TransLIST: A Transformer-Based Linguistically Informed Sanskrit Tokenizer

Jivnesh Sandhan[1], Rathin Singha[2], Narein Rao[1], Suvendu Samanta[1], Laxmidhar Behera[1,4] and Pawan Goyal[3]

[1]IIT Kanpur, [2]UCLA, [3]IIT Kharagpur, [4]IIT Mandi

jivnesh@iitk.ac.in, rsingha108@g.ucla.edu, nrao20@iitk.ac.in, pawang@cse.iitkgp.ac.in

Abstract

Sanskrit Word Segmentation (SWS) is essential in making digitized texts available and in deploying downstream tasks. It is, however, non-trivial because of the *sandhi* phenomenon that modifies the characters at the word boundaries, and needs special treatment. Existing *lexicon driven* approaches for SWS make use of Sanskrit Heritage Reader, a lexicon-driven shallow parser, to generate the complete candidate solution space, over which various methods are applied to produce the most valid solution. However, these approaches fail while encountering out-of-vocabulary tokens. On the other hand, *purely engineering* methods for SWS have made use of recent advances in deep learning, but cannot make use of the latent word information on availability.

To mitigate the shortcomings of both families of approaches, we propose **Trans**former based **L**inguistically **I**nformed **S**anskrit **T**okenizer (TransLIST) consisting of (1) a module that encodes the character input along with latent-word information, which takes into account the *sandhi* phenomenon specific to SWS and is apt to work with partial or no candidate solutions, (2) a novel soft-masked attention to prioritize potential candidate words and (3) a novel path ranking algorithm to rectify the corrupted predictions. Experiments on the benchmark datasets for SWS show that TransLIST outperforms the current state-of-the-art system by an average 7.2 points absolute gain in terms of perfect match (PM) metric.[1]

1 Introduction

Sanskrit is considered as a cultural heritage and knowledge preserving language of ancient India. The momentous development in digitization efforts has made ancient manuscripts in Sanskrit readily available for the public domain. However, the usability of these digitized manuscripts is limited due to linguistic challenges posed by the language. SWS conventionally serves the most fundamental prerequisite for text processing step to make these digitized manuscripts accessible and to deploy many downstream tasks such as text classification (Sandhan et al., 2019; Krishna et al., 2016b), morphological tagging (Gupta et al., 2020; Krishna et al., 2018), dependency parsing (Sandhan et al., 2021; Krishna et al., 2020a), automatic speech recognition (Kumar et al., 2022) etc. SWS is not straightforward due to the phenomenon of *sandhi*, which creates phonetic transformations at word boundaries. This not only obscures the word boundaries but also modifies the characters at juncture point by deletion, insertion and substitution operation. Figure 1 illustrates some of the syntactically possible splits due to the language-specific *sandhi* phenomenon for Sanskrit. This demonstrates the challenges involved in identifying the location of the split and the kind of transformation performed at word boundaries.

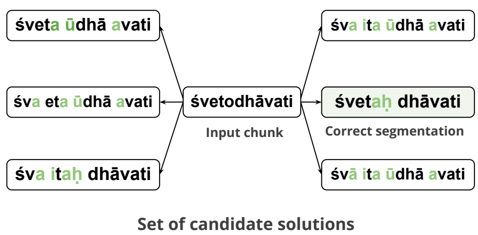


Figure 1: An example to illustrate challenges posed by *sandhi* phenomenon for SWS task.

The recent surge in SWS datasets (Krishna et al., 2017; Krishnan et al., 2020) has led to various methodologies to handle SWS. Existing *lexicon-driven* approaches rely on a *lexicon driven* shallow parser, popularly known as Sanskrit Heritage Reader (SHR) (Goyal and Huet, 2016a).[2] This line of approaches (Krishna et al., 2016a, 2018, 2020b)

[1]The codebase and datasets are publicly available at: https://github.com/rsingha108/TransLIST

