ABBIE: Attention-Based BI-Encoders for Predicting Where to Split Compound Sanskrit Words

Irfan Ali[1,*][a], Liliana Lo Presti[1][b], Igor Spano[2][c] and Marco La Cascia[1][d]
[1] *Department of Engineering, University of Palermo, Palermo, Italy*
[2] *Department of Cultures and Society, University of Palermo, Palermo, Italy*

Keywords: Word Segmentation, Sanskrit Language, Sandhi Rule, Bi-Encoders, Attention.

Abstract: Sanskrit is a highly composite language, morphologically and phonetically complex. One of the major challenges in processing Sanskrit is the splitting of compound words that are merged phonetically. Recognizing the exact location of splits in a compound word is difficult since several possible splits can be found, but only a few of them are semantically meaningful. This paper proposes a novel deep learning method that uses two bi-encoders and a multi-head attention module to predict the valid split location in Sanskrit compound words. The two bi-encoders process the input sequence in direct and reverse order respectively. The model learns the character-level context in which the splitting occurs by exploiting the correlation between the direct and reverse dynamics of the characters sequence. The results of the proposed model are compared with a state-of-the-art technique that adopts a bidirectional recurrent network to solve the same task. Experimental results show that the proposed model correctly identifies where the compound word should be split into its components in 89.27% of cases, outperforming the state-of-the-art technique. The paper also proposes a dataset developed from the repository of the Digital Corpus of Sanskrit (DCS) and the University of Hyderabad (UoH) corpus.

1 INTRODUCTION

NLP systems have come to a range of importance in recent years. The automation of text and speech processing tasks is facilitating daily life with enhanced ease and comfort. However, this progress benefits mostly the well-resourced and well-tooled languages. Computationally demanding languages still suffer from underperforming NLP systems. One such language is Sanskrit, an ancient Indian language with texts going as far back as 1500 B.C. (Hellwig, 2015). The digitization of Sanskrit scriptures has given impetus to research in most Sanskrit NLP tasks, but current efforts are mostly oriented towards general Sanskrit literature and scriptures (Goyal et al., 2012)

A feature of Sanskrit that makes it more complex to process is the use of compound words obtained from the phonetic union of words. In order to understand the meaning of the compound word, it is necessary to identify the words that compose it and, therefore, split the compound word into its component words (see Figure 1).

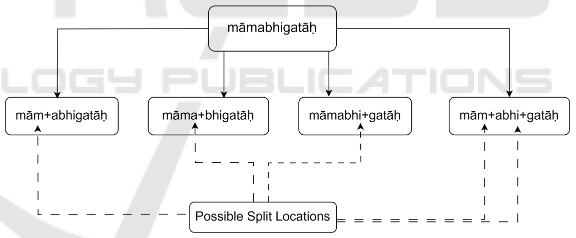

Figure 1: Different possible split locations of the compound word *māmabhigatāḥ* given by the standard Sandhi splitter. All splits are syntactically correct but only the first one is semantically meaningful.

Sanskrit word splitting is similar to other word segmentation tasks for Asian languages, such as Thai (Haruechaiyasak et al., 2008), Chinese, and Japanese (i.e., Kanji). Nowadays, most of the research is oriented towards Chinese and Japanese and focuses on enhancing the performance of single segmentation criterion. To reduce feature engineering efforts (Zheng et al., 2013; Pei et al., 2014; Cai and Zhao, 2016; Yao and Huang, 2016), in Chinese language, compound words are segmented through a sequence labeling process to assign labels to each character of the compound word and segment the component words.

[a] https://orcid.org/0009-0008-1664-5125
[b] https://orcid.org/0000-0003-0833-4403
[c] https://orcid.org/0000-0003-0304-0764
[d] https://orcid.org/0000-0002-8766-6395
[*] Corresponding author

Ali, I., Lo Presti, L., Spano, I. and La Cascia, M.
ABBIE: Attention-Based BI-Encoders for Predicting Where to Split Compound Sanskrit Words.
DOI: 10.5220/0013155300003890
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 2*, pages 334-344
ISBN: 978-989-758-737-5; ISSN: 2184-433X
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

In Sanskrit, the same process is generally accomplished by applying the Sandhi rule (Krishna et al., 2016). Besides Sandhi, a number of other morphological processes are involved in Sanskrit word formation, especially compounding and inflection. However, Sandhi deals explicitly with phonological and orthographic processes at word junctions or between words in a sentence. It is a process that changes the sounds and, on occasion, the structure of words, important for exact interpretation and generation of text in Sanskrit. Although compounding and inflection are also of prime importance for Sanskrit grammar, Sandhi is more crucial for it because it has a direct influence on the joining and pronunciation of words and hence their syntactic and semantic meaning. Segmenting Sanskrit words is not a trivial task because the Sandhi phenomenon causes phonetic transformations at word junctions. This not only results in word junctions being obscured, but the characters at the junctions are changed through deletion, insertion, and replacement operations. Learning how to predict the Sandhi split presents the additional challenge of not only correctly splitting compound words, but also predicting where to split them (Dave et al., 2021). The task is further complicated by the fact that the language is inflectional, with a large number of derivational affixes. Since Sanskrit compound words can be split at multiple locations, the split words will be syntactically correct but semantically meaningless. But till now researchers only consider the splitting of compound words that have one valid split location.

Figure 1 shows the possible split locations of the compound word *māmabhigatāḥ*. Although several splits are possible, only *mām+abhigatāḥ* is syntactically and semantically meaningful. Determining the semantically correct split for a word is context-dependent.

