task of detecting and labelling the language for each word or section in a text. Language tagging shares similarities with Named Entity Recognition (NER), a well-researched area. In NER, named entities in the text are identified and classified into predefined categories. Similarly, language tagging assigns each word a language label from a set of supported languages. Both tasks require contextual understanding: just as NER must consider the surrounding words to label an entity correctly, language tagging relies on context to accurately assign language labels, handling nuances such as borrowed words and homographs.

2.4 Conclusion

This chapter introduced the foundational concepts relevant to our work, starting with an overview of code-mixing in multilingual contexts. We distinguished between single-script and mixed-script code-mixing, illustrating how multiple languages—especially Indian languages and English—are combined at both the script and sentence levels. We then explored the tasks of language identification and language tagging, with examples highlighting the differences between sentence-level and word-level tagging. Emphasis was placed on the role of context in language tagging, particularly in handling borrowed words and ambiguous tokens. This background establishes the linguistic and computational challenges addressed in our work on Romanized and native-script multilingual datasets.

Chapter 3

Related Works

With the growth of social media, large amounts of code-mixed data have become available, creating new research opportunities in NLP. Social media platforms show great language diversity, as users often mix different languages in a single sentence or phrase. This unique setting has led recent NLP research to focus on processing code-mixed data. Recent studies in this area can be classified into three main categories.

3.1 Dataset Resources

Twitter is a popular source for code-mixed data in Indic languages, as many bilingual and multilingual users code-mix on the platform [CWT13, SBM$^+$14, JTJ17, RSC$^+$17]. Indic Wikis are also used as a pre-training resource due to their quality, though they are sparse. Other sources, such as CC100 [CKG$^+$20] and mC4 [XCR$^+$21], provide larger datasets but are often noisy and may contain inappropriate content [KCW$^+$22]. IndicCorp v1 [KKG$^+$20] is the first dataset specifically created for Indic languages.

3.1.1 Aksharantar

Aksharantar is a publicly available transliteration dataset containing 26 million word pairs across 20 Indic languages [AI422] at the time of writing (5 May 2022). The dataset consists of parallel word-level data, mapping words in native scripts to their transliterated forms in the Roman (Latin) script. However, it does not cover all 22 scheduled Indian languages—most notably, it lacks data for Santhali. While Aksharantar includes data for Dogri added later(apart from the 20 Indic languages), IndicXlit, a model trained on this dataset does not provide support for it.

Furthermore, Aksharantar does not provide data for language identification at the sentence level, which means that it does not capture syntactic or contextual relationships between words. This limits its utility for tasks that rely on sentence structure to detect language. Additionally, transliteration, especially in informal contexts like social media, often varies depending on the writer's interpretation and style, making consistent mapping challenging.

As a result, while Aksharantar is valuable for generating additional transliterated data, its use is constrained to word-level augmentation for only the 21 languages it covers.

3.1.2 Dakshina

The Dakshina dataset [RHS+20] provides text in both Latin and native scripts for 12 South Asian languages on sentence-level. It includes a large corpus of native script Wikipedia text, a romanization lexicon consisting of words in the native script with attested romanizations, and a subset of full-sentence parallel data in both the native script and the basic Latin alphabet. But Dakshina covers only 12 languages, whereas our dataset provides broader language coverage.

3.1.3 IndicXlit

IndicXlit is a transformer-based multilingual transliteration model (11M parameters) that supports 21 Indic languages for both Roman-to-native and native-to-Roman script conversions. It is trained on the Aksharantar dataset [AI422].

IndicXlit supports the 20 Indic languages covered by Aksharantar, plus Sinhala. It handles transliteration between languages in different scripts, providing flexibility in cross-script text conversion. However, it should be noted that the model does not support Dogri, despite its inclusion in the Aksharantar dataset, as it was added later. This model is particularly useful for tasks that involve transliterating informal, social media-style text, where variations in spelling and style are common.

3.2 Language identification

A reliable language identifier is essential for building corpora in low-resource languages, where challenges like noisy web data, small datasets, and similarity to high-resource languages make language identification difficult [C+20]. Existing tools like CLD3 [AI16], LangID [LB11], fastText [JGBM16], and NLLB [N+22] lack full support for all 22 official Indian languages and struggle to detect romanized text, except for CLD3's[AI16] limited support for Latin Hindi. The performance of general-purpose multilingual models like mBERT [DCLT19], mT5 [XCR+21], and XLM [CL19] trained on major Indic languages is limited by the need to share capacity with high-resource languages [CKG+20, K+22]. Indic-specific models, such as MuRIL [K+21] and IndicBERT v1 [KKG+20], achieve better results on Indian language tasks.

3.2.1 fastText

FastText is an efficient text classification tool designed to handle large-scale datasets effectively. It uses character n-gram features to capture subword information, which is especially useful for processing rare words. This approach also helps distinguish between languages with similar spellings, enhancing its performance in multilingual tasks. FastText incorporates n-gram features, dimensionality reduction, and an optimized version of the softmax classifier, making it fast and lightweight. Our current work employs fastText for language identification, leveraging its ability to accurately classify text at the sentence level across multiple Indic languages. These qualities make fastText well-suited for applications requiring scalable and resource-efficient text classification.

3.3 AI4Bharat

Ai4Bharat, a research lab dedicated to advancing AI resources for Indian languages, has developed and released an extensive collection of datasets, tools, and state-of-the-art models. Among these, two resources central to this report are IndicCorp and Bhasha-Abhijnaanam[MKK23].

3.3.1 IndicCorp

IndicCorp provides a large sentence-level monolingual corpus covering 11 languages from the Indo-Aryan and Dravidian language families, along with Indian English. This corpus is, on average, nine times larger than the OSCAR dataset, offering significant data resources for these languages.

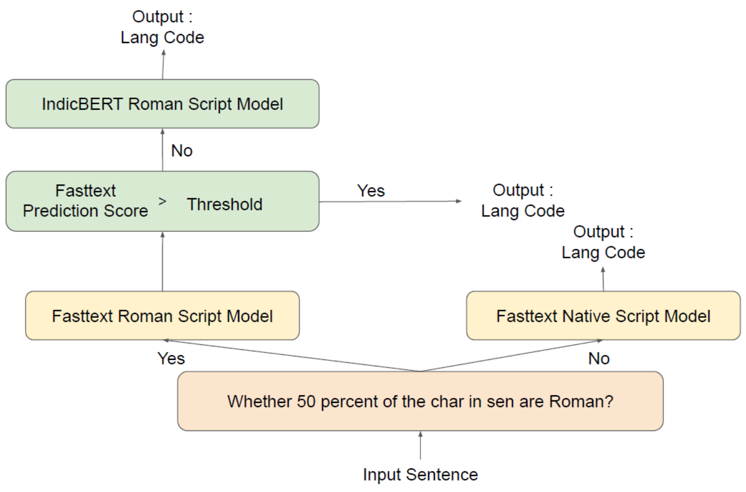
3.3.2 Bhasha-Abhijnaanam

The Bhasha-Abhijnaanam[MKK23] dataset and models focus on language identification (LID) for native-script and romanized text in 22 Indic languages. The dataset includes labelled data in both native and romanized scripts and contains clean sentences (grammatically correct, single script, etc.), which enhances its application for various NLP tasks in Indic languages. Bhasha-Abhijnaanam provides three LID models:

- Linear fastText-based model: This efficient model is split into two versions—one trained on native script data and the other on romanized script data. It utilizes character n-gram features, which help distinguish languages with similar spelling patterns.

