CONFLATOR: Incorporating Switching Point based Rotatory Positional Encodings for Code-Mixed Language Modeling

Mohsin Ali[1] Sai Teja Kandukuri[2] Neeharika Gupta[3] Parth Patwa[4]
Anubhab Chatterjee[3] Vinija Jain[5] Aman Chadha[5,6*] Amitava Das[7]
[1]San Diego State University, USA [2]San Jose State University, USA
[3]Wipro AI Labs, India [4]University of California Los Angeles, USA
[5]Stanford University, USA [6]Amazon AI, USA [7]University of South Carolina, USA
[1]mmohammed5956@sdsu.edu [2]saiteja.kandukuri@sjsu.edu
[7]amitava@mailbox.sc.edu

Abstract

The mixing of two or more languages is called Code-Mixing (CM). CM is a social norm in multilingual societies. Neural Language Models (NLMs) like transformers have been effective on many NLP tasks. However, NLM for CM is an under-explored area. Though transformers are capable and powerful, they cannot always encode positional information since they are non-recurrent. Therefore, to enrich word information and incorporate positional information, positional encoding is defined. We hypothesize that Switching Points (SPs), i.e., junctions in the text where the language switches (L1 $\rightarrow$ L2 or L2 $\rightarrow$ L1), pose a challenge for CM Language Models (LMs), and hence give special emphasis to SPs in the modeling process. We experiment with several positional encoding mechanisms and show that rotatory positional encodings along with switching point information yield the best results.

We introduce CONFLATOR: a neural language modeling approach for code-mixed languages. CONFLATOR tries to learn to emphasize switching points using smarter positional encoding, both at unigram and bigram levels. CONFLATOR outperforms the state-of-the-art on two tasks based on code-mixed Hindi and English (Hinglish): (i) sentiment analysis and (ii) machine translation.

1 Code-Mixing: Juxtaposition of two Languages

Code-mixing is defined as the alternation of two or more languages during articulation. Recently, code-mixing has gained a lot of attention in the area of NLP due to the prevalence of language mixing in multilingual societies such as India, Europe, US, South Africa, Mexico, etc. In such societies, code-mixing is fairly commonplace, especially in informal conversations, where the native language is often romanized and code-mixed with an auxiliary language. This effect occasionally manifests in posts originating from the aforementioned sources on social media platforms such as Twitter, Facebook, etc. An example of Hindi and English code-mixing is shown in the following phrase where an English word, *dance*, is mixed with Hindi romanized words: *Gaaye, aur, kare*.

Gaaye$_{\texttt{HI}}$ *aur*$_{\texttt{HI}}$ *dance*$_{\texttt{EN}}$ *kare*$_{\texttt{HI}}$
English translation: sing and dance

With the proliferation of code-mixing on the internet, it is important to study language processing and language modeling for code-mixed languages. While language modeling using neural networks has come a long way, replacing n-gram language models with distributed neural representations (Bengio et al., 2003) to recent large transformer-based pre-trained language models (LMs) such as GPT-x (Radford et al., 2019), BERT (Devlin et al., 2018a) etc., code-mixed language modeling using state-of-the-art (SoTA) Transformer-based models is still under-explored.

The biggest hindrance in the adoption of SoTA Transformer-based LMs for code-mixing can be attributed to data scarcity. While Transformer-based (Vaswani et al., 2017b) architectures such as BERT and GPT have set new benchmarks in the domain of language modeling, they are infamous for their low sample efficiency. In other words, the voracious data appetite of Transformers and the lack of substantial code-mixed datasets in the community is the primary reason for the technological hindrances in the area of code-mixed language modeling compared to vanilla language modeling.

To corroborate the aforementioned arguments, we experiment with Transformer-based models such as GPT-2 and BERT for code-mixing. We empirically observe that these models perform poorly on tasks involving code-mixed data. Our hypothesis is as follows: Since information related to switch-

[*]Work does not relate to position at Amazon.

ing point is a major component in the context of code-mixed content, it should thus be incorporated in downstream processing. Switching points are a bottleneck for a model's processing of code-mixed data and the reason for poor performance using SoTA neural language models (Chatterjere et al., 2020). Switching points play a crucial factor when dealing with CM data. In the next few sections, we discuss various positional encoding approaches, switching points, and our approaches for language modeling on code-mixed data. Our key contributions are:

- We propose *CONFLATOR*, an LM system that incorporates switching point related positional information.
- Our system improves the performance of existing models and achieves a new SoTA on two tasks.
- We investigate, experiment with, and introduce various switching point based positional encoding techniques.
- We introduce a novel Switching Point based Rotary matrix for Rotary Positional Encoding (RoPE).
- We curate a new dataset of code-mixed tweets.

2 Related Work

It is important to study code-mixing as it is a part of most multilingual societies and prevalent in social media. It is more complex to process code-mixed text than monolingual text for NLP tasks (Verma, 1976). Similar line of work was followed by Bokamba (1988) and Singh (1985) on the complexities of multi-languages on the basis of syntactics and grammar. The difficulties of processing code-mixed languages on social media is further exacerbated by unusual spellings, many unique ways of writing the same word, unnecessary capitalization etc (Das and Gambäck, 2014; Laddha et al., 2020).

With the growing popularity on social media, Various tasks like sentiment analysis (Patwa et al., 2020a; Chakravarthi et al., 2020), translation (Dhar et al., 2018; Srivastava and Singh, 2020), hate-speech detection (Bohra et al., 2018; Banerjee et al., 2020), POS tagging (Vyas et al., 2014), etc. have been performed on code-mixed data. Methods to handle code-mixing for text classification include the use of CNNs (Aroyehun and Gelbukh, 2018; Patwa et al., 2020b), Transformer or BERT like models (Samghabadi et al., 2020; Tang et al., 2020), ensemble models (Tula et al., 2021; Jhanwar and Das, 2018), focal loss (Tula et al., 2022; Ma et al., 2020) etc.

