Phonological Assimilation–Aware Neural Segmentation of Sanskrit Compounds

Sushant Dave
IIT Hyderabad
sushant.dave@gmail.com

Ramanan Sivasubramanian
IIT Hyderabad
ht25resch14001@iith.ac.in

Amba Kulkarni
University of Hyderabad
ambakulkarni@uohyd.ac.in

Ramakrishna Upadrasta
IIT Hyderabad
ramakrishna@cse.iith.ac.in

Abstract

Compound words (*samāsa*) are pervasive in Sanskrit and pose a major challenge for computational processing due to recursive structure, phonological assimilation (sandhi), and the absence of explicit boundary markers. While recent transformer-based models achieve strong performance on sentence-level Sanskrit segmentation, they require substantial computational resources and large training corpora, making them less suitable for lightweight or resource-constrained pipelines. In this work, we revisit Sanskrit compound segmentation as a *word-level, context-agnostic* problem and propose a two-phase neural approach that decouples approximate boundary detection from phonological assimilation reversal. The first phase predicts approximate component boundaries using a lightweight BiLSTM sequence labeling model, while the second phase performs sandhi reversal using a localized Seq2Seq model operating on a small character window. We derive a dataset of 86,518 compound words from an existing alignment corpus and train models with a total of approximately 0.55M trainable parameters. Despite their small size, our models achieve competitive accuracy on isolated compound words and outperform large transformer baselines under identical word-level, context-free evaluation conditions. The proposed approach offers an efficient and linguistically motivated alternative for Sanskrit compound analysis and can serve as a practical preprocessing component for downstream NLP tasks.

1 Introduction

Word compounding or *samāsa*[1] is the process of putting together two or more words to express a composite sense. Word compounding is mainly used to bring brevity in expression and is heavily employed in languages, both in literature and common usage. For example, the English word 'schoolboy' is a compound word that refers to a school-going boy. Although not explicitly stated, the compound word manages to convey the meaning, succinctly and precisely.

Almost all Indian languages, including Sanskrit, make extensive use of compound words. Kulkarni et al. (2012) have shown that the insights gained from Sanskrit grammar can be applied to the analysis of compounds in Hindi and Marathi. While the exact formation-rules and extent of usage of compounds may differ across languages, the idea of compound words (or, simply compounds) is present in most languages.

[1] All Sanskrit terms in the paper have been represented using the IAST Roman transliteration standard

1.1 Compound words in Sanskrit

In Sanskrit, compound words are employed as a linguistic tool that makes the language terse and powerful. A Sanskrit compound word is typically made up of two or more nouns (henceforth referred to as components in this paper). For example, the compound word *nīlotpala*, meaning blue-lotus, consists of two components, *nīla* and *utpala*. But components cannot be joined indiscriminately to form compounds. In Sanskrit, the creation of compound words is highly structured and is governed by specific rules. These rules dictate which words can be combined to create a compound, along with how the resultant compound will be interpreted. When correctly created, the compound exhibits certain characteristics. Kulkarni and Kumar (2013) have published a study on the rules that provide conditions for compound formation. This study analyzes these conditions from the point of view of compound-type identification. At a high level, these characteristics are as follows:

- Syntactically, a compound contains (any number of) components, but typically, the last one reflects the inflected form of the compound while the other components retain their non-inflected forms.
- Semantically, the components are connected and have the ability to refer to a single entity.
- All component boundaries within a compound are governed by the *sandhi* rules of Sanskrit grammar. *Sandhi* or the fusion of sounds across word boundaries, is used to facilitate pronunciation and needs to be addressed during compound analysis. The *sandhi* rules in Sanskrit are well defined and precise. These rules explain which phonemes can be merged under what conditions and what will be the resultant phoneme.

  As per *Śabdakaustubha*, a work by *Bhaṭṭojīdīkṣita* (17th century grammarian), *sandhi* is an optional transformation under certain conditions, but for the components within a compound word, *sandhi* is mandatory at the component boundary i.e., if the *sandhi* rules allow the phonetic merge of component words, the merge will take place in accordance with the rules of *sandhi*.

1.2 Sanskrit compound analysis

Compound analysis is critical for any Sanskrit NLP application. A good understanding of compounds and their associated semantic/syntactic analysis is important for decoding and understanding any Sanskrit text due to the abundance of compounds in them.

Anil Kumar (2012) proposed the following sequence of steps for Sanskrit compound word analysis:

1. Segmentation: the process of marking the segment/component boundaries in a compound word.
2. Constituency Parsing: The intermediate stage where possible semantic interconnectedness between the segments is derived. Between the multiple options possible, the best option is chosen. Kulkarni and Kumar (2011) introduce a constituency parser for Sanskrit compounds.
3. Type Identification: Deriving relationship between components using semantic analysis.
4. Paraphrase Generation: Applying predefined templates based on compound types identified above. Kumar et al. (2009) propose a high accuracy paraphrase generator.

Many NLP tasks are accomplished by statistical analysis of words. Multiple research works like Chen et al. (1999) and Fearn et al. (2021) have observed and analyzed how the decrease in dictionary size leads to improvements in efficacy of statistical and learning-based NLP methods. It is therefore desirable to have a method to analyze and reverse the compounding in Sanskrit to improve the performance of downstream NLP tasks.

1.3 Challenges in Sanskrit Compound Analysis

Sanskrit compound analysis presents a unique combination of linguistic and computational challenges that distinguish it from compound processing in many other languages. These challenges directly inform the modeling decisions adopted in this work.