- BERT-based model: This model, based on pre-trained BERT architecture, offers higher accuracy for LID tasks, albeit with a slower processing speed due to its complexity.

- Hybrid model: This model combines the strengths of both fastText and BERT-based approaches. In this model, the fastText classifier initially processes input for quick classification. If the fastText model's confidence is low, the BERT-based model is used, allowing a balance between computational efficiency and accuracy.



Output : Lang Code
IndicBERT Roman Script Model
No
Fasttext Prediction Score > Threshold
Yes
Output : Lang Code
Output : Lang Code
Fasttext Roman Script Model
Fasttext Native Script Model
Yes
No
Whether 50 percent of the char in sen are Roman?
Input Sentence

Figure 3.1: Hybrid IndicLID Classifier Workflow (referenced from [MKK23])

This structured approach allows Bhasha-Abhijnaanam to support reliable and accurate language tagging across Indic languages for single script scenarios, covering both native and romanized scripts effectively.

3.4 BiLSTM

Bidirectional Long Short-Term Memory (BiLSTM) networks are an extension of traditional LSTMs that process input sequences in both forward and backward directions, enabling the model to capture both preceding and succeeding contextual information [GS05]. This characteristic makes BiLSTMs particularly well-suited for word-level language identification in code-mixed text, where the language of a given word often depends on the surrounding context. In our work, we leverage BiLSTM-based sequence tagging models to perform fine-grained language tagging at the word level for Hindi-English and Telugu-English code-mixed sentences written in Roman script. The ability of BiLSTM models to incorporate bidirectional context significantly enhances their effectiveness in disambiguating short or ambiguous words and handling linguistic irregularities commonly found in code-mixed data.

3.5 Conclusion

This chapter reviews existing resources, tools, and models relevant to code-mixed data processing and language identification for Indic languages. A variety of distributed resources for NLP have been developed, notably by Ai4Bharat and other research initiatives, to support language tagging, classification, and resource building for low-resource Indian languages.

Chapter 4

Proposed Work

The aim of this BTP is to create a comprehensive dataset containing both native script and transliterated Romanized single-script text across 22 Indic languages, annotated with sentence-level language tags. Additionally, the project seeks to develop two separate sentence-level language identification models—one for each dataset type—and a third model for Romanized code-mixed text in Indian languages. This final model will perform word-level language tagging and serve as a general solution applicable across all the target languages.

4.1 Dataset

4.1.1 Native Script Dataset

The dataset contains sentences in multiple Indic languages, each labeled with one of 30 distinct language-script pairs for sentence-level language identification. All 22 Indian languages recognized in the Constitution are included in their native scripts.

Kashmiri is represented in both Arabic and Devanagari scripts, with more data available in Arabic. Manipuri is represented in both Bengali and Meetei Mayek, with more data in Bengali. Sindhi is represented in both Arabic and Devanagari, with more data in Arabic.

Additionally, widely spoken languages such as Awadhi, Bhojpuri, Magadhi, and Mizo—though not part of the 22 constitutionally recognized languages—are also included.

We were able to gather substantial data for traditionally low-resource languages like Mizo, Bhojpuri, and Manipuri. However, languages such as Awadhi, Bodo, Dogri, Kashmiri(in script Devanagari), Konkani, Magadhi, Maithili, Sindhi(in script Devanagari), Manipuri(in script Meetei Mayek), and Santali fall short of the 10,000-sentence target due to limited publicly available resources in native script.

For the remaining languages, we surpassed the 10,000-sentence benchmark, improving upon datasets like Dakshina (which includes 11 languages) by consolidating various publicly available sources and cleaning them to create a standardized dataset across all language-script pairs. Overall, the dataset spans 27 languages across 13 scripts, with all sentences containing more than three words.

From the initial dataset collected from publicly available and verified sources, we performed cleaning and formatting to convert the data into our standardized CSV format, with columns: `sentence` and `lang_tag`. Using this, we trained three separate fastText-based models for Arabic, Bengali, and Devanagari scripts. Since English language data is in Latin script it dealt with in the Transliterated Roman Script model. For each script, we first combined the data from all languages written in that script and then split the consolidated dataset into training (72%), validation (8%) and testing (20%). A model was trained on the training data using supervised learning using `fasttext.train_supervised`.

This model was then used to predict language tags for unlabeled or low-confidence sentences identified as being in this script (using Unicode-based detection) and likely belonging to one of the target languages for this script. If the model predicted a language with over 90% confidence, the sentence was assigned that label. As part of the final cleaning step, we removed all sentences with three or fewer words.

Finally, a new model was trained using both the original and newly labeled data, with a fresh split of training (72%), validation (8%), and testing (20%).

4.1.1.1 Arabic script

Code	Sentences (Initial)	Sentences (Final)	Language
urd	12164	12164	Urdu
kas	997	194794	Kashmiri
snd	997	274062	Sindhi

Table 4.1: Sentence counts for Arabic script languages before and after dataset expansion

4.1.2 Bengali Script

Code	Sentences (Initial)	Sentences (Final)	Language
ben	30580	427408	Bengali
asm	10728	158428	Assamese
mni	27673	140772	Manipuri

Table 4.2: Sentence counts for Bengali script languages before and after dataset expansion

4.1.3 Devanagari Script

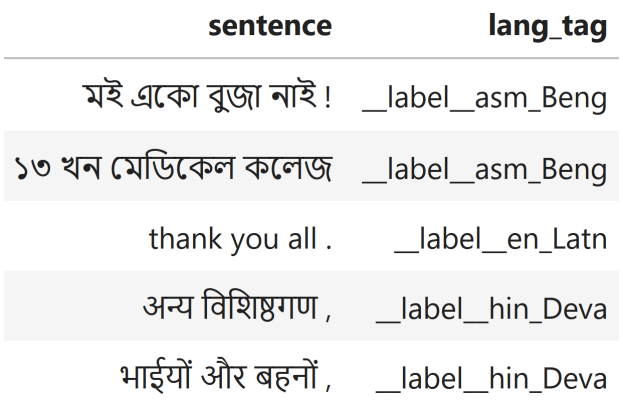
Code	Sentences (Initial)	Sentences (Final)	Language
san	208206	237432	Sanskrit
npi	80164	85580	Nepali
hin	57828	292381	Hindi
bho	47210	479970	Bhojpuri
mar	37128	38185	Marathi
awa	997	1085	Awadhi
brx	997	997	Bodo
dgo	997	1025	Dogri
gom	997	1010	Konkani
hne	997	1029	Chhattisgarhi
kas	997	997	Kashmiri
mag	997	1001	Magahi
mai	997	1005	Maithili
snd	997	1013	Sindhi

Table 4.3: Sentence counts for Devanagari script languages before and after dataset expansion

.