Vaswani et al. (2017a) proposed transformers for neural language modeling using masked language modeling (MLM) and next sentence prediction, which achieved SoTA performance on many NLP tasks. Devlin et al. (2018b) released mBERT, a model trained on multilingual corpus that includes 104 languages. A cross lingual language model XLM was proposed in Lample and Conneau (2019) which leveraged monolingual and crosslingual corpus for pretraining. Nayak and Joshi (2022) present a bert pretrained on CM data. However, they do not make changes to their language model or technique to handle code-mixed data in particular. Sengupta et al. (2021) propose a Hierarchical transformer based architecture that captures the semantic relationship among words and hierarchically learns the sentence level semantics of code-mixed data. Ali et al. (2022) Were one of the first to incorporate switching point information in positional encoding. They utilize dynamic positional encodings whereas our method, CONFLATOR infuses switching point information in rotatory positional encodings and also uses both unigram and bigram tokens to get the final embedding.

3 Data Extraction and Strategies

In this section, we discuss the details of code-mixed data extraction. Our primary aim is to extract naturally distributed code-mixed data.

3.1 Qualitative and Quantitative Checkpoints for Hinglish Corpus

The performance of LMs is dependent on the training data size and quality, along with the vocabulary size. Code-mixed language modeling suffers from the following challenges: i) data scarcity, ii) Words from 2 (or more) languages in the same sentence, iii) *Hindi* is written using *English* letters (i.e. transliteration), hence, there is no standardization of spelling - which in effect proliferates word forms (Laddha et al., 2020, 2022), iii) Code-mixing is usually found on social media and netizens often incorporate creativity in their mixing along with wordplay. We consider two fundamental questions to guide our data collection:

1. *The performance on any NLP task depends on the data complexity:*

Empirical measurement: Consider two 4-word tweets - i) $T_i : w_{L1}w_{L1}w_{L2}w_{L2}$ and ii) $T_j : w_{L1}w_{L2}w_{L1}w_{L2}$. Both the tweets have 2 words each from the languages $L1$ and $L2$. Thus the mixing ratio of both the tweets T_i and T_j is $(4-2)/4 = 0.50$. However, T_i only contains 1 code alternation point whereas T_j contains 3 switches. It is likely that T_j is harder to process. Hence, we need a metric for the level of mixing between the languages. We use *Code-Mixing-Index* (Gambäck and Das, 2016) (CMI) to measure such complexity. Please refer to section 3.2 for more details on CMI.

2. *How much data is good enough?*

   Empirical measurement: When two languages blend, it is quite natural that the number of unique word forms would be much higher in a Hinglish corpus in comparison to monolingual English or Hindi corpus. Therefore, we ask an essential question at the very beginning, *how much data is good enough?* We decide to keep collecting data, until the Heaps' curve starts converging so that we cover most of the unique words.

   Heaps' law (Gopalan and Hopkins, 2020) states that the number of unique words in a text of n words is approximated by $V(n) = Kn^{\beta}$ where K is a positive constant and β lies between 0 and 1, K invariably lies between 10 and 100 and β between 0.4 an 0.6. Heaps' law is often considered to be a good estimator to calculate the vocabulary size. To compare, from the figure 1, it can be seen that, for English Wiki, the flattening of the Heaps' law curve, starts at **40K-50K**, whereas for monolingual Hindi, it converges at **80K-90K**, but for *Hinglish* the same behavior starts around **800K** vocabulary and 50M words.

3.2 Code-Mixing Index (CMI)

As mentioned previously, we expect the difficulty of language processing tasks to increase as the level of code-mixing increases. To measure the level of code-mixing in our corpus, we use Code-mixing Index (Gambäck and Das, 2016) :

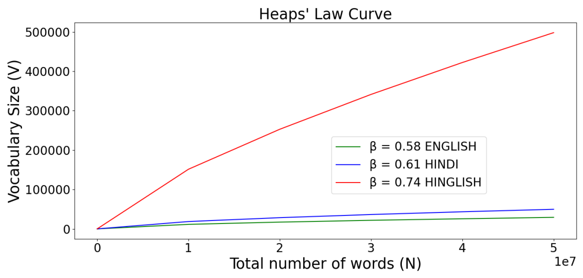

Figure 1: Heaps' plot on 50M word forms in English, Hindi and Hinglish corpora. The β values are 0.58, 0.61, 0.74 respectively.

$$\begin{aligned} C_u(x) &= w_m f_m(x) + w_p f_p(x) \\ &= w_m \frac{N(x) - \max_{L_i \epsilon L}(tLi)(x)}{N(x)} * 100 + w_p \frac{P(x)}{N(x)} * 100 \\ &= 100 * \frac{w_m((N(x) - \max_{L_i \epsilon L}(tLi)(x)) + w_p P(x)}{N(x)} \end{aligned} \quad (1)$$

Where x denotes utterance, N is the number of token in x belonging to any language L_i, w_m and w_n are weights. Please refer to Gambäck and Das (2016) for a detailed explanation of CMI.

3.3 Data Acquisition Pipeline

We follow a pipeline similar to (Chatterjere et al., 2020). We collect CM data from Twitter via the Twitter API. We need to use relevant keywords (words unique to Hindi) in our search to get CM tweets. Words with lexical overlap between Hindi and English should not be used for searching. for example, the word *do* is confusing because it means two in Hindi. We start with the *ICON 2017* Hinglish sentiment analysis dataset (Patra et al., 2018), which is annotated with word-level language. From this data, we create two vocabularies V_{HI} and V_{EN}, and generate a vocabulary of unique Hindi words $V_{HI-UNIQ} = V_{HI} - I$, where $I = V_{HI} \bigcap V_{EN}$. $V_{HI-UNIQ}$ set is then sorted in descending order, based on the word frequency, and is used as search words on the Twitter API. Once we get the tweets, we use a word-level language identifier (Barman et al., 2014) (having 90%+ accuracy) on the tweets and calculate the CMI of the tweet. Once we get the word-level language labels, we can also know where the switching points are. Tweets with CMI = 0 are discarded. Finally, we are left with 87k tweets. The CMI distribution of our data is given in table 1. This dataset is used to pretrain our models.