To account for this challenge, given in input a compound word, the method in (Dave et al., 2021) uses a bidirectional-LSTM to predict the Sandhi-window, which is a portion of the compound word wherein the Sandhi transformation has taken place. The model outputs a sequence where only the characters in the Sandhi-window are marked 1. All other characters are marked 0 (see Figure 2).

Similarly to (Dave et al., 2021), our model takes a Sanskrit compound word, i.e., a sequence of characters, as input. The model returns a sequence of 0 and 1 values, with 1 indicating the split location of the compound word. In contrast to (Dave et al., 2021), our novel model uses two BiLSTMs and a multi-head attention layer to predict the split location in the Sanskrit compound word. To the best of our knowledge this is the first work where two Bidirectional-LSTMs act as encoders to capture the character-level contextual information of the Sanskrit compound word. These two encoders process the input sequence in direct and reverse order and, thus, will be named direct and reverse encoders respectively. As highlighted in (Sutskever et al., 2014), the processing order of the input sequence allows for capturing diverse information about the character-level contexts of each input character. Furthermore, multi-head attention layers focus on different parts of the sequence, i.e., short-term and long-term dependencies, to capture the correlations between the outputs of the direct and reverse encoders and predict the exact split point in the Sanskrit compound word. Using two parallel BiLSTMs, a network can learn from sequential data with more enriched, and subtle information captured, while higher model capacity is achieved at minimal overhead and computational complexity. This in turn helps improve the performance of predicting the location of Sanskrit compound word. To train and validate our model, we augmented the dataset used in (Dave et al., 2021) with new compound words and their respective splits collected from the Digital Corpus of Sanskrit (Hellwig, 2021). Incorrect splits, and words not following the Sandhi rule of compound word formation were manually discarded.

Overall, this paper proposes:

- A novel multi-headed attention-based bi-encoder model that exploits direct and reverse dynamics at a character level to predict the split location in compound Sanskrit words; The code implementation will be made publicly available to ease future comparisons.
- An enhanced dataset of compound Sanskrit words following the Sandhi rule with their respective split points. The dataset will be made publicly available to the research community[1].

In the following, we first describe the main motivation behind solving the described problem, then discuss related works and illustrate the proposed neural architecture. Finally, we present and discuss our experimental results and conclusions at the end.

2 MOTIVATION

Sandhi splitting is important for proper understanding and analysis of Sanskrit text, as it reveals the correct meaning of each word within the compound words. The difficulty in applying Sandhi split lies in the ability to apply the correct division of compound words.

[1]Code and dataset available at https://github.com/IrfanAliBabar/ABBIE-Sanskrit

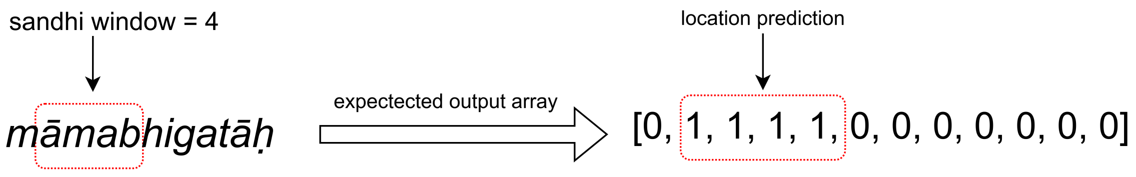

Figure 2: Sandhi window for predicting the location in a compound word.

We stress here that Sanskrit compound words can be divided at more than one point. Although the divided words will be syntactically correct, they can be semantically meaningless.

In this context, the work in (Patankar, 2023) highlights the need for tools that can effectively apply Sandhi splitting to ensure comprehensive morphological analysis. Such tools would enable computational linguistic research with better analysis and advanced models. Predicting the correct splitting location in Sanskrit compound words is important to develop better text understanding that supports digital preservation, revealing the cultural and historical significance of Sanskrit texts, and filling the resource gaps.

The few currently available Sandhi splitting tools use rule-based implementations (Linguistics, 2 24) and have significant drawbacks in that they are rigid and cannot handle exceptions or variations. Using these tools requires a lot of time and resources to maintain and update these rules as new linguistic phenomena are discovered. One of the main issues is maintaining consistency among these rules, which requires a great linguistic expertise. Furthermore, the performance of a rule-based system with very diverse data sets may not be good if the rules are too narrow leading to inconsistent results and, under conditions of incomplete or incorrect rules, to errors that are very difficult to correct. Addressing these drawbacks with better techniques, mostly based on artificial intelligence, could improve the reliability of Sanskrit text analysis tools.

Better methods of Sandhi splitting will be of great help in language learning and teaching, making Sanskrit user-friendly for both teachers and students. Additionally, learning to predict the position of the Sandhi split can provide interesting linguistic insights into word formation in almost all Dravidian languages. Finally, modeling how words are formed in Sanskrit can provide an NLP framework for organizing words in other Indian languages.

3 RELATED WORK

Most of the NLP systems for Sanskrit word splitting combine *Panini's* phonetic and morphological rules with lexical resources.

These systems are based on the application of formal methods (Huet, 2005; Goyal et al., 2007; Kulkarni and Shukl, 2009), or they use statistical approaches such as Dirichlet processes (Natarajan and Charniak, 2011), or finite-state methods (Mittal, 2010), graph queries (Krishna et al., 2016), or hybrid systems (Haruechaiyasak et al., 2008).