Unbounded Recursive Structure Sanskrit compounds have no theoretical upper bound on the number of components they may contain. A compound can recursively embed other compounds, resulting in deeply nested structures. Consequently, compound segmentation is inherently ambiguous, with multiple possible decompositions, only one of which is linguistically valid. It has been pointed out by Huet (2009), that the number of ways the constituents of a compound with n+1 characters can be grouped is the n^{th} Catalan number. Catalan numbers grow exponentially, making the exhaustive enumeration of all possibilities computationally infeasible for long compounds.

Fuzzy Component Boundaries due to Sandhi Component boundaries within a compound are often obscured by mandatory phonological assimilation (sandhi). Depending on the phonemes involved, sandhi may leave boundaries unchanged or may significantly alter surface forms, leading to ambiguity in boundary localization. As a result, precise boundary detection and sandhi reversal cannot always be performed simultaneously using a single global decision.

Locality of Phonological Transformations Although compound structure can be deeply recursive, sandhi operations themselves are highly local, typically affecting only a small number of characters around a component boundary. This locality suggests that sandhi reversal can be treated as a focused subproblem once an approximate boundary region has been identified.

Morphological and Semantic Underspecification When a compound is segmented, the resulting components may differ in gender, number, or case from the surface form of the compound. Furthermore, semantic relations between components are not explicitly encoded and must be inferred in later stages of analysis. These issues complicate downstream processing but are orthogonal to boundary detection itself.

Implications for Modeling Taken together, these challenges suggest that compound segmentation benefits from a staged approach: an initial model that identifies approximate boundary regions under ambiguity, followed by a localized mechanism that resolves sandhi effects. This observation motivates the two-phase architecture proposed in this work, which explicitly separates boundary detection from phonological assimilation reversal.

Rather than proposing a universally optimal segmentation model, this work aims to characterize a practical and linguistically grounded solution for word-level Sanskrit compound analysis under strict resource and context constraints.

2 Motivation and Contributions

Most prior work on Sanskrit compound segmentation relies either on handcrafted linguistic resources or on large neural models trained on sentence-level data. While sentence-level models benefit from contextual cues, many real-world Sanskrit NLP pipelines—such as lexicon building, morphological analysis, and dictionary indexing—require reliable segmentation of isolated compound words without access to surrounding context.

Recent transformer-based approaches (Sandhan et al., 2022; Nehrdich et al., 2024) demonstrate strong performance but involve hundreds of millions of parameters and substantial training cost. This limits their applicability in low-resource settings, rapid experimentation, and deployment scenarios where computational efficiency is critical.

Motivated by these constraints, we explore whether a linguistically informed decomposition of the task—separating boundary detection from sandhi reversal—can yield competitive performance using significantly smaller models. Our approach does not rely on external lexical resources and is trained entirely from data.

The contributions of this work are:

- We formulate Sanskrit compound segmentation as a word-level, context-agnostic task and empirically evaluate it under this constraint.
- We present a derived dataset of 86,518 Sanskrit compound words obtained from an existing alignment corpus.
- We propose a two-phase neural architecture that decouples boundary detection and sandhi reversal, achieving a strong accuracy–efficiency trade-off with approximately 0.55M trainable parameters.
- We provide an empirical comparison with transformer-based models under identical word-level evaluation conditions, highlighting scenarios where lightweight models are preferable.

The dataset and the code are being made public for other researchers to benefit from[2].

3 Relevant Works

Many researchers have tried different methods for Sanskrit compound identification and split. Among the Finite-State methods, Huet (2005) proposed Finite State Transducer as decorated lexical trees for marking of component boundary and compound segmentation. Mittal (2010) used a dual approach, employing a Finite State Transducer augmented by *sandhi* rules along with generating all possible segmentations and validating the resulting components using a dictionary.

Kumar et al. (2010) used the Optimality Theory proposed by Prince (2004) for segmentation. Their work selected the best candidate among all possible segmentation possibilities, by looking at the statistical distribution of the different types of *sandhi* used in a corpus of approximately 25K words.

Natarajan and Charnaik (2011) used a method similar to the one proposed by Kumar et al. (2010). Their work used different probabilistic models which claimed to have better results than earlier probabilistic approaches. In addition, they presented a segmentation method based on Bayesian word segmentation proposed by Goldwater et al. (2006). Krishna et al. (2016) treated the segmentation problem as a query expansion problem. An input sentence is analyzed for all possible segmentations and morphological information for each of the segments is extracted using the Sanskrit Heritage Reader[3]. All possible segments, are treated as nodes in a graph and a path-constrained random walk is performed to get the most likely candidate.

Machine learning based methods also have become increasingly popular for compound word segmentation. Hellwig (2015) performed segmentation by applying bidirectional LSTM to two parallel character based representations of a string. Hellwig and Nehrdich (2018) proposed a mixture of Recurrent and Convolutional Neural Networks for compound word split. Their segmentation models work on sentences and are language agnostic. Krishna et al. (2018) applied an energy based model to jointly perform word segmentation and morphological analysis. Reddy et al. (2018) applied Deep Seq2Seq RNN model for segmentation. Their method performs the segmentation task on sentence level. Aralikatte et al. (2018) proposed RNN based Seq2Seq models with 2 stage decoding, first to identify the component boundaries, and a second one to do the segmentation. The Seq2Seq model used attention mechanism to achieve high accuracy. Dave et al. (2021) also used a double decoder method, but their method focused on finding a split window instead of a split point. The split window is the sequence of maximum 5 characters over which segmentation is performed. Singh et al. (2022) used Seq2Seq model for segmentation of suffix based inflectional words in Sanskrit.