Training and Testing data: We collect 87K sentences distributed over all CMI ranges, instead of

CMI	# Tweets	Percentage
0-10	7,036	8.05%
11-20	16,481	18.9%
21-30	22,617	25.9%
31-40	22,722	26.0%
41-50	11,404	13.1%
50+	7,036	8.05%
Mean CMI: **28**	Total # of tweets: **87,296**	

Table 1: CMI distribution of the collected data. The total number of extracted tweets is 87K.

collecting equal data across the CMI ranges, so that the resultant languages trained on this corpus would be able to handle real data. We maintain the same distribution over both our training and testing corpora (*4:1 ratio*), for our language models.

4 The Bottleneck of Code-mixed Language Modeling: Switching Points

Formally, Switching Points (SPs) are the tokens in text, where the language switches. For code-mixed languages, consisting of a pair of languages, there can be two types of switching points. Suppose the two languages as part of the code-mixed language are *L1* and *L2*, a switching point occurs when the language in the text changes from L1 to L2 or L2 to L1. To explain it better, let us consider the following sample in *Hinglish*:

gaana$_{\text{HI}}$ *enjoy*$_{\text{EN}}$ *kare*$_{\text{HI}}$
English Translation: Enjoy the song.

In the above example, when the language switches from *Hindi* to *English* (*gaana*$_{\text{HI}}$ *enjoy*$_{\text{EN}}$) a **HI-EN** (HIndi-ENglish) switching point occurs. Similarly, a **EN-HI**(ENglish-HIndi) switching point occurs at - *enjoy*$_{\text{EN}}$ *kare*$_{\text{HI}}$.

In the context of modeling code-mixed languages, switching points can be considered as ordinary bigrams, that occur with other monolingual bigrams in a corpus. It is easy to infer that particular SP bigrams will be relatively rare in a given corpus. Hence, such sparse occurrences of switching point bigrams make it difficult for any Language Model to learn their probabilities and context. Since the language changes at the switching point, LMs are likely to find it difficult to process these tokens. In order to counter this challenge, we partition our code-mixed data into **(i)** *switching points*, and **(ii)** *non-switching points*. We then build LMs specifically for switching points and non-switching points, as discussed in the following sections.

CONFLATOR Hypothesis: The CONFLATOR is built on 2 hypotheses. i) Positional information is important for language models, especially when dealing with CM text. ii) Switching points are the bottleneck for code-mixed language models (CMLM). We incorporate positional information of switching points into our CMLM.

5 Positional Encoding Techniques

As discussed, SPs are a major bottleneck hence handling them separately is needed. Positional encoding are necessary for language models to learn dependencies between tokens. Positional embedding was first introduced by Vaswani et al. (2017b). The proposed sinusoidal positional encoding is composed of sine and cosine values with position index as inputs. The encoding techniques are further improved by Liu et al. (2020) where a dynamic function is introduced to learn position with gradient flow and Shaw et al. (2018) learned positional representation of relative positions using a learnable parameter. We talk about different positional encoding techniques in detail in the following subsections.

We experiment with several contemporary techniques and find that rotary positional encoding (Su et al., 2021) performs the best.

5.1 Sinusoidal Positional Encoding (SPE)

Vaswani et al. (2017b) introduced a pre-defined sinusoidal vector $p_i \in R^d$ which is assigned to each position *i*. This p_i is added to the word embedding $x_i \in R^d$ at position *i*, and $x_i + p_i$ is used as input to the model such that the Transformer can differentiate words coming from different positions and this also assigns each token a position-dependent attention. - equation 2.

$$e_{ij}^{abs} = \frac{1}{\sqrt{d}} \left((x_i + p_i)\, W^{Q,1} \right) \left((x_j + p_j)\, W^{K,1} \right)^T \quad (2)$$

Where W is the weight matrix, Q is query, K is key, l in the layer.

5.2 Dynamic Positional Encoding (DPE)

Instead of using predefined periodical functions like sin, Liu et al. (2020), introduced a dynamic function $\Theta(i)$ at every encoder layer. Improving upon sinusoidal PE, Dynamic PE learns $\Theta(i)$ instead of a predefined p_i to bring dynamic behavior to the model. At each utterance, this learnable function $\Theta(i)$ tries to learn the best possible representation for positional information with gradient flow.

$\Theta(i)$ is added to the word embedding w_i as given in equation 3.

$$e_{ij} = \frac{1}{\sqrt{d}} \left(\left(x_i + \Theta\left(i\right)\right) W^{Q,1}\right) \left(\left(x_j + \Theta\left(j\right)\right) W^{K,1}\right)^T \quad (3)$$

5.3 Relative Positional Encoding (RPE)

In absolute PE, using different p_i for different positions *i* helps the transformer distinguish words at different positions. However, the absolute PE is not effective in capturing the relative word order. Shaw et al. (2018) introduced a learnable parameter a^l_{i-j} which learns the positional representation of the relative position *i-j* at encoder layer *l*. With the help of this, we can explicitly capture word orders in our model as follows:

$$e^{rel}_{ij} = \frac{1}{\sqrt{d}} \left(\left(x_i\right)^l W^{Q,l}\right) \left(\left(x_i\right)^l W^{K,l} + a^l_{i-j}\right)^T \quad (4)$$