The biggest challenge to splitting words in Sanskrit is to find out the most semantically accurate word segmentation among all the possible splits of a compound word. (Krishna et al., 2016) solved this issue by modeling word segmentation as a query expansion task under a path-constrained random walks (PCRW) framework. They further fine-tuned their model by adding morphological information using Inductive Logic Programming (ILP). This resulted in significant improvement in the performance measures.

Several works have proposed the use of recurrent neural network to solve the problem. The paper (Hellwig, 2015) presents a neural network-based method that does compound splitting and Sandhi resolution jointly in Sanskrit text. The paper uses Long Short-Term Memory cells for labeling tasks in Sanskrit analysis. Hellwig proposed an alternative method of Sandhi resolution in Sanskrit by developing a classifier that depends on the gold standard string splits. The source sequence represents a string split in phonemes, while the target sequence presents transformations applied to each phoneme. A classifier will then be trained to correctly split compounds and apply the appropriate Sandhi resolution in the process, generating the corresponding Sandhi rule. Five possible transformations rules, namely *R1-5*, are defined that will guide the classification of phonemes in each string. Next, the paper (Hellwig and Nehrdich, 2018) proposes an end-to-end trained neural network for Sanskrit tokenization, which jointly performs compound splitting and resolves phonetic mergers (Sandhi), requiring neither feature engineering nor outside linguistic resources but working on parallel versions of raw and segmented text alone. The models are designed to work at a character level and allows word splitting in a sequence labeling framework for the Sanskrit language. Their best model uses convolutional and recurrent elements with shortcut connections (rcNNshort).

The work in (Dave et al., 2021) also presents neural networks for splitting Sanskrit compound words, focusing specifically on Sandhi splitting. The work formulates a sequence-to-sequence prediction problem, employing recurrent neural networks (RNNs) in a two-step process. Without additional lexical resources or a priori information, the model takes a compound word as input and generates the split of compound words separated by a "+" character as output. This data-driven approach demonstrates superior performance compared to existing methods without needing extra lexical or morphological resources.

Some works have included attention to solve the problem. In (Aralikatte et al., 2018), the authors presented an attention-based deep learning method to determine the split position of compound words. Based on the predicted position of split, the method determines the Sandhi components by graphically segmenting the compound word. Compared to rule-based Sandhi models, knowing the split position allows to massively reduce the number of possible splits to check. In (Reddy et al., 2018), the model primarily identifies word splits and the correctness of unsandhied strings from a sandhied input string, which shall not take into account morphological and semantic details. The paper shows that the attention module improved the results significantly.

The problem of splitting compound words has also been studied for other languages, such as Chinese. In (Gong et al., 2019), the authors described a model that automatically switches between several segmentation criteria to improve the segmentation process with the assistance of multiple LSTM networks and a switcher. The model mainly follows two solution approaches: 1. Several LSTM cells that represent the criteria of segmentation and a switcher for routing between the LSTMs. 2. Switch-LSTM offers a more flexible solution to the multi-criteria Chinese word segmentation problem and supports knowledge transfer to the new criteria. The use of multiple segmentation criteria is important not only for enhanced performance but also for knowledge transfer and flexibility, leading to improvements in multi-criteria Chinese word segmentation.

The work in (Chen et al., 2015) addressed Chinese word segmentation as a sequence labeling task and compared several stacked bidirectional recurrent architectures. A final sentence-level likelihood layer (Collobert et al., 2011) is added to maximize the transition score for the target sequence represented using BMES encoding. Their best model was based on a single-layer bidirectional LSTM with bigrams of pre-trained character embeddings supplied as inputs

Our proposed architecture is totally different from all the models mentioned above. It uses two bi-encoders to extract contextual information from the input sequence and the reverse input sequence. It can capture different types of relationships and dependencies in the character-level input sequence through the use of a multi-head attention module that correlates the outputs of the two bi-encoders to predict the accurate split position in the Sanskrit compound word.

3.1 Existing Tools

Three prominent Sandhi splitter tools are available in the open domain. These are: (i) JNU splitter (Kumar, 2007), (ii) UoH splitter (Kumar et al., 2010), and finally, (iii) INRIA Sanskrit Reader Companion (Huet, 2003) (Goyal and Huet, 2013). All these tools addressed the challenge of splitting differently, but they work on a principle that is essentially the same. For a given compound word, an exhaustive set of rules is applied to every character thus yielding an extensive set of possible word splits. Subsequently, a morpheme dictionary of Sanskrit words, together with a variety of heuristics, is utilized to remove infeasible combinations of splits. However, none of these approaches effectively solves the inherent problem of first identifying the location of the split to which the rules need to be applied. This would help narrow the range of applicable rules and possibly produce more accurate splits.

4 PROPOSED NEURAL ARCHITECTURE: ABBIE

Sanskrit compound word splitting can be viewed as a sequence-to-sequence problem where the input is the character sequence of the compound word, and the output is a sequence of the same length where each element takes values in $[0 \dots 1]$ and indicates whether a splitting occurs (1) or not (0). Therefore, all elements in these output sequences are 0 except the elements corresponding to the predicted splitting positions which are set to 1.

To make the prediction more robust, we consider a splitting window of size 4 in which the splitting can occur. The split occurs after the second element takes on the value 1 within the window. For example, if the inputs are characters of the compound word *"māmabhigatāḥ"*, the expected output will be the sequence $[0,1,1,1,1,0,0,0,0,0,0,0]$. So the split occurs after the second *"m"* of the compound word.