[2]https://sanskrit.inria.fr/DICO/reader.fr.html

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 6902–6912

formulate the task as finding the most accurate semantically and syntactically valid solution from the candidate solutions generated by SHR. With the help of the significantly reduced exponential search space provided by SHR and linguistically involved feature engineering, these *lexicon driven* systems (Krishna et al., 2020b, 2018) report close to state-of-the-art performance for the SWS task. However, these approaches rely on the completeness assumption of SHR, which is optimistic given that SHR does not use domain specific lexicons. These models are handicapped by the failure of this preliminary step. On the other hand, *purely engineering* based knowledge-lean data-centric approaches (Hellwig and Nehrdich, 2018; Reddy et al., 2018; Aralikatte et al., 2018) perform surprisingly well without any explicit hand-crafted features and external linguistic resources. These *purely engineering* based approaches are known for their ease of scalability and deployment for training/inference. However, a drawback of these approaches is that they are blind to latent word information available through external resources.

There are also lattice-structured approaches (Zhang and Yang, 2018; Gui et al., 2019; Li et al., 2020) (originally proposed for Chinese Named Entity Recognition (NER), which incorporate lexical information in character-level sequence labelling architecture). However, these approaches cannot be directly applied for SWS; since acquiring word-level information is not trivial due to *sandhi* phenomenon. To overcome these shortcomings, we propose **Trans**former-based **L**inguistically **I**nformed **T**okenizer (TransLIST). TransLIST is a perfect blend of *purely engineering* and *lexicon driven* approaches for the SWS task and provides the following advantages: (1) Similar to *purely engineering* approaches, it facilitates ease of scalability and deployment during training/inference. (2) Similar to *lexicon driven* approaches, it is capable of utilizing the candidate solutions generated by SHR, which further improves the performance. (3) Contrary to *lexicon driven* approaches, TransLIST is robust and can function even when candidate solution space is partly available or unavailable.

Our key contributions are as follows: (a) We propose the linguistically informed tokenization module (§ 2.1) which accommodates language-specific *sandhi* phenomenon and adds inductive bias for the SWS task. (b) We propose a novel soft-masked attention (§ 2.2) that helps to add inductive bias for prioritizing potential candidates keeping mutual interactions between all candidates intact. (c) We propose a novel path ranking algorithm (§ 2.3) to rectify the corrupted predictions. (d) We report an average 7.2 points perfect match absolute gain (§ 3) over the current state-of-the-art system (Hellwig and Nehrdich, 2018).

We elucidate our findings by first describing TransLIST and its key components (§ 2), followed by the evaluation of TransLIST against strong baselines on a test-bed of 2 benchmark datasets for the SWS task (§ 3). Finally, we investigate and delve deeper into the capabilities of the proposed components and its corresponding modules (§ 4).

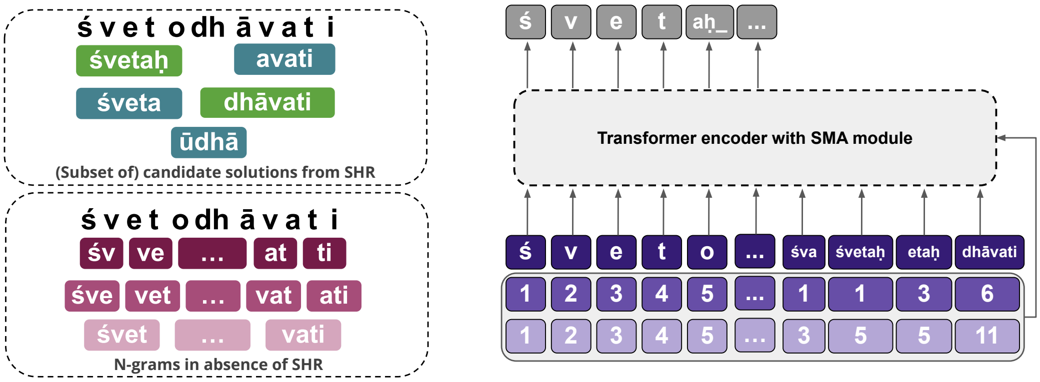
2 Methodology

In this section, we will examine the key components of TransLIST which includes a linguistically informed tokenization module that encodes character input with latent-word information while accounting for SWS-specific *sandhi* phenomena (§ 2.1), a novel soft-masked attention to prioritise potential candidate words (§ 2.2) and a novel path ranking algorithm to correct mispredictions (§ 2.3).