5.4 Switching Point-based Dynamic and Relative Positional Encoding (SPDRPE)

Ali et al. (2022) introduce a novel, switching point based PE. For illustration purposes, consider a code-mixed Hinglish text - *ye_{HI}* —

ye$_{HI}$ *gaana*$_{HI}$ *enjoy*$_{EN}$ *kare*$_{HI}$. SP-based indices **(SPI)** set the index to 0 whenever an SP occurs. Indexing would normally be *Index* = (0, 1, 2, 3), but due to switching point incorporation, this gets changed to *SPI* = (0, 1, 0, 0). In addition to this, they use a learning parameter a^l_{i-j}, which encodes the relative position *i-j* at the encoder layer *l*. This encoding approach learns representations dynamically based on SPs along with the embedding a^l_{i-j} so that it can also capture relative word orders, as follows:

$$e_{ij} = \frac{1}{\sqrt{d}} \left(\left(x_i + \Theta\left(S(l_i)\right)\right)^l W^{Q,l}\right) \left(\left(x_i + \Theta\left(S(l_j)\right)\right)^l W^{K,l} + a^l_{i-j}\right)^T \quad (5)$$

5.5 Rotary Positional Encoding (RoPE)

Analogous to the idea of electromagnetic waves going through a polarizer to preserve their relative amplitude, (Su et al., 2021) came up with the idea of Rotary Positional Encoding (RoPE). The idea is to use rotation matrices on the embedding vectors to generate the positional values. The rotation negates any absolute positional information and only retains information about the relative angles between every pair of word embeddings in a sequence. We know that the dot product between two vectors is a function of the magnitude of individual vectors and the angle between them. Keeping this in mind, the intuition for RoPE is to represent the embeddings as complex numbers and the positions as pure rotations that we apply to them.

Mathematically, the formulations for a simple 2-dimensional case are defined as follows:

$$f_Q(x_i, i) = (W_Q x_i) e^{\sqrt{-1} i\theta}$$
$$f_Q(x_j, j) = (W_K x_j) e^{\sqrt{-1} j\theta}$$
$$g(x_i, x_j, i-j) = Re[(W_Q x_i)(W_K x_i)^* e^{\sqrt{-1}(i-j)\theta}] \quad (6)$$

where $Re[]$ is the real part of a complex number and $(W_{\text{K}} x_{\text{i}})^*$ represents the conjugate complex number of $(W_{\text{K}} x_{\text{i}})$. $\theta \in R$ is a preset non-zero constant. Formulating $f_{\text{(Q,K)}}$ as a matrix multiplication, we get:

$$f_Q(x_i, i) = \begin{pmatrix} cosm\theta_1 & -sinm\theta_1 \\ sinm\theta_1 & cosm\theta_1 \end{pmatrix} \begin{pmatrix} W^{(11)}_{Q,K} & W^{(12)}_{Q,K} \\ W^{(21)}_{Q,K} & W^{(22)}_{Q,K} \end{pmatrix} \begin{pmatrix} x^{(1)}_i \\ x^{(2)}_i \end{pmatrix} \quad (7)$$

where $({x_{\text{i}}}^{(1)}, {x_{\text{i}}}^{(2)})$ is x_{i} expressed in the form of 2D coordinates. In the same way, we can turn function *g* into matrix form. By rotating the transformed embedding vector by an angle in multiples of its position index, we are able to incorporate relative position information. Due to this characteristic, it is termed as Rotary Position Embedding.

In order to generalize the result in 2D to any x_{i} in R_{d} where d is even, they divide the d-dimension space into $\frac{d}{2}$ sub-spaces and combine them in merit of the linearity of inner product, turning the attention formulation:

$$f_{Q,K} = e^{rotary}_{ij} = \frac{1}{\sqrt{d}} \left(RM^d_{\Theta,i} W^{Q,1}\left(x_i\right)\right)^T \left(RM^d_{\Theta,j} W^{K,1}\left(x_j\right)\right) \quad (8)$$

$$RM = \begin{pmatrix} cosm\theta_1 & -sinm\theta_1 & 0 & 0 & ... & 0 & 0 \\ sinm\theta_1 & cosm\theta_1 & 0 & 0 & ... & 0 & 0 \\ 0 & 0 & cosm\theta_2 & -sinm\theta_2 & ... & 0 & 0 \\ 0 & 0 & sinm\theta_2 & cosm\theta_2 & ... & 0 & 0 \\ . & . & . & . & ... & . & . \\ . & . & . & . & ... & . & . \\ 0 & 0 & 0 & 0 & ... & cosm\theta_{d/2} & -sinm\theta_{d/2} \\ 0 & 0 & 0 & 0 & ... & sinm\theta_{d/2} & cosm\theta_{d/2} \end{pmatrix} \quad (9)$$

where RM is orthogonal and sparse matrix predefined parameters

$$\Theta = \theta_{\text{i}} = 10000^{\text{-2(i-1)/d}}, i \in [1, 2, ..., d/2]. \quad (10)$$

In contrast to the additive nature of the position embedding methods used by other works, their approach is multiplicative. Moreover, RoPE naturally incorporates relative position information through rotation matrix product instead of altering terms in the expanded formulation of additive position encoding when applied with self-attention.

6 Incorporation of Switching Point Information in CMLM

Positional encodings help the transformer learn dependencies between tokens at different positions of the input sequence. To enhance the positional encodings for code-mixed text, we modify the rotatory positional encoding to incorporate switching point information.

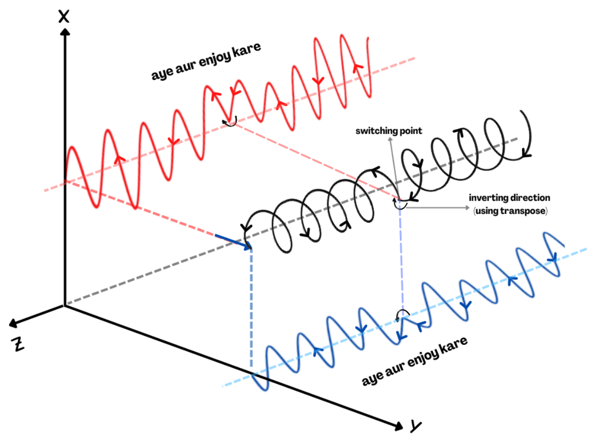

Figure 2: Visual intuition for our rotary approach with switching point incorporation. We consider a linearly polarized electromagnetic wave and show the change in rotation whenever a switching point occurs.