[2]https://github.com/SushantDave/Samasa_Prakarana
[3]http://sanskrit.inria.fr/DICO/reader.fr.html

Work	Input Level	Context	Lexicon	Method
Huet (2005)	Word	No	Yes	Finite-state transducers
Mittal (2010)	Word	No	Yes	FST + dictionary validation
Kumar et al. (2010)	Word	No	Partial	Optimality-theoretic model
Krishna et al. (2016)	Sentence	Yes	Yes	Graph-based random walks
Hellwig (2015)	Sentence	Yes	No	BiLSTM
Hellwig & Nehrdich (2018)	Sentence	Yes	No	CNN + RNN
Aralikatte et al. (2018)	Sentence	Yes	No	Seq2Seq with attention
Dave et al. (2021)	Word	No	No	Window-based RNN
Sandhan et al. (2022)	Sentence	Yes	No	Transformer (TransLIST)
Nehrdich et al. (2024)	Sentence	Yes	No	Transformer (ByT5-Sanskrit)
This work	**Word**	**No**	**No**	**Two-phase BiLSTM + Seq2Seq**

Table 1: Comparison of Sanskrit compound segmentation approaches.

In recent times, some authors have used Transformer models, originally proposed by Vaswani et al. (2017). Sandhan et al. (2022) and Nehrdich et al. (2024) used character-level encoding with Transformer models. These Transformer-based models show better performance than the other methods preceding them.

As shown in Table 1, most recent neural approaches operate at the sentence level and implicitly rely on contextual information. In contrast, our work focuses on word-level, context-agnostic compound segmentation and explicitly addresses phonological ambiguity through a staged architecture. This positioning enables a direct investigation of the trade-offs between model complexity and segmentation accuracy under strict input constraints.

4 A two-phase approach to segmentation

Our proposed method works at the word level, and is completely context-agnostic, making no assumptions about the context in which the compound word might have been used. It takes a given compound word and predicts its components in two sequential and distinct phases. Both phases employ different Recurrent Neural Network models; and each is trained in isolation from the other model.

- **Phase 1**: Find the approximate component boundaries in compound. As mentioned earlier in Section 1.3, *sandhi* may cause the component boundaries to be altered, and this makes it difficult to mark them precisely.
- **Phase 2**: Perform *sandhi* split at the approximate component boundaries found in Phase 1.

Details about the dataset creation and model training procedure are described in Sections 4.1 and 4.2, respectively.

4.1 Dataset Creation

To derive the dataset for Phase 1, we take the data introduced by Krishnan et al. (2023), referred to as the *Alignment Dataset*[4] by the author. The Alignment Dataset contains pairs of sentences where the first sentence in the pair contains a set of words, including compounds, and the second sentence in the pair contains the same content but with the phonetic merge reversed; i.e., with compounds split, and with a hyphen at the component boundary.

An example of a sentence pair from Alignment Dataset is as below:

Sentence-1: *nāsti māyāsamaḥ pāśo nāsti yogāt param balam*

Sentence-2: *na asti māyā-samaḥ pāśaḥ na asti yogāt param balam*

4.1.1 Phase1

The procedure to extract the data from DCS corpus, in order to train the Phase 1 model is as follows:

[4]Alignment Corpus. `https://github.com/samsaadhanii/datasets/tree/main/dcs_sh_alignment`

Algorithm 1 Data extraction from Alignment Dataset

$S_1 \leftarrow$ getDataset() ▷ Read alignment dataset in a set
$S_2 \leftarrow$ convertToSLP(S_1) ▷ Convert dataset to SLP1
$S_3 \leftarrow \emptyset$ ▷ Declare an empty set
for all $(s1, s2) \in S_2$ **do** ▷ Read sentence pair (s1, s2) in set
    for all $word2 \in s2$ **do**
        if hyphen present in the word **then**
            $mindist \leftarrow INT\_MAX$
            $minword \leftarrow$ "" ▷ Declare a variable and initialize to zero
            for all $word1 \in s1$ **do**
                $dist \leftarrow$ LevDist($word1, word2$) ▷ Compute Levenshtein Distance
                if dist $<$ mindist **then**
                    $mindist \leftarrow dist$
                    $minword \leftarrow word1$
                end if
            end for
            $S_3 \leftarrow S_3 \cup \{(minword, word2)\}$ ▷ Insert word pair into set
        end if
    end for
end for
▷ Handle cases where sandhi occurs without compounding
$S_4 \leftarrow \emptyset$
for all $(word1, word2) \in S_3$ **do**
    $min\_dist \leftarrow INT\_MAX$
    $min\_start\_idx \leftarrow 0$
    $min\_word\_len \leftarrow 0$
    for $start\_id \leftarrow 0$ to $|word1| - 1$ **do**
        for $window\_len \leftarrow 1$ to $|word1| - start\_id$ **do**
            $dist \leftarrow$ LevDist($word2, word1[start\_id : start\_id + window\_len]$)
            if $dist \leq min\_dist$ **then**
                $min\_dist \leftarrow dist$
                $min\_start\_idx \leftarrow start\_id$
                $min\_word\_len \leftarrow window\_len$
            end if
        end for
    end for
    $newword \leftarrow word1[min\_start\_idx : min\_start\_idx + min\_word\_len]$
    $S_4 \leftarrow S_4 \cup \{(newword, word2)\}$ ▷ Insert word pair into set
end for