Language	**Script**	**Native**	**Latin**
Kashmiri	Arabic	196291	30437
Sindhi	Arabic	27048	12288
Urdu	Arabic	12164	12054
Assamese	Bengali	105509	24556
Bengali	Bengali	631860	13366
Manipuri	Bengali	68571	38164
Awadhi	Devanagri	1071	-
Bhojpuri	Devanagri	351657	-
Bodo	Devanagri	997	1429
Dogri	Devanagri	1015	-
Hindi	Devanagri	178295	13367
Kashmiri	Devanagri	1017	30437
Konkani	Devanagri	1009	1441
Magadhi	Devanagri	1004	-
Maithili	Devanagri	3520	10438
Marathi	Devanagri	372747	12340
Nepali	Devanagri	128623	10465
Sanskrit	Devanagri	182885	17518
Sindhi	Devanagri	1012	12288
English	English	55887	55887
Mizo	English	195184	237168
Gujarati	Gujarati	842348	13090
Punjabi	Gurmukhi	35052	12440
Kannada	Kannada	35784	13131
Malayalam	Malayalam	34221	11799
Manipuri	Meetei Mayek	997	38164
Santali	Ol Chiki	9094	-
Odia	Odia	38936	10423
Tamil	Tamil	40098	12463
Telugu	Telugu	41261	11847

Table 4.4: Languages, their Scripts, and Samples in Native and Latin Scripts



sentence	lang_tag
মই একো বুজা নাই !	__label__asm_Beng
১৩ খন মেডিকেল কলেজ	__label__asm_Beng
thank you all .	__label__en_Latn
अन्य विशिष्ठगण ,	__label__hin_Deva
भाईयों और बहनों ,	__label__hin_Deva

Figure 4.1: Snapshot of the Native Script dataset.

4.1.4 Transliterated Latin dataset

The Latin script dataset was constructed using data collected from social media, web crawling, Wikipedia, and language-specific transliteration websites, along with consolidating publicly available datasets. Data from the Dakshina dataset was also adapted to our format and incorporated into the Latin script corpus. Sentence-level test data from the Aksharantar[AI422] was also incorporated into the dataset for the Latin script model.

In addition to these sources, for languages with limited available resources, we utilized IndicXlit [AI422] to transliterate native script data into Roman script. This approach was applied to Assamese, Kashmiri (from Arabic script), Maithili, Nepali, Malayalam, Manipuri, Oriya, and Sanskrit to ensure that the dataset size exceeded 10,000 samples for these languages. However, transliterated data for Santali and Dogri was not found, and Bodo and Konkani had significantly lower data availability.

To ensure consistency in Latin-script text, especially for languages with accented characters or diacritics, we applied Unicode-based normalization. Specifically, we used NFD (Normalization Form Decomposition) to decompose characters into their base forms and remove diacritical marks. For example, the text `"thugāire éçöl"` is normalized to `"thugaire ecol"`. This preprocessing step was applied to Latin-script data to remove diacritical marks using Unicode-based normalization. We found that only Manipuri and Mizo contained words affected by this process, while the rest of the languages did not include any characters requiring normalization. This step ensured consistent representation and improved model.

4.1.5 Romanized code-mixed dataset

The Hindi-English L3Cube-HingCorpus [BJPD22] and the Telugu-English sentiment corpus [BC20] are annotated with word-level language tags, where the primary language is Hindi or Telugu and the secondary language is English. Both are written in Latin script. These word-level annotations enable precise language identification at the individual word level, critical for code-mixed language processing.

These datasets include linguistic annotations, such as special markers for tokens like proper nouns, which enable fine-grained analysis of code-mixed text. Additionally, they preserve sentence structure, allowing models to utilize contextual information within a sentence to more accurately identify the language of individual words.

Category	**Count**
Hindi	44,452
Telugu	19,752
Total	64,204

Table 4.5: Language Item Counts in the Romanized Dataset

Tag	**Description**
`en`	English word
`univ`	Universal token (punctuation, numbers, emojis, etc.)
`te`	Telugu word
`ne`	Named entity (proper nouns such as names of people, places, organizations)

Table 4.6: Tags used for word-level annotation in the Telugu-English code-mixed dataset

Tag	Description
HI	Hindi word
EN	English word

Table 4.7: Tags used for word-level annotation in the Hindi-English code-mixed dataset

4.2 Models

This BTP began with an in-depth study of language identification for Indian languages in both native and Latin scripts. We initially developed sentence-level language identification models using fastText-based linear classifiers, creating separate models for Devanagari, Bengali, and Arabic scripts. Building on this, we explored Unicode-range-based preprocessing techniques to construct a sentence-level general-purpose language identifier applicable across all scripts. Finally, we extended our work to word-level language tagging, focusing on methods to handle intra-sentential script and code-mixing.

4.2.1 Sentence-Level Language Identification

4.2.1.1 Native Script-Specific Models

For Arabic, Bengali, and Devanagari scripts, we trained separate script-specific models for sentence-level language identification. Each model was trained exclusively on a subset of the Native Script Dataset. These contain sentences consisting of words from a single language and having a dominant script matching the target script of the model. Some languages (Manipuri, Kashmiri and Sindhi) appear in multiple models because they are written in more than one script. While some sentences may include borrowed words, such instances are minimal and do not significantly affect language identification performance.

4.2.2 Latin Script Model

In this model, we use the fastText [BGJM17] model to train a sentence-level language identification model for Indian languages written in Latin script. The dataset includes sentences across 22 Indian languages we have in the Latin Script Dataset, ensuring robust coverage. Since transliterated text lacks script-based cues, the model learns language-specific patterns directly from textual data. By leveraging fastText's subword embeddings, we can improve identification accuracy. Sentences are classified using a trained fastText model, and predictions are retained only if the confidence exceeds a predefined threshold. Otherwise, the sentence is labeled as unidentified.