More formally, given a compound word $X = \{x_1, x_2, x_3, \dots\dots, x_T\}$, our model analyzes the input se-

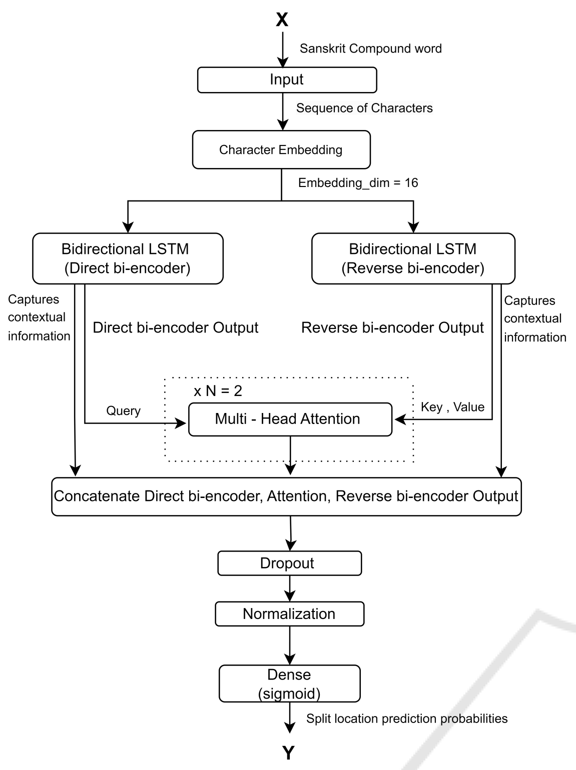
X
Sanskrit Compound word
Input
Sequence of Characters
Character Embedding
Embedding_dim = 16
Bidirectional LSTM (Direct bi-encoder)
Bidirectional LSTM (Reverse bi-encoder)
Captures contextual information
Direct bi-encoder Output
Reverse bi-encoder Output
Captures contextual information
x N = 2
Query
Multi - Head Attention
Key , Value
Concatenate Direct bi-encoder, Attention, Reverse bi-encoder Output
Dropout
Normalization
Dense (sigmoid)
Split location prediction probabilities
Y

Figure 3: Architecture of Sanskrit Compound Word Split Position Prediction Model.

quence X in direct and reverse order and, through attention mechanisms aimed at correlating the hidden states of the recurrent layers, for each input character x_t provides in output $y_t \in L = \{0,1\}$. The label y_t indicates whether the character is within the split window. Once the split window is calculated, the split position is identified as being between the second and third elements in the window.

To solve the Sandhi splitting problem, we propose ABBIE, whose architecture includes three main components:

1. a character Embedding Layer;
2. two Bi-LSTMs to encode character-level contextual information of the forward (direct) and backward (reverse) sequence;
3. multi-head attention layers to focus on different parts of the sequence (such as short-term and long-term dependencies) and capture more information.

Figure 3 shows the proposed architecture to predict the split location in the compound Sanskrit word. In the following, we provide a detailed description of each component.

4.1 Embedding Layer

Characters of the input sequence belongs to an alphabet (set of language symbols) of size V. We used a one-hot-encoding strategy to represent the characters of the given alphabet. Then, our model maps the language symbols into a learned embedding space.

Given an input sequence $X \in \mathbb{R}^{T\times V}$ where T is the sequence length, the embedding is achieved by using an Embedding Matrix $E \in \mathbb{R}^{V\times d}$ with d representing the embedding dimension. The embedding operation produces the embedded input E_X and can be written as:

$$E_X = XE, \quad \text{where } E_X \in \mathbb{R}^{T\times d}. \tag{1}$$

4.2 Direct and Reverse Bi-Encoders

We used two Bi-LSTMs (Hochreiter and Schmidhuber, 1997) as bi-encoders, which typically refers to an architecture encoding input sequences in both forward and backward directions to capture full information about the context. This makes information from the whole sequence in the past and future available to the model and very useful for tasks where context is important.

In particular, the first bi-LSTM, named *direct encoder*, processes the given input sequence as it is and, at each time t, computes:

$$\text{Forward LSTM:} \quad \overrightarrow{h_t} = \text{LSTM}_{\text{fw}}(x_t, \overrightarrow{h_{t-1}}) \tag{2}$$

$$\text{Backward LSTM:} \quad \overleftarrow{h}_t = \text{LSTM}_{\text{bw}}(x_t, \overleftarrow{h_{t+1}}) \tag{3}$$

The output of the first bidirectional-LSTM, h_t^{direct}, is the concatenation of the forward and backward LSTM outputs:

$$h_t^{\text{direct}} = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right], \quad \text{where } h_t^{\text{direct}} \in \mathbb{R}^{2n} \tag{4}$$

and n is the number of cell units of the LSTM layer. Over the entire input sequence, the first bi-LSTM produce $H^{\text{direct}} \in \mathbb{R}^{T\times 2n}$, whose rows are the vectors h_t^{direct} for $t \in [1,\ldots,T]$.

The second bi-LSTM, named *reverse encoder*, processes the input sequence backward. At each time t, we consider the reversed input x'_t, and the second bi-LSTM will compute:

$$\text{Forward LSTM:} \quad \overrightarrow{h'_t} = \text{LSTM'}_{\text{fw}}(x'_t, \overrightarrow{h'_{t-1}}) \tag{5}$$

$$\text{Backward LSTM:} \quad \overleftarrow{h'_t} = \text{LSTM'}_{\text{bw}}(x'_t, \overleftarrow{h'_{t+1}}) \tag{6}$$

Similar to the first bi-LSTM, the output of the second bi-LSTM, h_t^{reverse}, is the concatenation of the outputs of its forward and backward LSTMs:

$$h_t^{\text{reverse}} = \left[\overrightarrow{h'_t}, \overleftarrow{h'_t}\right], \quad \text{where } h_t^{\text{reverse}} \in \mathbb{R}^{2n}. \tag{7}$$

Over the entire reversed input sequence, the second bi-LSTM produce $H^{\text{reverse}} \in \mathbb{R}^{T\times 2n}$, whose rows are the vectors h_t^{reverse} for $t \in [1,\ldots,T]$.