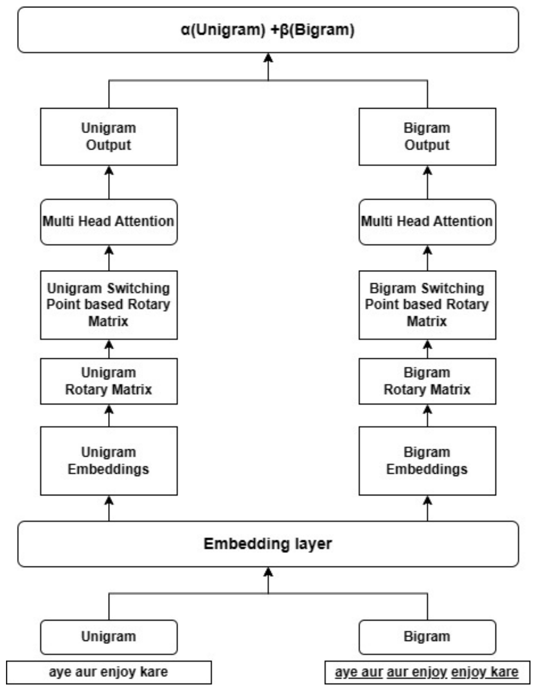

Figure 3: This diagram depicts the higher level understanding of the proposed positional embeddings.

6.1 Switching Point-based Rotary Matrix

Switching points are a potential bottleneck for code-mixing language modeling and to address this problem, we incorporate switching point based rotary positional encoding in our architecture. The intuition behind RoPE is electromagnetic waves. The embeddings are represented as complex numbers and the positions are represented as pure rotations that are applied to them. Keeping this in mind, we address the problem of switching points (SP) with the help of angles that participate in RoPE. Whenever we encounter a switching point, we change the rotation, i.e., we change the direction of these angles. To implement the rotation change, we define a switching point matrix. The switching point matrix helps our model identify and learn the patterns of code mixing in the corpus. Our matrix is defined with 1s and -1s. When there is a language shift (L1 $\rightarrow$ L2) or (L2 $\rightarrow$ L1), i.e., when we encounter a switching point, we annotate the column value as -1 and for the successive words in L2, we annotate column values as 1 until another switching point occurs.

$$
\begin{aligned}
&SPM \in R^d_{n*n} \\
&\text{if i == SP:} \\
&\qquad SPM_i = -1 \qquad (11) \\
&\text{else:} \\
&\qquad SPM_i = 1
\end{aligned}
$$

The visual intuition of our approach is shown in Figure 2. The switching point matrix (SPM) with 1s and -1s is defined in such a way that it transposes the rotary matrix, intuitively inverting the rotation at every switching point encounter. Therefore, the final matrix, i.e., switching point rotary matrix (SPRM) is a result of element-wise multiplication of the defined switching point matrix (SPM) with rotary matrix (RM):

$$SPRM = SPM \times RM \qquad (12)$$

$$e_{ij}^{SPRotary} = \frac{1}{\sqrt{d}}\left(SPRM^d_{\Theta,i}W^{Q,1}(x_i)\right)^T\left(SPRM^d_{\Theta,j}W^{K,1}(x_i)\right) \qquad (13)$$

6.2 Bigram and Switching Point-based Rotary Positional Encoding (BSPRoPE)

Since the language changes at the SPs, we get two consecutive tokens with different language hence we also incorporate the bigram level information in our model. In this positional encoding method, we get positional information among the bigrams in an utterance. We use the technique of switching point based rotary positional encoding at a word-to-word level and at bigram level as depicted in Figures 3,4 and mathematically expressed as Equation 16

$$e_{ij}^{UniSPRotary} = \frac{1}{\sqrt{d}}\left(SPRM^d_{\Theta,i}W^{Q,1}(x_i)\right)^T\left(SPRM^d_{\Theta,j}W^{K,1}(x_j)\right) \qquad (14)$$

$$e_{ij}^{BiSPRotary} = \frac{1}{\sqrt{d}}\left(SPRM^d_{\Theta,i}W^{Q,1}(x_i)\right)^T\left(SPRM^d_{\Theta,j}W^{K,1}(x_j)\right) \qquad (15)$$

$$prediction = a * e_{ij}^{UnigramSPRotary} + b * e_{ij}^{BigramSPRotary} \qquad (16)$$

where a and b are learnable coefficients. x_i and x_i in equation 14 refer to unigram inputs whereas as in equation 15 they refer to bigram inputs.