2.1 Linguistically Informed Sanskrit Tokenizer (LIST)

Lexicon driven approaches for SWS are brittle in realistic scenarios and *purely engineering* based approaches do not consider the potentially useful latent word information. We propose a win-win/robust solution by formulating SWS as a character-level sequence labelling integrated with latent word information from the SHR as and when available. TransLIST is illustrated with an example *śvetodhāvati* in Figure 2. SHR employs a Finite State Transducer (FST) in the form of a lexical juncture system to obtain a compact representation of candidate solution space aligned with the input sequence. As shown in Figure 2(a), we receive the candidate solution space from the SHR engine. Here, *śvetah dhāvati* and *śveta ūdhā avati* are two syntactically possible splits.[3] It does not suggest the final segmentation. The candidate space includes words such as *śva, śvetaḥ* and *etaḥ* whose boundaries are modified with respect to the input sequence due to *sandhi* phenomenon. SHR gives us mapping (head and tail position) of all the candidate nodes with the input sequence. In

[3]Only some of the solutions are shown for visualization.



(a) (b)

Figure 2: Illustration of TransLIST with a toy example "*śvetodhāvati*". Translation: "The white (horse) runs." (a) LIST module: We use the candidate solutions (two possible candidate solutions are highlighted with ■,■ colors where the latter is the gold standard) from SHR if available; in the absence of SHR, we resort to using n-grams ($n \leq 4$). (b) TransLIST architecture: In span encoding, each node is represented by head and tail position index of its character in the input sequence. ■, ■, ■ denote tokens, heads and tails, respectively. The SHR helps to include words such as *śva, śvetaḥ* and *etaḥ* whose boundaries are modified with respect to input sequence due to *sandhi* phenomenon. Finally, on the top of the Transformer encoder, classification head learns to predict gold standard output shown by ■ for the corresponding input character nodes only.

case such mapping is incorrect for some cases, we rectify it with the help of deterministic algorithm by matching candidate nodes with the input sentence and finding the closest match. In the absence of SHR, we propose to use all possible n-grams ($n \leq 4$)[4]which helps to add inductive bias about neighboring candidates in the window size of 4.[5] We feed the candidate words/n-grams to the Transformer encoder and the classification head learns to predict gold standard output for the corresponding input character nodes only. The output vocabulary consists of unigram characters (e.g., ś, v), bigrams and tri-grams (e.g., aḥ_). The output vocabulary contains '_' to represent spacing between words. Consequently, TransLIST is capable of using both character-level modelling as well as latent word information as and when available. On the other hand, *purely engineering* approaches rely only on character-level modelling and *Lexicon driven* approaches rely only on word-level information from SHR to handle *sandhi*.

2.2 Soft Masked Attention (SMA)

Transformers (Vaswani et al., 2017) have been proven to be effective for capturing long-distance dependencies in a sequence. The self-attention property of a Transformer facilitates effective interaction between character and available latent word information. There are two preliminary prerequisites for effective modelling of inductive bias for tokenization: (1) Allow interactions between the candidate words/characters within and amongst chunks. (2) Prioritize candidate words containing the input character for which a prediction is being made (e.g., in Figure 2(b), *śva* and *śvetaḥ* are prioritized amongst the candidate words when predicting for the character *ś*).[6] The vanilla self-attention (Vaswani et al., 2017) can address both the requirements; however, it has to self-learn the inductive bias associated with prioritisation. It may not be an effective solution in low-resourced settings. On the other hand, if we use hard-masked attention to address the second prerequisite, we lose mutual interactions between the candidates. Hence, we propose a novel soft-masked attention which helps to address both the requirements effectively. To the best of our knowledge, there is no existing soft-masked attention similar to ours. We formally discuss this below.

Self-attention maps a query and a set of key-value pairs to an output as discussed in Vaswani et al. (2017). For an input $x = (x_1, ..., x_n)$

[4]We do not observe significant improvements for $n > 4$.