Algorithm 2 Create an indicator string for every compound

function FIND_MAPPING_LEN($word2, word1, start\_id$)
    $min\_dist \leftarrow INT\_MAX$
    $min\_word\_len \leftarrow 0$
    for $curr\_id \leftarrow start\_id + 1$ to $|word1| - 1$ **do**
        $dist \leftarrow$ LevDist($word2, word1[start\_id : curr\_id]$)
        if $dist \leq min\_dist$ **then**
            $min\_dist \leftarrow dist$
            $min\_word\_len \leftarrow curr\_id - start\_id$
        end if
    end for
    return min_word_len
end function
$S_5 \leftarrow \emptyset$
for all $(word1, word2) \in S_4$ **do**
    $wlist \leftarrow$ split($word_2, '-'$)
    $start\_id \leftarrow 0$
    for all $word \in wlist$ **do**
        $wlen \leftarrow$ find_mapping_len($word, word1, start\_id$)
        $IndicatorString \leftarrow$ String of ($wlen$ - 1) 0s
        $start\_id \leftarrow start\_id + wlen$
    end for
    $S_5 \leftarrow S_5 \cup \{(word_1, word_2, IndicatorString)\}$
end for

1. Convert the *Alignment Dataset* to SLP1 format.[5]

2. Select only the hyphenated words from second sentence in the sentence pair.

3. Find corresponding word in the first sentence by using the Levenshtein distance, proposed by Vladimir Levenshtein (1966). For each hyphenated compound selected in previous step, select the word in the first sentence that has the smallest Levenshtein distance to it, thus creating a set of pairs. In the example sentences above, only the segmented word *māyā-samaḥ* is chosen and the word *māyāsamaḥ* from the first sentence is extracted to create the pair of words.

4. Final step is to create Indicator string for every compound. This is a string of the same size as the compound word, containing zeros (0) at all character positions, except the positions where the new word boundary is expected, at which location, it will contain a one (1). For example, if the compound is *deśakālādhvayugam*, the indicator string for it is '00001000100010000'. Notice that the presence of 1 in the indicator string marks the start of a new component. As mentioned earlier, this marker is approximate, and can be off by one position to left or right.

4.1.2 Notes

1. It is important to note that sometimes *sandhi* can occur even when word compounding does not take place, and therefore, in such scenarios we had to leave out the part of the word that is not a part of the compound. For example, *kaṣāyastiktamadhuraḥ* in sentence-1 will become *kaṣāyaḥ tikta-madhuraḥ* in sentence-2 where *kaṣāyaḥ* is a separate word and *tikta-madhuraḥ* is the compound. So in this case, from sentence-1 we extract only *tiktamadhuraḥ*

[5]The Sanskrit Library Phonetic basic encoding scheme (SLP1) is an ASCII transliteration scheme for the Sanskrit language from and to the Devanagari script. `https://en.wikipedia.org/wiki/SLP1`

Compound Word	Segmented Equivalent	Indicator String
triveṇum	*tri-veṇum*	00010000
mahotpātāḥ	*mahā-utpātāḥ*	0000100000
trijātadhānyamaricam	*tri-jāta-dhānya-maricam*	0001000100001000000
nityāvadhānena	*nitya-avadhānena*	0000010000000

Table 2: Final Dataset. Column 1 shows the original word picked from the alignment dataset. Column 2 shows the same word with its components separated by a hyphen('-'). The third column contains the indicator string which marks where the component boundaries exist in the compound word. The indicator string is mapped on to the compound word in first column.

Our method enables this by ignoring the right amount of characters of the word in sentence-1 by using Levenshtein distance to find the smallest sub-string in the word that matches the compound in sentence-2. For the example mentioned in the above step, we will be left with *tiktamadhuraḥ* as the compound and *tikta-madhuraḥ* as its segmented form.

2. Possible inconsistencies, if and when found, were detected and corrected through a program that analyzed the characters around the hyphen. After this point, we are left with a set of word pairs. The first word in each pair is a compound word and the second word is the same compound with its component boundaries marked with a hyphen ('-') as shown in first and second column of Table 2.

   The detailed algorithm for extracting compounds and corresponding segmented equivalents is described in Algorithm 1. The algorithm for indicator string generation is described in Algorithm 2. Compounds and their equivalent indicator strings are given in Columns 1 & 3 of Table 2; these constitute our dataset for Phase 1, as input and output respectively for the training. Using the above steps, we created a database of 86,518 words.

4.1.3 Phase2

For Phase 2, we curate a database of characters around the component boundary and train a model that learns to detect and if needed, reverse the changes (caused due to *sandhi*) around the component boundary. Dave et al. (2021) have shown that high accuracy *sandhi* split can be achieved by analyzing a small window of characters containing the *sandhi* split point, which the authors refer to as Sandhi-Window. This Sandhi-Window is between 3-5 characters. We take a similar approach here and use a window of 4-6 characters, that our model tries to analyze and split.

To create this dataset, we start with the word pairs in Columns 1 & 2 of Table 2. For each word pair in the dataset, 3 characters before and 3 characters after the component boundary were extracted as given in Table 3. In case there are only 2 characters before or after the boundary, 2 characters are picked. The cases with one character before or after component boundary are ignored. This gives us a Sandhi-Window of 4-6 characters.

The algorithm for data curation of Phase 2 is described in Algorithm 3. Using this method, a total of 128,538 data samples were generated for training the Phase 2 model. This is larger than the dataset for Phase 1, since there are multiple components in some compounds.

4.2 Procedure

For Phase 1, we train a model that would generate an indicator string for a given input compound. For Phase 2, we train a model that splits the components of the compound by splitting the characters around the component boundary, predicted by the indicator string. This flow is illustrated in Figure 1 for an example compound *deśakālādhvayugam*.