Hyperbolic tangent (tanh) activation function is used by all the LSTMs.

Using two bidirectional LSTMs that separately process the input sequence in different order (direct and reverse) has advantages over using a single Bi-LSTM with $2n$ units. The paper in (Sutskever et al., 2014) shows that the order (direct and reverse) of processing a sequence enables the LSTM to learn different information. In particular, a single Bi-LSTM learns the left and right context information of each input character. This will lead to a nuanced understanding of the preceding and subsequent sequence of characters at each sequential element.

In the proposed approach, we use two Bi-LSTMs. The direct encoder acquires the left (LSTM_{fw}) and right (LSTM_{bw}) context of each character of the sequence X. The reverse encoder analyzes the input sequence in reverse order, *reverse*(X). To put it simply, the reverse encoder learns the right (LSTM'_{fw}) and left (LSTM'_{bw}) context of the original (direct) sequence X. This is important in the subsequent processing stages when the outputs of the two bi-encoders is used in the multi-head attention layer.

Furthermore, two Bi-LSTMs with fewer units each provide better regularization and generalization than a single Bi-LSTM with many units. At the same time, independent regularization after each bi-encoder, especially the use of dropout, reduces the chances of overfitting. This effectively balances the complexity involved in splitting Sanskrit compounds and improves the overall performance of the model.

4.3 Multi-Head Attention Layers

One of the mechanisms incorporated into neural networks for natural language processing tasks, more particularly in the Transformer architecture, is the Multi-Head Attention (Vaswani et al., 2017). We utilize this mechanism with the bi-LSTM outputs to help our model learn to focus on different parts of the input sequence simultaneously and learn various aspects of relationships between different positions of the sequence. In particular, the multi-head attention layers will correlate different information about left and right contexts at a character-level. In the case of Sanskrit Sandhi splitting, this mechanism will help the model attend to different parts of the compound word to correctly identify the split positions.

Attention transforms inputs into three different vectors: query Q, key K, and value V. It is well known that the attention mechanism computes (Vaswani et al., 2017):

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \quad (8)$$

The idea behind attention is to compare a query vector against a set of key vectors to compute attention scores, which are used to create a weighted sum of the value vectors. The Softmax function is applied to the product $QK^\top$ to obtain a set of normalized attention weights to determine how relevant each position is relative to every other position in computing the output. The scaling factor $\sqrt{d_k}$ is used to prevent the dot product from being too large in magnitude, leading to unstable training and slow convergence.

In our model, the attention mechanism considers the outputs of the two bi-LSTMs as follows:

$$Q = H^{\text{direct}} \quad (9)$$

$$K = H^{\text{reverse}} \quad (10)$$

$$V = H^{\text{reverse}} \quad (11)$$

Multiple attention heads allow to capture diverse kinds of relationships and dependencies in the data, which is very helpful in a complex task such as predicting the split position. In particular,

$$\text{MultiHead}(Q,K,V) = [\text{head}_1,\ldots,\text{head}_H]W^O \quad (12)$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

and W^O is the output weight matrix.

In our model, the multi-head attention output A can be written as:

$$A = \text{MultiHead}(H^{\text{direct}}, H^{\text{inverse}}, H^{\text{inverse}}). \quad (13)$$

4.4 Target Sequence

The outputs of the two bi-encoders and of the multi-head attention module are concatenated:

$$H^{\text{merged}} = \text{Concat}\left(H^{\text{direct}}, A, H^{\text{reverse}}\right) \quad (14)$$

where $H^{\text{merged}} \in \mathbb{R}^{T\times 6n}$, and n is the hidden state size.

In this way, H^{merged} gives a richer, more informative representation of the input sequence that is useful to predict more accurately where the split occurs in the compound word. This concatenation guarantees that both the local context and the global dependencies, which the attention mechanism would capture, will be available to the dense layer (with one neuron), which is used to predict the splitting window. A sigmoid activation function is used by the dense layer.

4.5 Detecting the Split Location

Our main objective is to predict the most likely split location of a compound Sanskrit word based on the output of the model. The model gives an output sequence in which values can range between 0 and 1. As already stated, the model is trained so to predict a splitting window of length $input\_length = 4$.

The challenge lies in how to exploit the sequence of output values for the identification of the most probable window where the actual split occurs. Thus, we use a sliding window over the output sequence and select the window with the highest sum of values. This sliding window helps in smoothing the predictions and makes sure it considers a sequence of characters as a potential split location. Thus, for every position j in the sequence, we compute a sum of model's output values for the window starting at position j, ending at position $j + input\_length - 1$.

The starting position of this window, max_start is considered as the most likely starting point of the window split. Once the window split is found, the predicted split arises in the middle of the splitting window. In our implementation, being $input\_length = 4$, the split arises after the second element taking the value 1 within the window.