4.2.2.1 Multilingual Sentence-Level Language Identification Pipeline

This pipeline performs sentence-level language identification for native script and Romanized text. The script with the highest characters is identified as the dominant script if characters in this script exceed a threshold of 90% of the sentence length. Among Indian languages for this model, 9 have unique scripts, allowing us to use a Unicode-based preprocessing method to determine sentences in these languages. If dominant script corresponds to a language with a unique script-language pair, we directly assign the language and script label. Otherwise, we apply a fastText [BGJM17] model, selecting from four separate models, for native scripts Devanagri, Arabic and Bengali, and Romanized model for Latin script trained on data of script groups. For sentences without a dominant script, we use a general model trained on all native scripts and Romanized data, providing a result only if confidence is high; otherwise, the sentence is marked as unidentifiable.

Script (Code)	**Language (Code)**
Gujarati (Gujr)	Gujarati (guj)
Tamil (Taml)	Tamil (tam)
Odia (Orya)	Odia (ory)
Ol Chiki (Olck)	Santali (sat)
Kannada (Knda)	Kannada (kan)
Malayalam (Mlym)	Malayalam (mal)
Meetei Mayek (Mtei)	Manipuri (mni)
Gurmukhi (Guru)	Punjabi (pan)
Telugu (Telu)	Telugu (tel)

Table 4.8: Mapping of scripts to their corresponding unique languages

Algorithm 1 Sentence-Level Language Detection

```
1: function languageDetectionSentenceLevel(models, sentence)
2:     script ← getMajorScript(sentence)
3:     if script.percentage > thresholdScript then
4:         if script.value in [Arabic, Bengali, Devanagri, Latin] then
5:             return predictLanguage(models[script.value], sentence)
6:         else
7:             return lookupTable[script.value]
8:         end if
9:     else
10:        prediction, confidence = generalModel(sentence)
11:        if confidence > thresholdConfidence then
12:            return prediction
13:        else
14:            return undetectable
15:        end if
16:    end if
17: end function
```

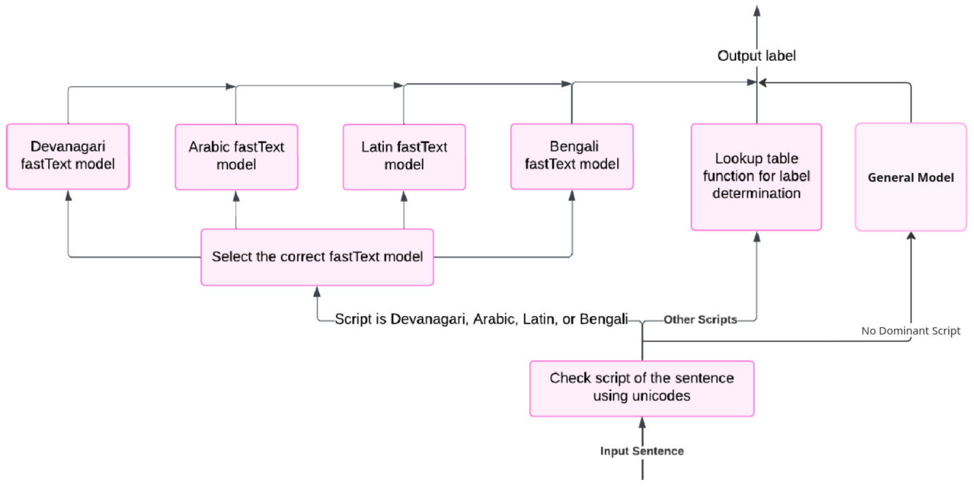

Output label
Devanagari fastText model
Arabic fastText model
Latin fastText model
Bengali fastText model
Lookup table function for label determination
General Model
Select the correct fastText model
Script is Devanagari, Arabic, Latin, or Bengali
Other Scripts
No Dominant Script
Check script of the sentence using unicodes
Input Sentence

Figure 4.2: fastText + Unicode-Range based sentence based language identification for native and Romanized languages in a single script dataset.

4.2.3 Code-Mixed

4.2.3.1 Word-Level Language Tagging Models

We first use the sentence-level language identification model to detect the dominant language of the sentence. Based on this prediction, we route the sentence to the corresponding word-level tagging model. For instance, if the detected language is Hindi, we apply the Hindi-English word-level model; if Telugu is detected, the Telugu-English model is used. This modular approach ensures that the appropriate model is used for code-mixed sentences, improving tagging accuracy by leveraging language-specific patterns.

For word-level language identification, we developed two separate models: one for Hindi-English and another for Telugu-English code-mixed data. Each model is trained using a BiLSTM-based sequence tagging architecture, tailored to the tag sets of the respective datasets. The Hindi-English dataset uses tags `HI` and `EN`, while the Telugu-English dataset uses a richer tag set: `te`, `en`, `univ`, and `ne`. Since the tag sets differ, it was necessary to train and evaluate two independent models.

4.2.3.2 Model Architecture

- **Embedding Layer:** Converts input word indices into dense vector representations.
- **BiLSTM Layer:** A bidirectional LSTM captures contextual information from both past and future word sequences.
- **Linear Layer:** Maps BiLSTM outputs to tag scores for each word.

4.2.3.3 Model Workflow

Each word and its corresponding tag in the dataset is first converted into a numerical representation using separate vocabularies for words and labels. Sentences and their label sequences are then padded to ensure uniform length across batches.

We use randomly initialized, trainable word embeddings via PyTorch's `nn.Embedding` layer. These embeddings are learned from scratch during training and are optimized for the code-mixed language identification task, enabling the model to capture task-specific word representations.

The embedded sequences are then passed through a Bidirectional LSTM (BiLSTM), which learns contextual dependencies in both forward and backward directions. This bidirectional processing helps the model utilize both left and right context, which is especially valuable in code-mixed text where the language of a word can depend heavily on its surrounding words.

The output of the BiLSTM is passed through a linear layer that maps the hidden representations to tag scores for each word. The final prediction is made by selecting the tag with the highest score at each position. We use a cross-entropy loss function during training, ignoring padding tokens, and update the model parameters using backpropagation.

4.2.3.4 Why BiLSTM?

BiLSTM models are particularly effective for sequence tagging tasks because they can leverage both left and right context in a sentence. This is crucial in language tagging, where the meaning—and hence the language—of a word can depend heavily on surrounding words, especially in code-mixed sentences.

4.3 Conclusion

In this chapter, we developed a native script dataset covering 30 language-script pairs and trained script-specific fastText models. For Romanized text, we constructed a Latin-script dataset covering 22 Indian languages collected from various sources and trained a single unified fastText-based sentence-level language identification model. These components together form a multilingual identification pipeline across both native and Romanized scripts.

Additionally, we built BiLSTM-based word-level tagging models for Hindi-English and Telugu-English code-mixed data. These models effectively leverage contextual information to perform accurate intra-sentential language identification.