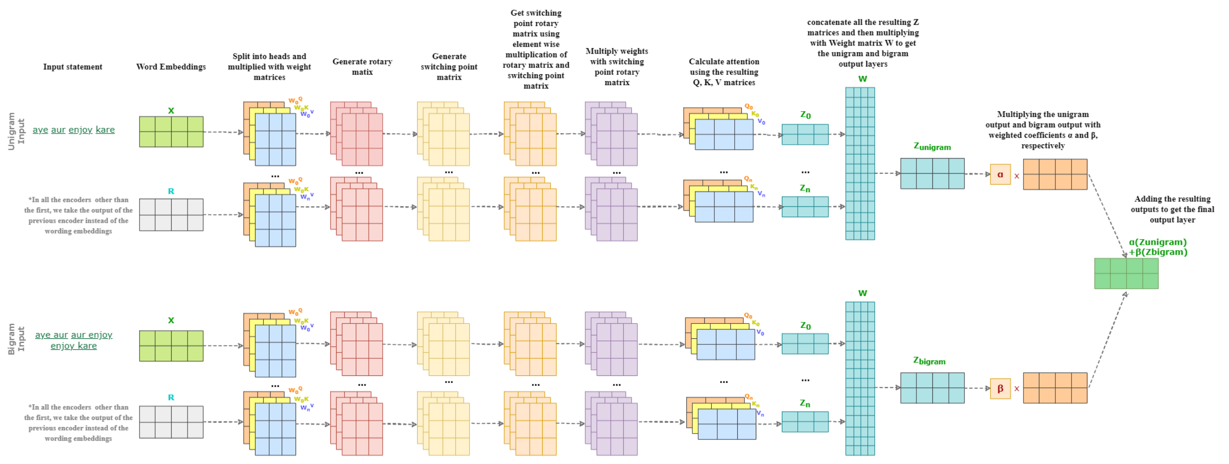
Input statement
Word Embeddings
Split into heads and multiplied with weight matrices
Generate rotary matix
Generate switching point matrix
Get switching point rotary matrix using element wise multiplication of rotary matrix and switching point matrix
Multiply weights with switching point rotary matrix
Calculate attention using the resulting Q, K, V matrices
concatenate all the resulting Z matrices and then multiplying with Weight matrix W to get the unigram and bigram output layers
Multiplying the unigram output and bigram output with weighted coefficients α and β, respectively
Adding the resulting outputs to get the final output layer
Unigram Input
aye aur enjoy kare
Bigram Input
aye aur aur enjoy enjoy kare
*In all the encoders other than the first, we take the output of the previous encoder instead of the wording embeddings
X
R
W
Z0
Zn
Zunigram
Zbigram
α
β
α(Zunigram) +β(Zbigram)

Figure 4: CONFLATOR architecture within the encoder layer. It depicts how the unigrams and bigrams of the input statement are passed as inputs to our encoder decoder architecture. In this framework, we generate a rotary matrix and a switching point matrix. By performing element-wise multiplication of the aforementioned matrices, we get our proposed novel switching point based rotary matrix. We represent the embeddings as complex numbers and their positions as pure rotations that we apply to them with the help of our switching point based rotary matrix. Then, upon getting the output layers for unigram and bigram statements separately. We introduce weighted coefficients a and b for unigram outputs and bigram outputs, respectively. We get our final output layer by adding these weighted unigram and bigram outputs.

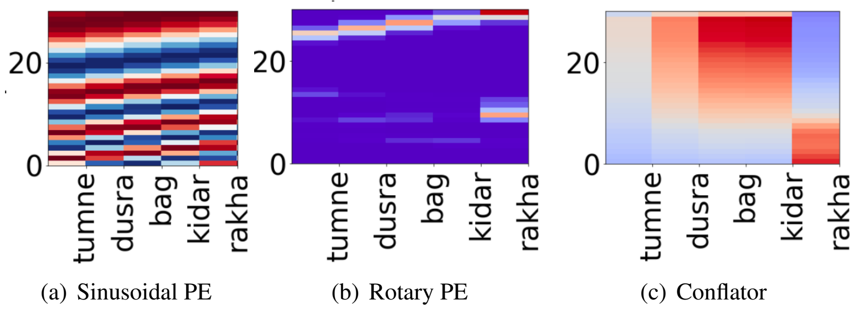

Figure 5: CONFLATOR is able to differentiate words coming from different positions and give high attention when a switching point occurs (at bag_{EN} and $kidar_{HI}$) while the other models cannot do so.

6.3 CONFLATOR Architecture

The local dependencies for Unigram and Bigram (Word2Vec trained from scratch) along with unigram and bigram SPRM are fed to a 6-headed Multi-Head attention (MHA) in each encoder layer of the transformer separately, resulting in 2 attention matrices. We introduce 2 learnable parameters α and β that are used as weight coefficient for the unigram and bigram matrix respectively. The final matrix is passed to the decoder layer. The embedding and architecture in depicted in figs. 3 and 4.

7 Experiments and Results

For our base models, each training step takes about 0.5 seconds. We train the base models for a total of 100,000 steps or 12 hours. For the big models like bigram and SPM-based models, the step time is 1.0 seconds. The big models were trained for 250,000

CMI Range	Transformer	GPT-2	BERT	Conflator
0-10	1018.54	823.71	666.48	492.96
11-20	1210.11	967.01	782.19	501.44
21-30	1401.37	1334.72	1007.34	544.71
31-40	2688.00	2334.73	1007.34	800.62
41-50	4421.22	3905.87	4337.02	1095.12
Average	2147.85	1873.20	1701.49	**578**

Table 2: Perplexity comparison between different models based on ranges of CMI. Lower Perplexity is better.

steps (2 days). We use ADAM optimizer with β_1 = 0.9, β_2 = 0.98 and ϵ = 1e^{-9}. We use the method of varying the learning rate over the course of training from Vaswani et al. (2017b).

We use two types of regularization during our training process: We apply dropout to the output of each encoder and decoder layer followed by

Models	Positional representation							Bigram	F1 (%)
	Sin/Cos	Index	Dynamic	SPI	Relative	RM	SPRM		
Word2Vec + LSTM	✗	✗	✗	✗	✗	✗	✗	✗	56
BERT	✓	✗	✗	✗	✗	✗	✗	✗	60
3HA + Sinusoidal PE	✓	✓	✗	✗	✗	✗	✗	✗	74.34
3HA + Dynamic PE	✗	✓	✓	✗	✗	✗	✗	✗	75.02
3HA + Relative PE	✗	✗	✗	✗	✓	✗	✗	✗	75.32
3HA + Rotary PE	✓	✓	✗	✗	✗	✓	✗	✗	76.04
SOTA (PESTO)	✗	✗	✓	✓	✓	✗	✗	✗	75.6
Unigram SP Relative (USPR)	✗	✗	✗	✓	✓	✗	✗	✗	75
Bigram SP Relative BSPR)	✗	✗	✓	✓	✓	✗	✗	✓	75
Unigram SPRoPE + Good Tuning	✓	✓	✗	✓	✗	✓	✓	✗	74.6
Unigram SPRoPE	✓	✓	✗	✓	✗	✓	✓	✗	75
Conflator (BSPRoPE)	✓	✓	✗	✓	✗	✓	✓	✓	76.23
Conflator with StableLM	✓	✓	✗	✓	✗	✓	✓	✓	76.11
Conflator with Alpaca	✓	✓	✗	✓	✗	✓	✓	✓	75.69
Conflator with LLaMA	✓	✓	✗	✓	✗	✓	✓	✓	**76.45**