[5]Our probing analysis (Figure 4) suggests that char-char attention mostly focuses on immediate neighbors. Refer to § 4 for detailed ablations on LIST variants.

[6]We find that failing to meet any one of the prerequisites leads to drop in performance (§ 4).

where $x_i \in R^{d_x}$, self-attention gives an output $z = (z_1, ..., z_n)$ where $z_i \in R^{d_z}$. We presume the standard formulation of vanilla self-attention (Vaswani et al., 2017) where d_x is the dimension of input word representation and d_z is the projection dimension. Here, $W^Q, W^K, W^V \in R^{d_x \times d_z}$ are parameter matrices. For simplicity, we ignore multi-head attention in equations 1, 2 and 3.

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V) \tag{1}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \tag{2}$$

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}} \tag{3}$$

In **soft-masked attention**, we provide a prior about interactions between candidate words and the input characters using a span encoding ($s_{ij} \in R^{d_z}$) (Li et al., 2020). Intuitively, it helps inject inductive bias associated with prioritisation whilst maintaining mutual interactions between the candidates.

Formally, we modify Equation 2 to define soft masked attention as:

$$\alpha_{ij}^{SM} = \frac{M_{ij} \exp(e_{ij})}{\sum_{k=1}^{n} M_{ik} \exp(e_{ik})} \tag{4}$$

where $M \in R^{n \times n}$, $M_{ij} \in [0, 1]$. M_{ij} is defined as:

$$M_{ij} = \frac{(x_i W^Q)(s_{ij} W^R)^T}{\sqrt{d_z}} \tag{5}$$

$W^R \in R^{d_z \times d_z}$ is a learnable parameter which projects s_{ij} into a location-based key vector space. Summarily, the proposed SMA module helps to prioritize potential candidate words with the help of separation, inclusion and intersection information between nodes. Finally, we calculate the output z with the help of the proposed SMA as follows:

$$z_i = \sum_{j=1}^{n} \alpha_{ij}^{SM}(x_j W^V) \tag{6}$$

Next, we discuss the span position encoding.

Span position encoding is one of the backbones of the proposed soft-masked module. It is utilized to capture the interactions between the candidate words and the sequence of input characters. Each span/node (which is a character/word and its corresponding position in the input sentence) is represented by the head and tail which denote the position index of the initial and final characters of the token in the input sequence, as shown in Figure 2(b). The span of character is characterized by the same head and tail position index. For example, $head[i]$ and $tail[i]$ represent the head and tail index of span x_i, respectively. The separation, inclusion and intersection information between nodes x_i and x_j can be captured by the four distance equations 7-10.

$$d_{ij}^{(hh)} = head[i] - head[j] \tag{7}$$

$$d_{ij}^{(ht)} = head[i] - tail[j] \tag{8}$$

$$d_{ij}^{(th)} = tail[i] - head[j] \tag{9}$$

$$d_{ij}^{(tt)} = tail[i] - tail[j] \tag{10}$$

The final span encoding is a non-linear transformation of these 4 distances:

$$s_{ij} = \mathrm{ReLU}(w_s(p_{d_{ij}^{(hh)}} \oplus p_{d_{ij}^{(ht)}} \oplus p_{d_{ij}^{(th)}} \oplus p_{d_{ij}^{(tt)}})) \tag{11}$$

where $w_s \in R$ is a learnable parameter, $\oplus$ is a concatenation operation and $p_d \in R^{\frac{d_z}{4}}$ is a sinusoidal position encoding similar to Vaswani et al. (2017).

2.3 Path Ranking for Corrupted Predictions (PRCP)

Our error analysis (§ 4) suggests that sometimes the proposed system predicts words that are not part of the candidate solution space. These mistakes can be rectified with the help of SHR's candidate solutions by appropriately substituting suitable candidates. We refer to the prediction corresponding to a chunk that does not fall in the candidate solution space, as a *corrupted prediction* and define a *path* as the sequence of characters in a candidate solution for a given input. We enumerate all the possible directed paths (In Figure 2(a), two possible candidate solutions are highlighted with ■,■ colors) corresponding to the input (with a corrupted prediction) and formulate the task as a *path ranking problem*. While designing the path scoring function (S), we consider the following criteria: (1) Select a path consisting of semantically coherent candidate words. We use an integrated judgment from two sources. First, we prefer a path having a high log-likelihood (LL) score as per TransLIST to choose a semantically coherent path in line with the contextual information of TransLIST. Second, we reinforce the scoring function (S) by considering the perplexity score (ρ) for the path from the character-level language model. (2) To avoid paths consisting of over-generated segmentation provided by SHR, we use a penalty proportional to the number of words ($|W|$) present in the path to prefer paths with less number of words. This gives us the following path scoring function (S):