Chapter 5

Experimental Setup

5.1 Setup

The proposed approach is tested on a workstation featuring an Intel Xeon W-1370P processor with 16 cores and two threads per core, coupled with 125GB of RAM and equipped with an NVIDIA T400 GPU with 1867 MiB of VRAM, utilizing NVIDIA driver version 470.239.06. The operating system used was Ubuntu 20.04.6 LTS.

5.2 Dataset

The Native Script dataset includes only single-script sentences, with each sentence primarily belonging to a single language. While some sentences may contain borrowed words from other languages, such cases are minimal and do not impact the script consistency.

In contrast, the Code-Mixed dataset consists of sentences where an Indian language is mixed with English. Each word in these sentences is tagged with its corresponding language, and additional tags such as `univ` (universal terms) and `proper noun` are included to handle named entities and language-agnostic tokens.

In models data is split into training (72%), validation (8%) and testing (20%) datasets.

For testing the Combined Native Script and Romanized text Pipeline, separate evaluation data—distinct from the training sets of the three individual script-specific models—was collected. This data was used to evaluate the performance of the combined model across scripts and was not included in any training. Including this data in training would defeat the purpose of testing, as the combined model relies on the generalization ability learned from the individually trained script-specific models.

5.3 Model parameters and Hyperparameters

5.3.1 FastText based

In both fastText and a combination of fastText and Unicode-range based models, supervised training of fastText has been used with default parameters, where these settings were identified as optimal for language identification tasks in Indian languages, reflecting its suitability for lightweight and efficient language identification. The default fastText parameters yielded better performance across all evaluation metrics—precision, recall, F1-score, and accuracy—compared to the Bhasha Abhijanana configuration, which only achieved a higher throughput.

Metric	**Default Parameters**	**Bhasha Abhijanana Parameters**
Precision	0.9987	0.9919
Recall	0.9988	0.9918
F1-score	0.9993	0.9918
Accuracy	0.9988	0.9918
Throughput	33572.07	101777.00

Table 5.1: Performance comparison (Ablation Studies) of Native Script model using default fastText parameters vs. Bhasha Abhijanana configuration (`loss=hs`, `verbose=1`, `dim=8`)

Parameter	Default	Description
General		
`-verbose`	2	Logging level
Dictionary Parameters		
`-minCount`	1	Min word frequency
`-minCountLabel`	0	Min label frequency
`-wordNgrams`	1	Max word n-gram length
`-bucket`	2000000	Hash bucket size for n-grams
`-minn`	0	Min char n-gram length
`-maxn`	0	Max char n-gram length
`-t`	0.0001	Subsampling threshold
`-label`	___label___	Label prefix
Training Parameters		
`-lr`	0.1	Learning rate
`-lrUpdateRate`	100	LR update interval
`-dim`	100	Word vector size
`-ws`	5	Context window size
`-epoch`	5	No. of epochs
`-neg`	5	No. of negative samples
`-loss`	softmax	Loss function type
`-thread`	12	No. of threads
`-pretrainedVectors`	(empty)	Pretrained vectors path
`-saveOutput`	0	Save output vectors
Quantization Parameters		
`-cutoff`	0	Retain top N features
`-retrain`	0	Finetune after cutoff
`-qnorm`	0	Quantize norms separately
`-qout`	0	Quantize classifier
`-dsub`	2	Sub-vector size

Table 5.2: FastText parameters with default values used

5.3.2 BiLSTM Model Parameters

The architecture used for word-level language tagging is a BiLSTM-based sequence tagging model. Below is a summary of the key model components and their configurations for the two variants trained on different datasets.

Component	**Telugu-English (4 Tags)**	**Hindi-English (2 Tags)**
Embedding Layer	61296 words, 64-dim	75912 words, 64-dim
BiLSTM Layer	Hidden size: 64 (bidirectional)	Hidden size: 64 (bidirectional)
Linear Layer	Input: 128, Output: 6	Input: 128, Output: 4

Table 5.3: BiLSTMTagger architecture summary for Telugu-English and Hindi-English models

5.4 Evaluation Metrics

We evaluate our system across multiple metrics, each capturing different aspects of performance. The evaluation metrics are as follows:

- **Precision**: Precision is the ratio of true positive predictions to the sum of true positive and false positive predictions. It measures the accuracy of positive predictions and is formulated as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall**: Recall is the ratio of true positive predictions to the sum of true positives and false negatives. It measures the system's ability to identify all relevant instances and is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1 Score**: The F1 score is the harmonic mean of precision and recall, providing a balanced metric that considers both false positives and false negatives. It is calculated as:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Accuracy**: Accuracy is the ratio of correct predictions (both true positives and true negatives) to the total number of predictions. It provides an overall measure of how often the model makes correct predictions and is given by:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}$$

- **Throughput**: Throughput refers to the number of records processed by the system per second. This metric evaluates the efficiency and speed of the system, particularly in real-time or high-volume processing environments. It is calculated as:

$$\text{Throughput} = \frac{\text{Total Records Processed}}{\text{Total Time (seconds)}}$$

Each metric is plotted against varying dimensions to observe its behavior and impact on system performance. These plots help identify optimal configurations for achieving high accuracy, balanced precision and recall, and maximum throughput.

Chapter 6

Results and Analysis

6.1 Model Performance

Each model was assessed based on its precision, recall. F1 score, accuracy and throughput to determine its suitability for real-world applications.

6.1.1 Native Script

We present the confusion matrix using normalized values. This allows for easier comparison across languages, especially when the number of test samples per class varies.

In the normalization matrices for the Bengali script model—initially, Assamese and Bengali were often confused with each other due to Assamese having the least data. After data addition, Manipuri became the least represented among the three, resulting in increased confusion between Bengali and Manipuri. This suggests that underrepresented languages are more likely to be misclassified into dominant ones during model training.

6.1.2 Transliterated Latin Script

Due to the large number of language classes (22 in total) in the Latin script setting in addition to 'Unknown' tag for unidentifiable language, we present the confusion matrix using percentages to make interpretation more accessible. This also allows for easier comparison across languages, given the number of test samples per class varies. A notable observation in the matrix is the frequent misclassification of **Konkani** as **Marathi**. However, the reverse—Marathi being misclassified as Konkani—is negligible. One likely reason is the imbalance in training data: the Marathi dataset consists of 12,340 samples, while Konkani has only 1,441 in Latin script. This data skew causes the model to favor Marathi, especially since both languages share lexical and phonetic similarities when transliterated into Latin script and the geographical and linguistic proximity increases lexical borrowing between the two languages.