Table 3: Results of various position sensitive experiments for ***Sentiment Analysis*** on CM text. *n*HA refers to n-headed attention.

Models	Positional representation							Bigram	BLEU
	Sin/Cos	Index	Dynamic	SPI	Relative	RM	SPRM		
3HA + Sinusoidal PE	✓	✓	✗	✗	✗	✗	✗	✗	17.2
3HA + Dynamic PE	✗	✓	✓	✗	✗	✗	✗	✗	17.9
3HA + Relative PE	✗	✗	✗	✗	✓	✗	✗	✗	18.4
3HA + Rotary PE	✓	✓	✗	✗	✗	✓	✗	✗	24.9
SOTA (IIITH-mrinaldhar)	✗	✗	✓	✓	✓	✗	✗	✗	28.4
Unigram SP Relative (USPR)	✗	✗	✗	✓	✓	✗	✗	✗	9.8
Bigram SP Relative (BSPR)	✗	✗	✓	✓	✓	✗	✗	✓	7.6
Unigram SPRoPE	✓	✓	✗	✓	✗	✓	✓	✗	29.1
Conflator (BSPRoPE)	✓	✓	✗	✓	✗	✓	✓	✓	25.16
Conflator with StableLM	✓	✓	✗	✓	✗	✓	✓	✓	29.06
Conflator with Alpaca	✓	✓	✗	✓	✗	✓	✓	✓	29.89
Conflator with LLaMA	✓	✓	✗	✓	✗	✓	✓	✓	**30.15**

Table 4: Results of position sensitive experiments for ***Machine Translation*** on CM text. Higher BLEU is better.

Normalization. In addition, we apply dropout and normalization to the sums of the word embeddings and the positional encodings in both the encoder and decoder layers. We use a rate of $P_{drop} = 0.2$.

Intrinsic Evaluation: The perplexity scores of baseline language models in comparison with CONFLATOR on code-mixed language modeling task are shown in 2. We see that our model performs much better than other models.

Extrinsic Evaluation: We evaluate our model on two downstream tasks: (i) sentiment analysis, and (ii) machine translation. For sentiment analysis, (Table. 3) we use the data provided by Patwa et al. (2020a). CONFLATOR achieves 76.23% F1 score and outperforms the SOTA (Ali et al., 2022). The main reason for this is learning SP by aggregating with the help of rotary positional encoding with a variable length MHA framework. For the machine translation (Table 4), we use the data provided by (Dhar et al., 2018). We achieve 29.1 bleu score and outperform the SOTA (Dhar et al., 2018) using the Unigram SPRoPE model which is able to learn the patterns of language mixing with the help of switching point based rotary positional encoding.

8 Conclusion & Takeaways

In this work, we report experiments on *Hinglish* sentiment analysis and Machine translation problems through the lens of language modeling. Our contribution could be seen as following:

(i) We introduce the idea of switching point based rotary positional encoding. Whenever a switching point is encountered, we incorporate rotation change to learn the patterns of language mixing.

(ii) We introduce CONFLATOR, a neural language modeling approach for code-mixed languages. CONFLATOR tries to learn better representations by means of switching point-based rotary positional encoding, initially at unigram level and then at bigram level.

(iii) We empirically prove that CONFLATOR is learning the patterns of code-mixing which other models with different positional encodings prove unsuccessful, as shown in Figure 5.

(iv) It is also noteworthy that CONFLATOR achieves comparable to SOTA results even without any pre-trained heavy language model.

9 Limitations

Although our bigram model achieves SOTA on sentiment analysis using unigram, it is slightly behind the bigram model when it comes to machine translation, where using bigram at the decoder level resulted in poor performance. Despite conducting extensive experiments, there lacks a detailed explanation on why the bigram-based approach for MT fails. Future experiments will focus on exploring or understanding the issue of bigrams for MT and coming up with a solution for the same.

References

Mohsin Ali, Sai Teja Kandukuri, Sumanth Manduru, Parth Patwa, and Amitava Das. 2022. Pesto: Switching point based dynamic and relative positional encoding for code-mixed languages (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36.

Segun Taofeek Aroyehun and Alexander Gelbukh. 2018. Aggression detection in social media: Using deep neural networks, data augmentation, and pseudo labeling. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 90–97, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Shubhanker Banerjee, Bharathi Raja Chakravarthi, and John P McCrae. 2020. Comparison of pretrained embeddings to identify hate speech in indian code-mixed text. In *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 21–25. IEEE.

Utsab Barman, Amitava Das, Joachim Wagner, and Jennifer Foster. 2014. Code mixing: A challenge for language identification in the language of social media. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 13–23, Doha, Qatar. Association for Computational Linguistics.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Aditya Bohra, Deepanshu Vijay, Vinay Singh, Syed Sarfaraz Akhtar, and Manish Shrivastava. 2018. A dataset of hindi-english code-mixed social media text for hate speech detection. In *Proceedings of the second workshop on computational modeling of people's opinions, personality, and emotions in social media*, pages 36–41.