5 TRAINING AND IMPLEMENTATION DETAILS

ABBIE has been implemented in Python 3.10.14 with Keras API running on the TensorFlow backend. We used a character embedding size of 16, bi-directional encoders with 128 units. A dropout layer of 0.5 is applied after each Bi-LSTM. We used 2 attention heads in the multi-head attention layer and dropout layer 0.5 after the multi-head attention layer and the normalization layer. We used batches of size 64, and trained the model for 40 epochs by the Adam optimizer using a workstation equipped with an NVIDIA GeForce RTX 2070 GPU. The total number of trainable parameters in our model is 150,737. We adopted the mean squared error (MSE) as loss function.

6 EXPERIMENTAL RESULTS

In this section, we discuss data preparation, accuracy metric for our model, evaluation and comparison of our results with RNN-BiLSTM architecture (Dave et al., 2021). We also present ablation studies by comparing different variants of our model.

6.1 Dataset

We developed a new dataset containing a total number of compound words equal to 96535 from different books of the Digital Corpus of Sanskrit repository (Hellwig, 2021). We combined the dataset from the University of Hyderabad corpus (UoH, 2024), comprising 77842, with the data we collected. Overall, the full number of samples in the merged data before preprocessing was 174377. We pre-processed the merged data to remove duplicates. We discarded all compounds with more than two splits, incorrect splits, and words not following the Sandhi rules of compound word formation. Overall, the final dataset includes 99900 samples after the pre-processing. 80% of the data was used for training the model, and the remaining for testing. 10% of the training data is used for validation of the model. The dataset, originally in the IAST (International Alphabet of Sanskrit Transliteration) format (Hellwig, 2021), is pre-processed to convert to SLP1 (Sanskrit Library Phonetic) (Learn Sanskrit, 2024), a transliteration scheme better suited for computational analysis due to its phonetic nature.

6.2 Split Location Prediction Accuracy

To measure the accuracy of our model, we compare the predicted split location j with the true one j^*. Here, j indicates the character location within the compound word after which the split occurs.

The Split Location Prediction Accuracy is calculated as:

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} \delta(j, j^*)}{N} \tag{15}$$

where the function $\delta(j, j^*)$ returns 1 if $j = j^*$ and 0 otherwise. N is the number of compounds in the test set.

6.3 Results and Comparison

In Tables 1 and 2 we compare our model to the RNN-BiLSTM model presented in (Dave et al., 2021). We trained our ABBIE model by minimizing the MSE loss function on the dataset in (Dave et al., 2021). As shown in Table 1, the RNN-BiLSTM achieves an accuracy value of 92.3%, and our ABBIE model achieves a higher accuracy value of 93.1% on the same dataset and minimizing the same loss function.

However, the dataset in (Dave et al., 2021) is limited and smaller than ours. Our dataset is built from various religious books, poems, etc, and contains complex compound words as well. As shown in Table 2, on our dataset, the RNN-BiLSTM achieves a lower accuracy value, of 87%, suggesting that the proposed dataset is more challenging than the one proposed in (Dave et al., 2021). We stress here that we used the publicly available implementation of (Dave et al., 2021) to train the RNN-BiLSTM on our dataset, where the MSE loss function is minimized as well. Overall, compared with RNN-BiLSTM, ABBIE achieves superior performance. In particular, ABBIE achieves an increase in the accuracy value of about 2.3% on equal terms of data and experimental protocol.

To study the effect of adopting different loss functions, we train our model, ABBIE, on our dataset by using Mean Squared Error (MSE) and Binary Cross-Entropy (BCE) functions. As shown in Table 2, when using BCE, ABBIE achieves an accuracy value of 87.7%. Instead, when using MSE, ABBIE achieves an accuracy value of 89.27%. Overall, for our problem, an error-based loss function seems to be more effective.

Figs. 4 and 5 help to visualize how attention values are distributed across different characters in a (padded) compound word. For each character of the word (horizontal axes), the figures show the 128-dimensional attention vector obtained from the Multi-Head attention layer by Eq. 13. Under each figure the True split window and the predicted split window are also reported. The red dashed vertical line represents the True split location, while the green vertical line represents the predicted one.

In Fig. 4, for the compound word "bhArasADhanam", the predicted split position (green dashed line) exactly corresponds to the true split position (red dashed line), which is between "bhAra" and "sAdhanam". This means the model has learned to focus its attention on the correct characters within the sequence. A subset of lines, especially in the middle of the figure, shows focused attention close to the true split position of higher intensity (blue or red regions) around the respective positions. The presence of more lines with high intensity at positions around the split gives a strong and clear signal about the position, reducing the ambiguity and hence ensuring correctness. This behavior indicates that the model learns to identify key features indicating the presence of a split position. On the other hand, there is some redundancy since several elements of the attention vectors are observed to contribute to the same prediction, likely adding to robustness.

In Fig. 5, for the compound word "kalApaVAtAH", the predicted split locations are far from the true split locations that are between "kalApa" and "VAtAH". This is clear from the misalignment of the green and red dashed lines. The attention heads do

Table 1: Comparison with the state-of-the-art on the dataset (Dave et al., 2021).

Model	Accuracy
RNN-BiLSTM (Dave et al., 2021)	92.3%
ABBIE (ours)	93.1%

Table 2: Comparison with the state-of-the-art on our dataset.

Model	Accuracy
RNN-BiLSTM (Dave et al., 2021)	87.0%
ABBIE - MSE (ours)	89.27%
ABBIE - BCE (ours)	87.7%

not pay proper attention to the true split locations, either diffused or misaligned, which drives the model to attend to other positions. Some elements of the attention vector appear to focus on higher intensities around the position of a true split but do not give sharp attention. Attention is diffused across neighboring positions. Some lines show a broader attention to irrelevant positions, which might result in the incorrect split position prediction. The true split window clearly identifies the start of the correct split position (1 at the true split) and the predicted split window deviates significantly, with earlier activations (1s) that align with the incorrect split.