The model training details are described in the following sections.

Algorithm 3 Create Phase 2 Dataset

function FIND_FIRST_NON_MATCH_INDEX($word1, word2$)
    $idx \leftarrow 0$
    while $word1[idx] = word2[idx]$ **do**
        $idx \leftarrow idx + 1$
    end while
    $word1 \leftarrow \text{Reverse}(word1)$
    $word2 \leftarrow \text{Reverse}(word2)$
    $revidx \leftarrow 0$
    while $word1[revidx] = word2[revidx]$ **do**
        $revidx \leftarrow revidx + 1$
    end while
    $revidx \leftarrow |word1| - revidx$
    return $(idx, revidx)$
end function
$newlist \leftarrow \emptyset$
for all $(word1, word2) \in S_4$ **do**
    $(idx, revidx) \leftarrow$ FIND_FIRST_NON_MATCH_INDEX($word1, word2$)
    for $L = 2$ to 3 **do**
        for $R = 2$ to 3 **do**
            if $[idx - L : revidx + R]$ is within bounds of $word1$ and $word2$ **then**
                $newword1 \leftarrow word1[idx - L : revidx + R]$
                $newword2 \leftarrow word2[idx - L : revidx + R]$
                $newlist \leftarrow newlist \cup \{(newword1, newword2)\}$
            end if
        end for
    end for
end for

Input for training	Output for training
triveṇ	*tri-veṇ*
mahotp	*mahā-utp*
trijāt, ātadhā, nyamar	*tri-jāt, āta-dhā, nya-mar*
tyāvad	*tya-avad*

Table 3: Dataset for training Phase 2. This table shows the same examples as shown in Table 2. The first column shows the characters of the Sandhi-Window picked from the compound. Column 2 shows the corresponding character sequence with component boundary marked with a hyphen('-').

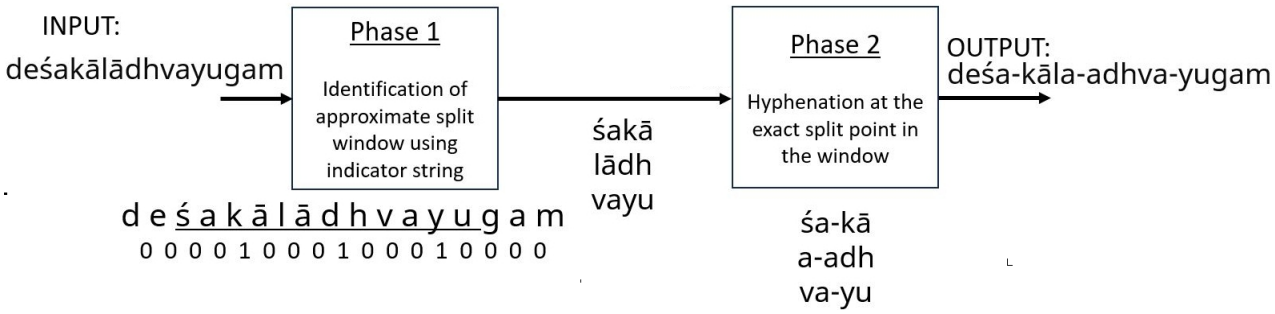

Figure 1: Working of Phase 1 and Phase 2 in tandem

4.2.1 Phase 1

Using the dataset curated for Phase 1, the process for which is described in the previous section, we train a Recurrent Neural Network to generate the Indicator string from the input string. The input to this RNN is a character from the input compound at each time step. The characters are passed sequentially, from left to right. The output for this RNN at each time step, is the number 0 or 1 corresponding to the input character. Thus for each character, RNN is expected to generate a 0 or 1 value which are then put together to create the indicator string.

For training the Phase 1 model, the input characters are mapped to numbers, which are then converted to one-hot vectors. The input sequence is passed to the RNN where the basic unit is a Bidirectional LSTM. A Bidirectional LSTM provides better accuracy compared to the standard LSTM cell. The code is written in Python using Keras library. From the total dataset of 86,518 samples, 17,304 samples (20% of the dataset) are retained for test and the remaining 69,214 samples are used for training. Training batch consists of 64 samples and the training is done for 200 epochs. RMSprop (Root Mean Squared Propagation) optimization is used for training with dropout regularization to minimize bias. The architecture for Phase 1 model is laid out in Figure 2

4.2.2 Phase 2

For the Phase 2, we use a Seq2Seq model introduced by Sutskever et al. (2014). The characters are mapped to numbers, which are converted to one-hot vectors. Characters '&' and '$' are used as start and end markers respectively in the output sequence. These one-hot vectors serve as the input to the encoder model. Once the sequence is finished, the final vector output of last stage is passed to the decoder stage. This model performs the task of reading the characters around the component boundary and return the character sequence with phonetic merge undone and a hyphen ('-') inserted to mark the component boundary.