Eyamba G Bokamba. 1988. Code-mixing, language variation, and linguistic theory:: Evidence from bantu languages. *Lingua*, 76(1):21–62.

Bharathi Raja Chakravarthi, Vigneshwaran Muralidaran, Ruba Priyadharshini, and John P McCrae. 2020. Corpus creation for sentiment analysis in code-mixed tamil-english text. *arXiv preprint arXiv:2006.00206*.

Arindam Chatterjere, Vineeth Guptha, Parul Chopra, and Amitava Das. 2020. Minority positive sampling for switching points - an anecdote for the code-mixing language modeling. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 6230–6238, Marseille, France. European Language Resources Association.

Amitava Das and Björn Gambäck. 2014. Identifying languages at the word level in code-mixed indian social media text.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018a. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018b. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Mrinal Dhar, Vaibhav Kumar, and Manish Shrivastava. 2018. Enabling code-mixed translation: Parallel corpus creation and MT augmentation approach. In *Proceedings of the First Workshop on Linguistic Resources for Natural Language Processing*, pages 131–140, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Björn Gambäck and Amitava Das. 2016. Comparing the level of code-switching in corpora. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1850–1855, Portorož, Slovenia. European Language Resources Association (ELRA).

Vinay Gopalan and Mark Hopkins. 2020. Reed at semeval-2020 task 9: Sentiment analysis on code-mixed tweets. *CoRR*, abs/2007.13061.

Madan Gopal Jhanwar and Arpita Das. 2018. An ensemble model for sentiment analysis of hindi-english code-mixed data. *arXiv preprint arXiv:1806.04450*.

Abhishek Laddha, Mohamed Hanoosh, Debdoot Mukherjee, Parth Patwa, and Ankur Narang. 2020. Understanding chat messages for sticker recommendation in messaging apps. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(08):13156–13163.

Abhishek Laddha, Mohamed Hanoosh, Debdoot Mukherjee, Parth Patwa, and Ankur Narang. 2022. Large scale multilingual sticker recommendation in messaging apps. *AI Magazine*, 42(4):16–28.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining.

Xuanqing Liu et al. 2020. Learning to encode position for transformer with continuous dynamical model. In *ICML 2020*.

Yili Ma, Liang Zhao, and Jie Hao. 2020. Xlp at semeval-2020 task 9: Cross-lingual models with focal loss for sentiment analysis of code-mixing language. In *Proceedings of the fourteenth workshop on semantic evaluation*, pages 975–980.

Ravindra Nayak and Raviraj Joshi. 2022. L3cube-hingcorpus and hingbert: A code mixed hindi-english dataset and bert language models.

Braja Gopal Patra, Dipankar Das, and Amitava Das. 2018. Sentiment analysis of code-mixed indian languages: An overview of sail_code-mixed shared task @icon-2017. *CoRR*, abs/1803.06745.

Parth Patwa, Gustavo Aguilar, Sudipta Kar, Suraj Pandey, Srinivas PYKL, Björn Gambäck, Tanmoy Chakraborty, Thamar Solorio, and Amitava Das. 2020a. SemEval-2020 task 9: Overview of sentiment analysis of code-mixed tweets. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 774–790, Barcelona (online). International Committee for Computational Linguistics.

Parth Patwa, Srinivas Pykl, Amitava Das, Prerana Mukherjee, and Viswanath Pulabaigari. 2020b. Hater-O-genius aggression classification using capsule networks. In *Proceedings of the 17th International Conference on Natural Language Processing (ICON)*, pages 149–154, Indian Institute of Technology Patna, Patna, India. NLP Association of India (NLPAI).

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Niloofar Safi Samghabadi, Parth Patwa, Srinivas Pykl, Prerana Mukherjee, Amitava Das, and Thamar Solorio. 2020. Aggression and misogyny detection using bert: A multi-task approach. In *Proceedings of the second workshop on trolling, aggression and cyberbullying*, pages 126–131.

Ayan Sengupta, Sourabh Kumar Bhattacharjee, Tanmoy Chakraborty, and Md Shad Akhtar. 2021. Hit: A hierarchically fused deep attention network for robust code-mixed language representation.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *NAACL 2018*.

Rajendra Singh. 1985. Grammatical constraints on code-mixing: Evidence from hindi-english. *Canadian Journal of Linguistics/Revue canadienne de linguistique*, 30(1):33–45.

Vivek Srivastava and Mayank Singh. 2020. Phinc: A parallel hinglish social media code-mixed corpus for machine translation. *arXiv preprint arXiv:2004.09447*.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864.

Tiancheng Tang, Xinhuai Tang, and Tianyi Yuan. 2020. Fine-tuning bert for multi-label sentiment analysis in unbalanced code-switching text. *IEEE Access*, 8:193248–193256.

Debapriya Tula, Prathyush Potluri, Shreyas Ms, Sumanth Doddapaneni, Pranjal Sahu, Rohan Sukumaran, and Parth Patwa. 2021. Bitions@DravidianLangTech-EACL2021: Ensemble of multilingual language models with pseudo labeling for offence detection in Dravidian languages. In *Proceedings of the First Workshop on Speech and Language Technologies for Dravidian Languages*, pages 291–299, Kyiv. Association for Computational Linguistics.

Debapriya Tula, MS Shreyas, Viswanatha Reddy, Pranjal Sahu, Sumanth Doddapaneni, Prathyush Potluri, Rohan Sukumaran, and Parth Patwa. 2022. Offence detection in dravidian languages using code-mixing index-based focal loss. *SN Computer Science*, 3(5):330.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017a. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017b. Attention is all you need. *CoRR*, abs/1706.03762.

S.K. Verma. 1976. Code-switching: Hindi-english. *Lingua*, 38(2):153–165.

Yogarshi Vyas, Spandana Gella, Jatin Sharma, Kalika Bali, and Monojit Choudhury. 2014. Pos tagging of english-hindi code-mixed social media content. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 974–979.