These heatmaps indicate that not all attention features contribute the same in predicting the split window. Some features show sharper focus at critical positions, whereas others distribute their attention over the sequence. This also aligns with the behavior of a multi-head attention mechanism that allows to learn aspects complementary to one another when attending to the input.

6.4 Ablation Studies

To better understand the contribution of each component to the ABBIE model, we trained and compared different variants of our model by using the MSE function and on equal terms of experimental protocol. Results are summarized in Table 3 and discussed in details in the following.

To assess the contribution of the multi-head attention module, we trained the *"Two Bi-Encoders"* model. In practice, this model concatenates the out-

Table 3: Ablation Study. The table shows the contribution of the multi-head attention module and of the two bi-encoders to the model accuracy.

Model	Accuracy
Two Bi-Encoders	86.5%
One Bi-Encoder + Attention	87.5%
ABBIE (ours)	89.27%

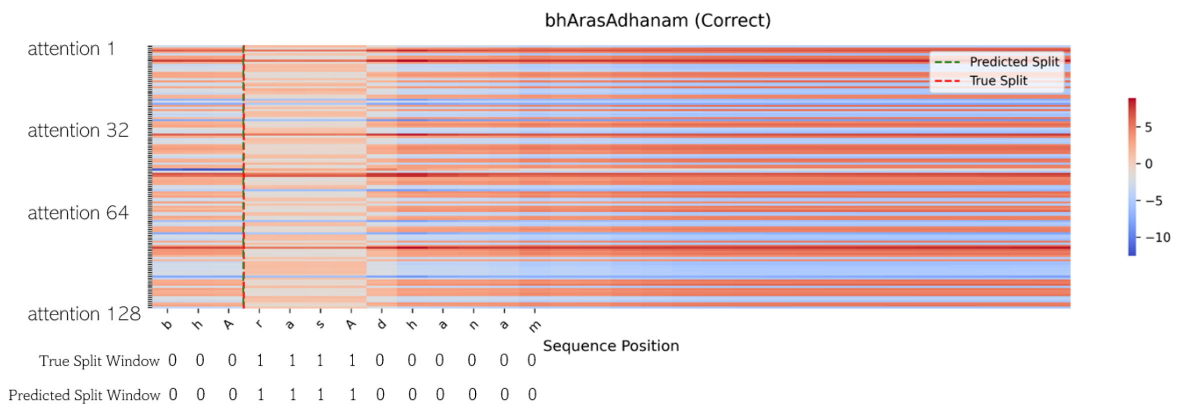

Figure 4: Heatmap for Correct Split Window Prediction.

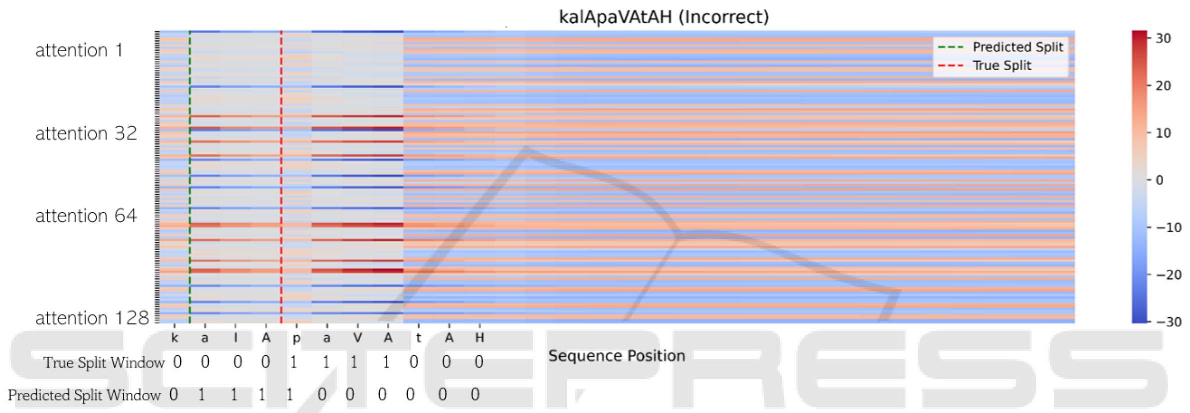

Figure 5: Heatmap for Incorrect Split Window Prediction.

puts of the direct and reverse bi-LSTMs and uses a dense layer to output the target sequence. No attention mechanism is used in this model. The modified model achieves an accuracy of 86.5% showing a clear reduction of 2.8% compared to the ABBIE model. The results are also comparable to the one reached by the RNN-BiLSTM model, which does not include any attention mechanism as well. This result suggests that multi-head attention plays a crucial role in capturing the most relevant information in the compound word, thereby enhancing the accuracy of the split location prediction.

To assess the contribution of using direct and reverse Bi-Encoders, we trained the *"One Bi-Encoder + Attention"* model. This model only includes a single Bi-directional LSTM. The output of the direct Bi-Encoder feeds the multi-head attention module. The model achieves an accuracy of 87.5%, showing a reduction of 1.8% compared to ABBIE. The accuracy value is slightly superior to the one reached by the model in (Dave et al., 2021). The result also confirms the findings of (Sutskever et al., 2014), which can be summarized here by stating that the direct and reverse Encoders capture different contextual information. Moreover, the results highlight the importance of multi-head attention in comparing the direct and reverse dynamics of the input sequence to predict the Sandhi split location.