Bi-LSTM cell is used as the basic unit for encoder and standard LSTM cell is used as basic unit for the decoder. For both, encoder and decoder, the hidden unit size is set as 128. Of the 128,538 samples in the dataset, 25,708 (20% of the dataset) are randomly selected and

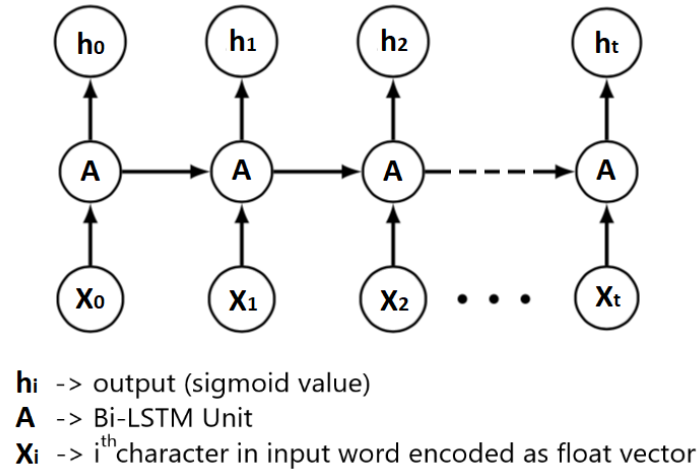

Figure 2: Architecture for Phase 1 model

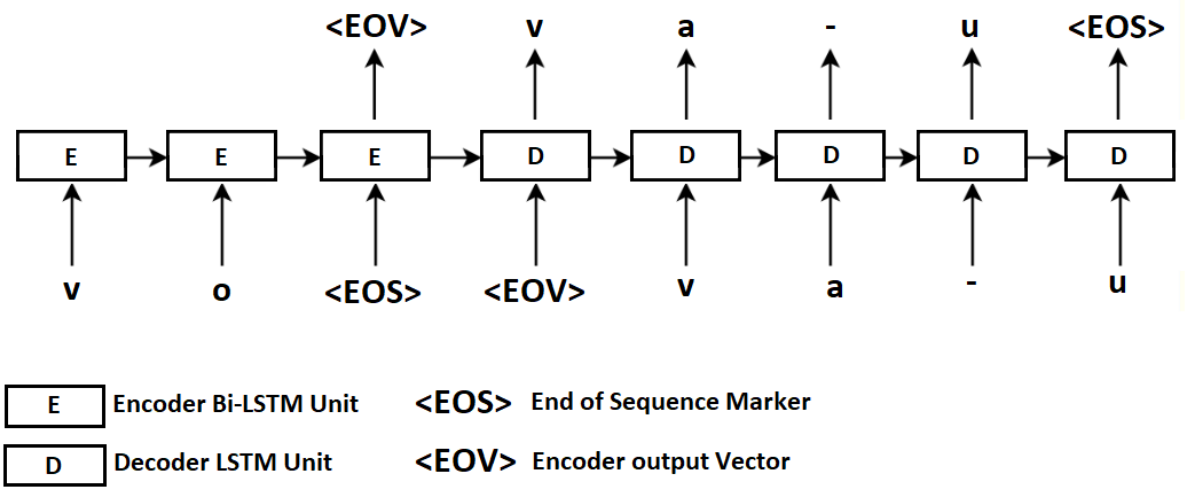

Figure 3: Architecture for Step 2 model that decodes the sequence 'vo' to 'va-u'

set aside for testing. Remaining 102,840 samples are used for training the model. Training is done for 30 epochs with a batch size of 64. Model is trained with RMSProp optimizer and categorical-cross-entropy error loss. The architecture for Phase 2 model is laid out in Figure 3

4.2.3 Final System (Phase 1 and Phase 2)

Once the Phase 1 and Phase 2 models were trained, the inference was done by these 2 models sequentially. Phase 1 marks the component word boundary by generating the indicator string and Phase 2 operates at the word window indicated by the indicator string to split the compound into components and ensuring the boundary characters of these components are restored by applying the Seq2Seq model of Phase 2. A few examples below show the flow:

5 Evaluation and Results

The training of both Phase 1 and Phase 2 models was done using four NVIDIA A100 (80GB) GPUs. The combined training time of both models is approximately 180 minutes. Phase 1 model has 37,897 trainable parameters and Phase 2 model has 5,17,429 trainable parameters. The inference was done on the same hardware that was used for training. The average inference times per test sample for the Phase 1 and Phase 2 models are 180 milliseconds and 570

Phase 1 Input	Phase 1 Output	Word Window
mandāgnim	000001000	*ndāgn*
śuddhātmanaḥ	00000100000	*dhātma*
pitṛtarpaṇam	000010000000	*tṛtar*

Table 4: Phase 1 results. The first column shows the input compound passed to Phase 1 model. The output of the model is shown in column 2. This is a sequence of '0' and '1' for every character in the input word, where 1 marks the start of a new component, thus indicating the component boundary. Column 3 shows the character sequence of 4-6 characters around the component boundary. This forms the input to the Phase 2 model.

Phase 2 Input	Phase 2 Output
ndāgn	*nda-agn*
dhātma	*dha-ātma*
tṛtar	*tṛ-tar*

Table 5: Phase 2 results. Column 1 here is mapped to Column 3 in Table 4. This is the input to Phase 2 model. Phase 2 model reverses the *sandhi* operation. The output of Phase 2 model is shown in Column 2.

Input	Final Output
mandāgnim	*manda-agnim*
śuddhātmanaḥ	*śuddha-ātmanaḥ*
pitṛtarpaṇam	*pitṛ-tarpaṇam*

Table 6: Final Results. Column 1 here is the original compound. By replacing the sub-strings in Column 1 of Table 5 by Column 2 of Table 5 in these compounds, we get the final output, as shown in Column 2 of this table.

milliseconds, respectively.