$$S = \frac{LL_{TransLIST}}{\rho_{CharLM} \times |W|}$$

where

$$\begin{aligned} LL_{TransLIST} &= \text{log-likelihood by TransLIST} \\ \rho_{CharLM} &= \text{Perplexity score by CharLM} \\ |W| &= \text{Number of words present in path} \end{aligned}$$

3 Experiments

Data and Metrics: Currently, Digital Corpus of Sanskrit (Hellwig, 2010, DCS) has more than 600,000 morphologically tagged text lines. It consists of digitized constructions composed in prose or poetry over a wide span of 3000 years. Summarily, DCS is a perfect representation of various writing styles depending on time and domains. We use two available benchmark datasets (Krishna et al., 2017, SIGHUM)[7] and (Krishnan et al., 2020, Hackathon) for SWS. Both datasets are subset of DCS (Hellwig, 2010). These datasets also come with candidate solution space generated by SHR for SWS. We prefer Krishna et al. (2017, SIGHUM) over a relatively larger dataset (Hellwig and Nehrdich, 2018) to obviate the time and efforts required for obtaining candidate solution space. We obtain the ground truth segmentation solutions from DCS. We could not use DCS10k (Krishna et al., 2020b) due to partly missing gold standard segmentation (inflections) for almost 50% data points. SIGHUM consists of 97,000, 3,000 and 4,200 sentences as train, dev, test set, respectively. Similarly, Hackathon consists of 90,000, 10,332 and 9,963 sentences as train, dev and test set, respectively. We use the following word-level evaluation metrics: macro-averaged Precision (P), Recall (R), F1-score (F) and the percentage of sentences with perfect matching (PM).

[7]https://zenodo.org/record/803508#.YRdZ43UzaXJ

Hyper-parameter settings: For the implementation of TransLIST, we build on top of codebase by Li et al. (2020). We use the following hyper-parameters for the best configuration of TransLIST: number of epochs as 50 and a dropout rate of 0.3 with a learning rate of 0.001. We release our codebase and datasets publicly under the Apache license 2.0. All the artifacts used in this work are publicly available for the research purpose. For all the systems, we do not use any pretraining. All the input representations are randomly initialized. We use GeForce RTX 2080, 11 GB GPU memory computing infrastructure for our experiments.

Baselines: We consider two *lexicon-driven* approaches where Krishna et al. (2016a, **SupPCRW**) formulate SWS as an iterative query expansion problem and Krishna et al. (2018, **Cliq-EBM**) deploy a structured prediction framework. Next, we evaluate four *purely-engineering* based approaches, namely, Encoder-Decoder framework (Reddy et al., 2018, **Seq2Seq**), character-level sequence labelling system with combination of recurrent and convolution element (Hellwig and Nehrdich, 2018, **rcNN-SS**), vanilla **Transformer** (Vaswani et al., 2017) and character-level Transformer with relative position encoding (Yan et al., 2019, **TENER**). Finally, we consider lattice-structured approaches originally proposed for Chinese NER which incorporate lexical information in character-level sequence labelling architecture. These approaches consist of lattice-structured LSTM (Zhang and Yang, 2018, **Lattice-LSTM**), graph neural network (GNN) based architecture (Gui et al., 2019, **Lattice-GNN**) and Transformer based architecture (Li et al., 2020, **FLAT-Lattice**). **TransLIST:** As per § 2.1, we report two variants: (a) $\text{TransLIST}_{\text{ngrams}}$ which makes use of only n-grams, and (b) TransLIST which makes use of SHR candidate space.