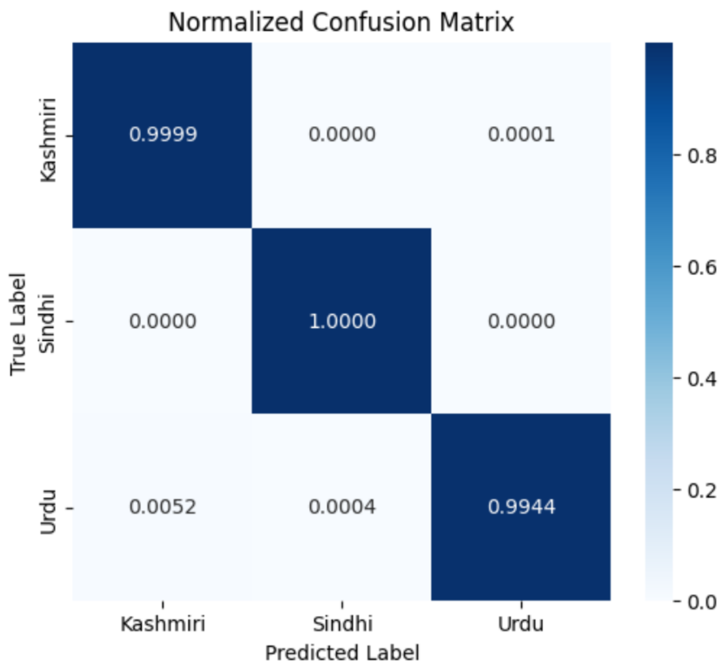


Figure 6.1: Normalization matrix for the Arabic script

Model	Precision	Recall	F1-score	Accuracy	Throughput
Devanagari (Native)	0.9865	0.9865	0.9865	0.9865	38258.96
Bengali (Native)	0.9917	0.9917	0.9917	0.9917	45816.03
Arabic (Native)	0.9998	0.9998	0.9998	0.9998	36314.46
Native Model (Combined)	0.9987	0.9988	0.9993	0.9988	33572.07
Latin Model	0.9877	0.9873	0.9874	0.9874	31798.92
Code-Mixed (Romanized)	0.9423	0.9441	0.9430	0.9431	28574.60

Table 6.1: Performance metrics for different models.

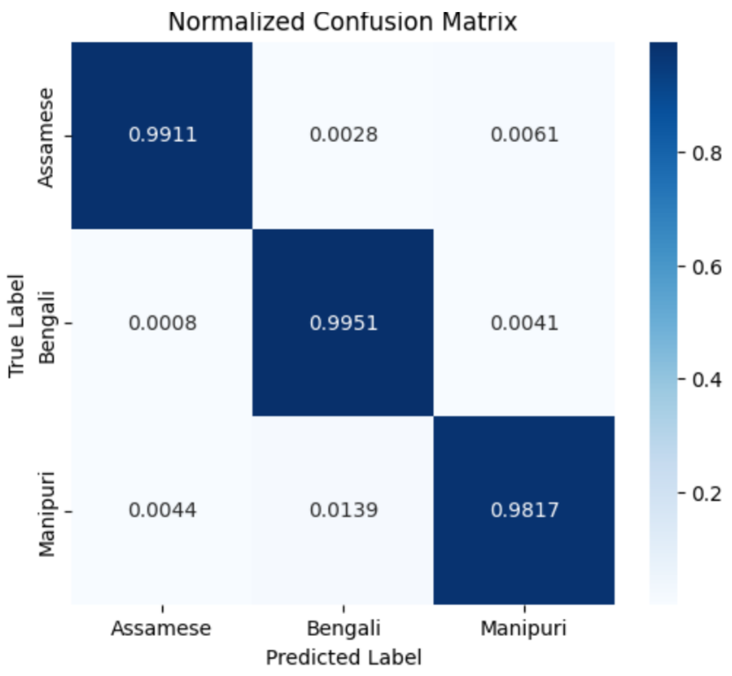


Figure 6.2: Normalization matrix for Bengali script model

Model	Precision	Recall	F1-score	Accuracy	Throughput
Unicode + Script-based	0.9987	0.9988	0.9993	0.9988	33572.07
Native (Single Model)	0.9917	0.9917	0.9917	0.9917	45816.03

Table 6.2: Performance metrics for different Native Script models.

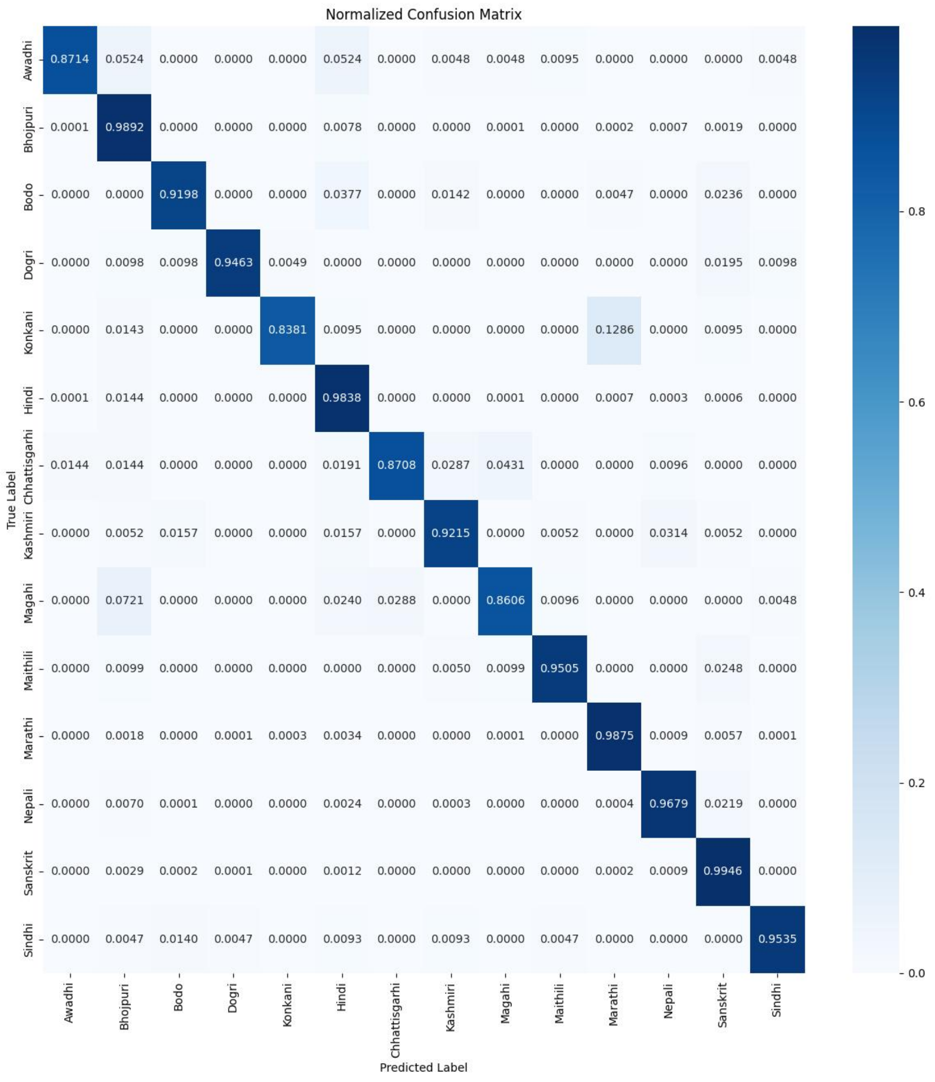
6.1.3 Code-Mixed

6.1.3.1 Hindi-English Code-Mixed

This model is trained on data with 2 tags: `HI` (Hindi) and `EN` (English), representing word-level language labels in Romanized Hindi-English sentences.