The evaluation was done on 17,304 test cases that were kept separate from training of the Phase 1 model. The Results for Phase 1, Phase 2 and the combined stages is shown in the Table 7. There are 3 rows in the table for Phase 1. The first row indicates how many words were predicted correctly. There can be multiple components in a word, so in this case, a test case is passed only if all the components are predicted correctly. The second row shows how many component boundaries, overall, were predicted correctly. In our method, it is possible to predict the word split correctly even if the split position indicated by the indicator string is off by 1 position to left or right. If the cases in which the split position was off by 1 position are included as correct prediction, the overall accuracy is as shown in the third row in the table as Approximate. Fourth row shows the results achieved for Phase 2 and the fifth row shows the final result when both phases are used together.

The state of the art solutions for Sanskrit word segmentation are Transformer based like the ByT5-Sanskrit proposed by Nehrdich et al. (2024) that work on sentences. Since our solution works on words, not sentences, we evaluate the ByT5-Sanskrit model on our test dataset. To diversify the test dataset, approximately 5000 extra compound words were taken from the SandhiKosh dataset presented by Bhardwaj et al. (2018). Table 8 shows the accuracy achieved by our solution and ByT5-Sanskrit. Every word in the test consists of multiple components. A word is considered pass if all its components are correctly identified. While transformer-based models such as ByT5-Sanskrit achieve strong performance in sentence-level settings, our results indicate that under word-level, context-free evaluation conditions, lightweight neural models can

Total tests	Pass	Fail	Pass %	Description
17304	15177	2127	87.71	Phase 1 results, word-wise
25861	24010	1851	92.84	Phase 1 results, component-wise
25861	24232	1629	93.70	Phase 1 results, Approximate
25708	23661	2047	92.03	Phase 2 results
25861	22296	3565	86.21	Final results, Phase 1 and Phase 2 together

Table 7: Results. Each row displays the results achieved by the models. Final row indicates the final results achieved on component level when the two models were executed sequentially, with Phase 1 preceding Phase 2.

Model	Total Words	Words Pass	Words Pass %	Components	Components Pass	Components Pass %
ByT5-Sanskrit	29977	22001	73.39	74344	57114	76.82
Two-Phase NN decoder	29977	24354	81.24	74344	64471	86.72

Table 8: Two-Phase Decoder compared with Transformer based solution (ByT5-Sanskrit)

offer a better accuracy–efficiency trade-off. However, it bears repetition that ByT5-Sanskrit was trained on sentences, not words and hence works best with context which is missing here.

6 Ablation Study

To better understand the contribution of individual design choices, we conduct a set of ablation experiments on the held-out test set. All results are reported at the component level. The following ablation experiments are intended to assess design decisions rather than to optimize absolute performance.

6.1 Impact of Two-Phase Decomposition

Configuration	**Component Accuracy (%)**
Single-stage Seq2Seq (no Phase 1)	78.6
Proposed Two-Phase Model	**86.2**

Table 9: Effect of two-phase decomposition on segmentation accuracy.

The results in Table 9 show that decoupling boundary detection from sandhi reversal leads to a substantial improvement, validating the design choice.

6.2 Effect of Recurrent Architecture

As shown in Table 10, Bidirectional context improves boundary detection, especially in longer compounds.

6.3 Effect of Sandhi Window Size

A moderate window size captures sufficient phonological context without introducing noise. This is reflected in the results of Table 11.

7 Error Analysis

We manually analyzed 300 randomly sampled errors from the test set. Table 12 summarizes common failure modes.

Many errors arise from rare phonological patterns or deeply recursive compounds, suggesting directions for future improvements.

Phase 1 Architecture	Component Accuracy (%)
Unidirectional LSTM	84.1
BiLSTM	**86.2**

Table 10: Effect of recurrent architecture choice for Phase 1.

Window Size	Phase 2 Accuracy (%)
4 characters	89.4
5 characters	**92.0**
6 characters	91.7

Table 11: Effect of sandhi window size on Phase 2 performance.

8 Conclusion and Future Works

In this paper, we propose the use of a 2-phase, neural-network based method to tackle the long-standing challenge of processing compound words in Sanskrit. Compound segmentation is the first step in analysis of compound words. A high accuracy segmentation engine will ensure that the subsequent steps of the analysis process can also have a higher performance. We will be making our code and data open-source for other researchers to benefit from. The following directions may further improve performance:

1. Our proposed solution works at word level. It will be useful to explore the possibility of taking the context into account. The context can be neighbouring words and their morphological features.

2. The accuracy of our proposed solution is limited by the data and models that we have used. With more training data and more powerful models, it is reasonable to expect the segmentation accuracy to improve.

3. Our proposed solution does not make use of any external lexical resource. By taking cues from existing database of Sanskrit words and their morphology, it should be possible to improve upon the accuracy of our proposed method.

Apart from improvements, it is desirable to see how this model performs for languages other than Sanskrit, especially other Indian languages. Possibilities to adapt or fine-tune our models for compound analysis in other Indian languages need to be explored.

More broadly, our findings suggest that Sanskrit compound processing benefits from explicitly modeling the linguistic separation between structural boundary detection and phonological realization. This separation, implicit in traditional grammatical analysis, proves computationally effective when reflected in neural architectures.

Error Type	Proportion (%)
Deeply nested compounds (> 4 components)	31
Rare sandhi rules	27
Boundary ambiguity without sandhi	22
Annotation noise in alignment corpus	20

Table 12: Distribution of observed error types.

References

Rahul Aralikatte, Neelamadhav Gantayat, Naveen Panwar, Anush Sankaran, and Senthil Mani. 2018. Sanskrit sandhi splitting using seq2(seq)2. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4909–4914, Brussels, Belgium, October-November. Association for Computational Linguistics.