Results: Table 1 reports the results for the best performing configurations of all the baselines on the test set of benchmark datasets for the SWS task.[8] Except *purely engineering* based systems (Seq2seq, TENER, Transformer and rcNN-SS), all systems leverage linguistically refined candidate solution space generated by SHR. Among the lattice-structured systems, FLAT-Lattice demonstrates competing performance against rcNN-SS.

[8]We do not compare with recently proposed variant of Clique-EBM (Krishna et al., 2020b) and seq2seq baseline (Aralikatte et al., 2018) due to unavailability of codebase. Also, they do not report performance on these two datasets.

	SIGHUM				Hackathon			
Model	**P**	**R**	**F**	**PM**	**P**	**R**	**F**	**PM**
Seq2seq	73.44	73.04	73.24	29.20	72.31	72.15	72.23	20.21
SupPCRW	76.30	79.47	77.85	38.64	-	-	-	-
TENER	90.03	89.20	89.61	61.24	89.38	87.33	88.35	49.92
Lattice-LSTM	94.36	93.83	94.09	76.99	91.47	89.19	90.31	65.76
Lattice-GNN	95.76	95.24	95.50	81.58	92.89	94.31	93.59	70.31
Transformer	96.52	96.21	96.36	83.88	95.79	95.23	95.51	77.70
FLAT-Lattice	96.75	96.70	96.72	85.65	<u>96.44</u>	<u>95.43</u>	<u>95.93</u>	<u>77.94</u>
Cliq-EBM	96.18	<u>97.67</u>	<u>96.92</u>	78.83	-	-	-	-
rcNN-SS	<u>96.86</u>	96.83	96.84	<u>87.08</u>	96.40	95.15	95.77	77.62
TransLIST$_{\text{ngrams}}$	96.97	96.77	96.87	86.52	96.68	95.74	96.21	79.28
TransLIST	**98.80**	**98.93**	**98.86**	**93.97**	**97.78**	**97.44**	**97.61**	**85.47**

Table 1: Performance evaluation between baselines in terms of P, R, F and PM metrics. The significance test between the best baselines, rcNN-ss, FLAT-lattice and TransLIST in terms of recall/perfect-match metrics: $p < 0.05$ (as per t-test, for both the datasets). We do not report the performance of SupPCRW and Cliq-EBM on Hackathon dataset due to unavailability of codebase. On SIGHUM, we report numbers from their papers. The best baseline's results for the corresponding datasets are underlined. The overall best results per column are highlighted in bold.

We find that rcNN-SS and FLAT-Lattice perform the best among all the baselines on SIGHUM and Hackathon datasets, respectively.

Both the TransLIST variants outperforms all the baselines in terms of all the evaluation metrics with TransLIST providing an average 1.8 points (F) and 7.2 points (PM) absolute gain with respect to the best baseline systems, rcNN-SS (on SIGHUM) and FLAT-Lattice (on Hackathon). Even when the SHR candidate space is not available, the proposed system can use TransLIST$_{\text{ngrams}}$, which provides an average 0.11 points (F) and 0.39 points (PM) absolute gain over the best baselines. TransLIST$_{\text{ngrams}}$ gives comparable performance to rcNN-SS on SIGHUM dataset, while on the Hackathon dataset, it performs significantly better than FLAT-Lattice ($p < 0.05$ as per t-test). The wide performance gap between TransLIST and TransLIST$_{\text{ngrams}}$ demonstrates the effectiveness of using SHR candidate space, when available. Summarily, we establish a new state-of-the-art results with the help of meticulously stitched LIST, SMA and PRCP modules. The knowledge of the candidate space by SHR gives an extra advantage to TransLIST. Otherwise, natural choice is the proposed purely engineering variant TransLIST$_{\text{ngrams}}$ when that is not available.

4 Analysis

In this section, we investigate various questions to dive deeper into the proposed components and investigate the capabilities of various modules. We use SIGHUM dataset for the analysis.