7 CONCLUSION AND FUTURE WORK

In this research work, we focus on the problem of predicting where to split compound Sanskrit words whose formation follows the Sandhi rule without any use of external resources such as phonetic or morphological analyzers. To address the problem, we propose a novel deep learning method that uses two Bi-LSTMs to encode character-level contextual information for direct and reverse sequence. The two bi-encoders are used with a multi-head attention module to focus on the different parts of the compound word and predict where the split arises. We also collected an enhanced dataset that augment the UoH dataset with challenging compound words from the Digital Corpus of Sanskrit repository. We compared the performance of our proposed ABBIE model with other RNN based architecture.

Overall, this paper shows that bidirectional encoders on the direct and reverse input sequences can be used together with an attention-based module to get better contextual information on the input sequence.

The main limitation of our work, as well as of all other works on the topic, is that it considers compound words having only one Sandhi split. Despite we could use our model recursively until no split is found anymore, in future work, we intend to enhance our model to also consider the splitting of compound words which have more than one valid split locations integrate with the morphological analyzer and observe the common error patterns.

ACKNOWLEDGMENTS

This work is partially supported by the PNRR MUR Project ITSERR (Italian Strengthening of the ESFRI RI RESILIENCE) (IR0000014, CUP B53C22001770006).

REFERENCES

Aralikatte, R., Gantayat, N., Panwar, N., Sankaran, A., and Mani, S. (2018). Sanskrit sandhi splitting using seq2 (seq)ˆ 2. *arXiv preprint arXiv:1801.00428*.

Cai, D. and Zhao, H. (2016). Neural word segmentation learning for chinese. *arXiv preprint arXiv:1606.04300*.

Chen, X., Qiu, X., Zhu, C., Liu, P., and Huang, X.-J. (2015). Long short-term memory neural networks for chinese word segmentation. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1197–1206.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12:2493–2537.

Dave, S., Singh, A. K., AP, D. P., and Lall, P. B. (2021). Neural compound-word (sandhi) generation and splitting in sanskrit language. In *Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD)*, pages 171–177.

Gong, J., Chen, X., Gui, T., and Qiu, X. (2019). Switch-lstms for multi-criteria chinese word segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6457–6464.

Goyal, P., Arora, V., and Behera, L. (2007). Analysis of sanskrit text: Parsing and semantic relations. In *International Sanskrit Computational Linguistics Symposium*, pages 200–218. Springer.

Goyal, P. and Huet, G. (2013). Completeness analysis of a sanskrit reader. In *International Symposium on Sanskrit Computational Linguistics*, pages 130–171.

Goyal, P., Huet, G., Kulkarni, A., Scharf, P., and Bunker, R. (2012). A distributed platform for sanskrit processing. In *Proceedings of COLING 2012*, pages 1011–1028.

Haruechaiyasak, C., Kongyoung, S., and Dailey, M. (2008). A comparative study on thai word segmentation approaches. In *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, volume 1, pages 125–128. IEEE.

Hellwig, O. (2010–2021). *The Digital Corpus of Sanskrit (DCS)*.

Hellwig, O. (2015). Using recurrent neural networks for joint compound splitting and sandhi resolution in sanskrit. In *4th Biennial workshop on less-resourced languages*.

Hellwig, O. and Nehrdich, S. (2018). Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2754–2763.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Huet, G. (2003). Towards computational processing of sanskrit. In *International Conference on Natural Language Processing (ICON)*. CiteSeer.

Huet, G. (2005). A functional toolkit for morphological and phonological processing, application to a sanskrit tagger. *Journal of Functional Programming*, 15(4):573–614.

Krishna, A., Santra, B., Satuluri, P., Bandaru, S. P., Faldu, B., Singh, Y., and Goyal, P. (2016). Word segmentation in sanskrit using path constrained random walks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 494–504.

Kulkarni, A. and Shukl, D. (2009). Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70(1-4):169–177.

Kumar, A., Mittal, V., and Kulkarni, A. (2010). Sanskrit compound processor. In *Sanskrit Computational Linguistics: 4th International Symposium*, pages 57–69, New Delhi, India. Springer.

Kumar, S. (2007). Sandhi splitter and analyzer for sanskrit (with reference to ac sandhi). Submitted, 2007.

Learn Sanskrit (2024). Sanscript: Sanskrit transliteration. Accessed: 2024-08-08.

Linguistics, S. C. (2002-24). Sanskrit computational linguistics.

Mittal, V. (2010). Automatic sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 85–90.

Natarajan, A. and Charniak, E. (2011). s3-statistical sandhi splitting. In *Proceedings of 5th international joint conference on natural language processing*, pages 301–308.

Patankar, M. P. S. (2023). Exploring the intricacies of sandhi in sanskrit: phonological rules and linguistic significance. *International Journal of Applied Engineering & Technology*.

Pei, W., Ge, T., and Chang, B. (2014). Max-margin tensor neural network for chinese word segmentation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 293–303.

Reddy, V., Krishna, A., Sharma, V. D., Gupta, P., Goyal, P., et al. (2018). Building a word segmenter for sanskrit overnight. *arXiv preprint arXiv:1802.06185*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

UoH (2024). Sanskrit corpus.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Yao, Y. and Huang, Z. (2016). Bi-directional lstm recurrent neural network for chinese word segmentation. In *Neural Information Processing: 23rd International Conference, ICONIP 2016, Kyoto, Japan, October 16–21, 2016, Proceedings, Part IV 23*, pages 345–353. Springer.

Zheng, X., Chen, H., and Xu, T. (2013). Deep learning for chinese word segmentation and pos tagging. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 647–657.