Shubham Bhardwaj, Neelamadhav Gantayat, Nikhil Chaturvedi, Rahul Garg, and Sumeet Agarwal. 2018. SandhiKosh: A benchmark corpus for evaluating Sanskrit sandhi tools. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Bhaṭṭojīdīkṣita. 17th century grammarian. Śabdakaustubha. Caukhambā Sanskrit Series Office.

Stanley F. Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

Sushant Dave, Arun Kumar Singh, Dr. Prathosh A.P., and Prof. Brejesh Lall. 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)*, CODS-COMAD '21, page 171–177, New York, NY, USA. Association for Computing Machinery.

Wilson Fearn, Orion Weller, and Kevin Seppi. 2021. Exploring the relationship between algorithm performance, vocabulary, and run-time in text classification. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3069–3082, Online, June. Association for Computational Linguistics.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2006. Contextual dependencies in unsupervised word segmentation. In Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors, *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 673–680, Sydney, Australia, July. Association for Computational Linguistics.

Oliver Hellwig and Sebastian Nehrdich. 2018. Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2754–2763, Brussels, Belgium, October-November. Association for Computational Linguistics.

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

Gérard Huet. 2005. A functional toolkit for morphological and phonological processing, application to a sanskrit tagger. *J. Funct. Program.*, 15:573–614, 07.

Gérard Huet. 2009. Sanskrit segmentation. In *Proceedings of the South Asian Languages Analysis Roundtable (SALA) XXVIII*, October. South Asian Languages Analysis Roundtable.

Amrith Krishna, Bishal Santra, Pavankumar Satuluri, Sasi Prasanth Bandaru, Bhumi Faldu, Yajuvendra Singh, and Pawan Goyal. 2016. Word segmentation in Sanskrit using path constrained random walks. In Yuji Matsumoto and Rashmi Prasad, editors, *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 494–504, Osaka, Japan, December. The COLING 2016 Organizing Committee.

Amrith Krishna, Bishal Santra, Sasi Prasanth Bandaru, Gaurav Sahu, Vishnu Dutt Sharma, Pavankumar Satuluri, and Pawan Goyal. 2018. Free as in free word order: An energy based model for word segmentation and morphological tagging in Sanskrit. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2550–2561, Brussels, Belgium, October-November. Association for Computational Linguistics.

Amba Kulkarni and Anil Kumar. 2011. Statistical constituency parser for sanskrit compounds. Chennai, India. Proceedings of ICON-2011: 9th International Conference on Natural Language Processing, Macmillan Publishers, India.

Amba Kulkarni and Anil Kumar. 2013. Clues from ashtadhyayi for compound type identification. Mumbai. 5th international SCLS 2013.

Amba Kulkarni, Soma Paul, Malhar Kulkarni, Anil Kumar, and Nitesh Surtani. 2012. Semantic processing of compounds in Indian languages. In Martin Kay and Christian Boitet, editors, *Proceedings of COLING 2012*, pages 1489–1502, Mumbai, India, December. The COLING 2012 Organizing Committee.

Anil Kumar, V. Sheebasudheer, and Amba Kulkarni. 2009. Sanskrit compound paraphrase generator. Hyderabad, India. Proceedings of ICON-2009: 7th International Conference on Natural Language Processing, Hyderabad, India.

Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In Girish Nath Jha, editor, *Sanskrit Computational Linguistics*, pages 57–69, Berlin, Heidelberg. Springer Berlin Heidelberg.

Anil Kumar. 2012. An automatic sanskrit compound processing. A dissertation submitted to the University of Hyderabad for the award of the degree of Doctor of Philosophy in Sanskrit Studies.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Vipul Mittal. 2010. Automatic Sanskrit segmentizer using finite state transducers. In Seniz Demir, Jan Raab, Nils Reiter, Marketa Lopatkova, and Tomek Strzalkowski, editors, *Proceedings of the ACL 2010 Student Research Workshop*, pages 85–90, Uppsala, Sweden, July. Association for Computational Linguistics.

Abhiram Natarajan and Eugene Charniak. 2011. s^3 - statistical sandhi splitting. In Haifeng Wang and David Yarowsky, editors, *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 301–308, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

Sebastian Nehrdich, Oliver Hellwig, and Kurt Keutzer. 2024. One model is all you need: ByT5-Sanskrit, a unified model for Sanskrit NLP tasks. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13742–13751, Miami, Florida, USA, November. Association for Computational Linguistics.

Alan Prince. 2004. Optimality theory: Constraint interaction in generative grammar. *University, New Brunswick, and University of Colorado.*

Vikas Reddy, Amrith Krishna, Vishnu Sharma, Prateek Gupta, Vineeth M R, and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Jivnesh Sandhan, Rathin Singha, Narein Rao, Suvendu Samanta, Laxmidhar Behera, and Pawan Goyal. 2022. TransLIST: A transformer-based linguistically informed Sanskrit tokenizer. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6902–6912, Abu Dhabi, United Arab Emirates, December. Association for Computational Linguistics.

Arun Kumar Singh, Sushant Dave, Shresth Mehta, and Prathosh A.P. 2022. A data driven approach for sanskrit derivative nouns analysis. In *Proceedings of the 5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD)*, CODS-COMAD '22, page 288–289, New York, NY, USA. Association for Computing Machinery.

Krishnan Sriram, Amba Kulkarni, and Gérard Huet. 2023. Validation and normalization of DCS corpus and development of the Sanskrit heritage engine's segmenter. In Amba Kulkarni and Oliver Hellwig, editors, *Proceedings of the Computational Sanskrit & Digital Humanities: Selected papers presented at the 18th World Sanskrit Conference*, pages 38–58, Canberra, Australia (Online mode), January. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.