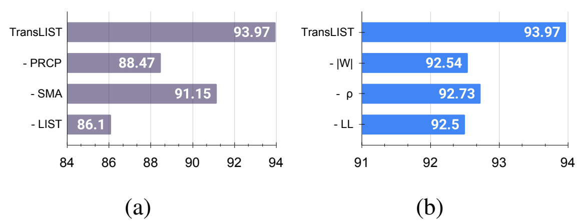


Figure 3: Ablations on (a) TransLIST (b) PRCP module in terms of PM (SIGHUM-test). Each ablation in (a) removes a single module from TransLIST. For example, "-SMA" removes SMA from TransLIST. For (b), ablations are shown by removing a particular term from path scoring function (S).

(1) Ablation analysis: Here, we study the contribution of different modules towards the final performance of TransLIST. Figure 3(a) illustrates ablations in terms of PM when a specific module is removed from TransLIST. For instance, '-LIST' corresponds to character-level transformer encoder with SMA and PRCP. Removal of any of the modules degrades the performance. Figure 3(a) shows that LIST module is the most crucial for providing inductive bias of tokenization. Also, removal of 'PRCP' module has a large impact on the performance. We observe that the PRCP module gets activated for 276 data points out of 4,200 data points in the test set. We then deep dive into the PRCP path scoring function in Figure 3(b), which consists of 3 terms, namely, penalty ($|W|$), perplexity

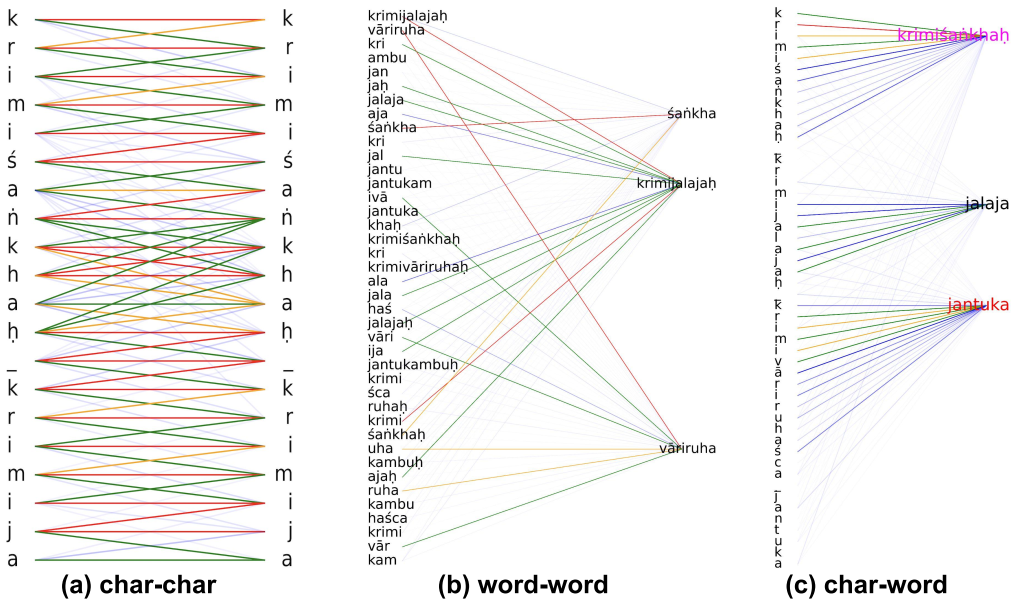


Figure 4: SMA probing: Illustration of char-char, char-word and word-word interactions. The strength of the SMA decreases in the following order: red, orange, green and blue. Char-char attention mostly focuses on characters present in the vicinity of window size 1. Word-word interactions are able to capture whether a word is subword of another word or not. Finally, we find that quality of attention goes down for char-word as we move as per the following order: in vocabulary gold words (pink), in vocabulary non-golds (black) and out-of-vocabulary words (red). Some of the attentions are invisible due to very low attention score.

score by CharLM ($|\rho|$) and log-likelihood (LL) by TransLIST, respectively. We remove a single term at a time from the path scoring function, and observe each of the terms used in the scoring function plays a major role in the final performance.