Label	Precision	Recall	F1-score	Support
`HI`	0.9856857	0.9859822	0.9858339	192826
`EN`	0.9649034	0.9641773	0.9645402	77074
Accuracy	0.9797555			
Macro Avg	0.9752945	0.9750797	0.9751871	269900
Weighted Avg	0.9797510	0.9797555	0.9797532	269900

Table 6.3: Performance metrics for Hindi-English word-level language tagging using BiLSTM



True \ Predicted	Awadhi	Bhojpuri	Bodo	Dogri	Konkani	Hindi	Chhattisgarhi	Kashmiri	Magahi	Maithili	Marathi	Nepali	Sanskrit	Sindhi
Awadhi	0.8714	0.0524	0.0000	0.0000	0.0000	0.0524	0.0000	0.0048	0.0048	0.0095	0.0000	0.0000	0.0000	0.0048
Bhojpuri	0.0001	0.9892	0.0000	0.0000	0.0000	0.0078	0.0000	0.0000	0.0001	0.0000	0.0002	0.0007	0.0019	0.0000
Bodo	0.0000	0.0000	0.9198	0.0000	0.0000	0.0377	0.0000	0.0142	0.0000	0.0000	0.0047	0.0000	0.0236	0.0000
Dogri	0.0000	0.0098	0.0098	0.9463	0.0049	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0195	0.0098
Konkani	0.0000	0.0143	0.0000	0.0000	0.8381	0.0095	0.0000	0.0000	0.0000	0.0000	0.1286	0.0000	0.0095	0.0000
Hindi	0.0001	0.0144	0.0000	0.0000	0.0000	0.9838	0.0000	0.0000	0.0001	0.0000	0.0007	0.0003	0.0006	0.0000
Chhattisgarhi	0.0144	0.0144	0.0000	0.0000	0.0000	0.0191	0.8708	0.0287	0.0431	0.0000	0.0000	0.0096	0.0000	0.0000
Kashmiri	0.0000	0.0052	0.0157	0.0000	0.0000	0.0157	0.0000	0.9215	0.0000	0.0052	0.0000	0.0314	0.0052	0.0000
Magahi	0.0000	0.0721	0.0000	0.0000	0.0000	0.0240	0.0288	0.0000	0.8606	0.0096	0.0000	0.0000	0.0000	0.0048
Maithili	0.0000	0.0099	0.0000	0.0000	0.0000	0.0000	0.0000	0.0050	0.0099	0.9505	0.0000	0.0000	0.0248	0.0000
Marathi	0.0000	0.0018	0.0000	0.0001	0.0003	0.0034	0.0000	0.0000	0.0001	0.0000	0.9875	0.0009	0.0057	0.0001
Nepali	0.0000	0.0070	0.0001	0.0000	0.0000	0.0024	0.0000	0.0003	0.0000	0.0000	0.0004	0.9679	0.0219	0.0000
Sanskrit	0.0000	0.0029	0.0002	0.0001	0.0000	0.0012	0.0000	0.0000	0.0000	0.0000	0.0002	0.0009	0.9946	0.0000
Sindhi	0.0000	0.0047	0.0140	0.0047	0.0000	0.0093	0.0000	0.0093	0.0000	0.0047	0.0000	0.0000	0.0000	0.9535

Figure 6.3: Normalization matrix for Devanagari script model

6.1.3.2 Telugu-English Code-Mixed

This model is trained on data with 4 tags: `te` (Telugu), `en` (English), `univ` (universal tokens like punctuation and emojis), and `ne` (named entities). These tags represent word-level labels in Romanized Telugu-English code-mixed sentences.

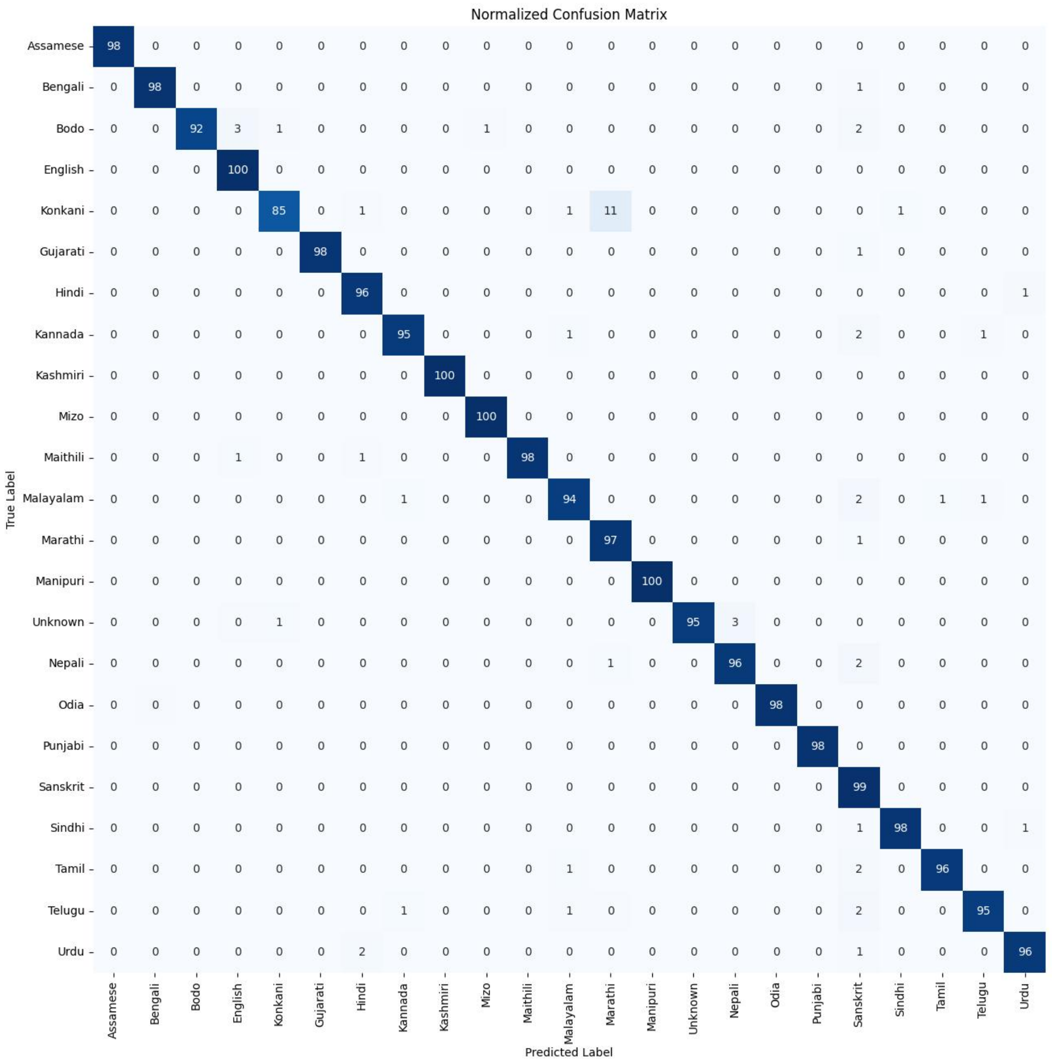


Figure 6.4: Normalization matrix for Transliterated Latin script model

6.1.4 Analysis

As shown in Figure 6.2, the pipeline using four separate models for Arabic, Bengali, Devanagari, and Romanized scripts outperforms the FastText model trained on the combined dataset. This highlights the effectiveness of script-specific modeling for sentence-level language identification.

Label	**Precision**	**Recall**	**F1-score**	**Support**
`en`	0.9025767	0.9248479	0.9135766	26626
`univ`	0.9128501	0.9056508	0.9092362	13821
`te`	0.9196362	0.9212431	0.9204390	32048
`ne`	0.7447346	0.5936870	0.6606876	2978
Accuracy	0.9067349			
Macro Avg	0.8699494	0.8363572	0.8509848	75473
Weighted Avg	0.9054738	0.9067349	0.9057173	75473

Table 6.4: Performance metrics for Telugu-English word-level language tagging using BiLSTM

Chapter 7

Conclusion and Future Work

In Phase 2 of this BTP, we explored a sentence-level language identification pipeline using script-specific models trained on native script data and a unified model for Romanized text. These sentence-level models laid the foundation for the development of word-level language tagging systems for Romanized code-mixed data.

As part of our work, we developed sentence-level labeled datasets in both native and Latin scripts, annotated with language and script labels. Additionally, we used Hindi-English and Telugu-English code-mixed datasets to train separate BiLSTM-based sequence tagging models for each language pair, accounting for their specific tagging schemes. When combined with the sentence-level pipeline, this setup provides a general-purpose framework for word-level language tagging—currently supporting two language pairs—with potential for extension to others. Looking ahead, we plan to significantly expand the word-level tagged dataset to include more Indian languages in Romanized script, moving toward a unified system capable of handling intra-sentential code-mixed inputs across multiple languages. Future work will focus on improving tagging accuracy, extending coverage, and refining model architectures to support efficient and accurate processing of Romanized Indic languages in multilingual NLP applications.

Bibliography

[AI16] Google AI. Compact language detector 3 (cld3). 2016. https://github.com/google/cld3.

[AI422] AI4Bharat. Aksharantar: Open-source indic transliteration datasets. https://github.com/AI4Bharat/aksharantar, 2022. Accessed: 2025-04-16.

[BC20] Lavanya Bandi and Rahul Choudhary. Sentiment analysis on code-mix telugu-english text. In *Proceedings of the Second Workshop on Computational Modeling of People's Opinions, Personality, and Emotion's in Social Media*, pages 1–6, 2020.

[BGJM17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[BJPD22] Aditya Bhardwaj, Hardik Joshi, Harsh Palod, and Amitabha Deshmukh. L3cube-hingcorpus and hingbert: A code mixed hindi-english dataset and bert language models. *arXiv preprint arXiv:2201.09578*, 2022.

[C$^+$20] Isaac A. Caswell et al. Language identification for low resource languages: A case study of tigrinya. *arXiv preprint arXiv:2009.10717*, 2020.

[CKG$^+$20] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzman, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[CL19] Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, 2019.

[CWT13] Simon Carter, Wouter Weerkamp, and Manos Tsagkias. Microblog language identification: Overcoming the limitations of short, unedited, and idiomatic text. *Language Resources and Evaluation*, 2013.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019.

[GS05] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 2047–2052. IEEE, 2005.

[JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[JTJ17] David Jurgens, Yulia Tsvetkov, and Dan Jurafsky. Incorporating dialectal variability for socially equitable language identification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017.

[K$^+$21] Simran Khanuja et al. Muril: Multilingual representations for indian languages. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[K$^+$22] Simran Khanuja et al. Supervised models for multilingual token classification on low-resource indian languages. *arXiv preprint arXiv:2205.03983*, 2022.

[KCW$^+$22] J. Kreutzer, I. Caswell, Y. Wang, A. Wahab, and S. Wu. Quality at a glance: An audit of web-crawled multilingual datasets. *arXiv preprint arXiv:2201.08239*, 2022.

[KKG$^+$20] D. Kakwani, A. Kunchukuttan, D. Golla, A. Bhattacharjee, M. Khapra, and P. Kumar. Indicnlpsuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for indian languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020.

[LB11] Marco Lui and Timothy Baldwin. langid.py: An off-the-shelf language identification tool. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[MKK23] Yash Madhani, Mitesh M. Khapra, and Anoop Kunchukuttan. Bhasha-abhijnaanam: Native-script and romanized language identification for 22 indic languages. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 816–826. Association for Computational Linguistics, 2023.

[N$^+$22] NLLB Team et al. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*, 2022.

[RHS$^+$20] Brian Roark, Keith Hall, Supheakmungkol Sarin, Nandakishore Arivazhagan, and Richard Sproat. The dakshina dataset: A benchmark for cross-script natural language processing in indian languages. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 3783–3792. European Language Resources Association, 2020.

[RSC$^+$17] Shyamanthe Rijhwani, Russell Sequiera, Monojit Choudhury, Kalika Bali, and C. Sandeep Maddila. Estimating code-switching on twitter with a novel generalized word-level language detection technique. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[SBM$^+$14] Thamar Solorio, Emily Blair, Suraj Maharjan, Steven Bethard, Mona Diab, Mahmoud Ghoneim, Abdelati Hawwari, Fahad AlGhamdi, Julia Hirschberg, Angel Chang, et al. Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, 2014.

[SS21] Vivek Srivastava and Mayank Singh. Challenges and considerations with code-mixed nlp for multilingual societies. 2021.

[XCR$^+$21] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2021.