(2) Comparative analysis of potential LIST module variants to add inductive bias for tokenization: We evaluate possible LIST variants which can help inject inductive bias for tokenization via auxiliary (word) nodes illustrated in Figure 2(b): (a) *sandhi* rules: We use *sandhi* rules as a proxy to indicate potential modifications at specific position in the input sequence. For example, if input chunk contains the character *'o'* (Figure 1) then it can be substituted with two possibilities $\bar{o} \rightarrow$ *a-ū/aḥ*. We provide this proxy information through auxiliary nodes. (b) Sanskrit vocab: We obtain a list of vocabulary words from DCS corpus (Hellwig, 2010) and add the words which can be mapped to the input character sequence using a string matching algorithm. (c) n-grams: This is TransLIST$_{\text{ngrams}}$ (d) SHR: We follow the exact settings as described in § 2.1 except that we do not use the PRCP component. In Table 2, we compare these with the *purely engineering* variant of TransLIST (Base system: only character-level Transformer) where no inductive bias for tokenization is injected. Clearly, due to availability of enriched candidate space, SHR variant outperforms all its peers. However, competing performance of n-gram variant is appealing because it completely obliviates the dependency on SHR and remains unaffected in the absence of SHR's candidate space.

System	P	R	F	PM
Base system	92.75	92.62	92.69	72.33
+*sandhi* rules	93.53	93.70	93.62	75.71
+Sanskrit Vocab	96.75	96.70	96.72	85.65
+n-grams	96.97	96.77	96.87	86.52
+SHR	**97.79**	**97.45**	**97.62**	**88.47**

Table 2: The comparison (on SIGHUM-test set) between LIST variants. '+' indicates system where the corresponding variant is augmented with the base system. We do not activate PRCP for any of these systems.

(3) Probing analysis on SMA: Here we analyze whether SMA upholds the prerequisite for effective modelling of inductive bias, i.e., prioritize candidate words which contain the input character for which the prediction is being made. Figure 4 illustrates three types of interactions, namely, char-char, char-word and word-word. We use color coding scheme to indicate the strength of attention weight. The attention weight decreases in the following order: Red, Orange, Green and Blue. Char-char attention mostly focuses on characters present in the vicinity of window size 1. This local information is relevant to make decisions regarding possible *sandhi* split. Word-word interactions are able to capture whether a word is subword of another word or not. Finally, for char-word attention, we find that quality of attention goes down as we move as per the following order: in vocabulary gold words (pink), in vocabulary non-golds (black) and out-of-vocabulary (unseen in training but recognized by SHR) gold words (red). While the drop in attention from in-vocabulary gold tokens to out-of-vocabulary gold tokens is expected, the drop in attention from gold tokens to non-gold tokens is desired. Thus, this probing analysis suggests that SMA module helps to improve intra/inter interactions between character/words and this substantiates the need of SMA module in TransLIST.

(4) How does TransLIST perform in a non-trivial situation where multiple *sandhi* rules are applicable? In Table 3, we report the comparison with rcNN-SS for a critical scenario of a *sandhi* phenomenon. Table 3 represents the possible *sandhi* rules that generate the surface character *ā*. Following Goyal and Huet (2016b), the sandhi rewrite rules are formalized as $u|v \rightarrow f/x\_\_$ (Kaplan and Kay, 1994) where $x, v, f \in \Sigma$, and $u \in \Sigma^{+}$. Σ is the collection of phonemes, Σ^{*}: a set of all possible strings over Σ, and $\Sigma^{+} = \Sigma^{*} - \epsilon$. For example, the potential outputs for the input ā can be ā, ā-ā, ā-a, a-a and aḥ. The correct rule can be decided based on the context. These multiple rules pose a non-trivial challenge for a system to identify the applicability of specific rule. Therefore, it is interesting to compare the TransLIST with current state-of-the-art system to verify its ability for semantic generalization. We observe that TransLIST consistently outperforms rcNN-SS in terms of all metrics.[9] Table 3 describes rules in decreasing order of their frequency. Interestingly, we notice large improvements over the current state-of-the-art system, especially for rare *sandhi* rules. This observation confirms superior performance of TransLIST over the current state-of-the-art system.

[9]Follwing Hellwig and Nehrdich (2018), we report character-level F-score metric. $P = \frac{|S_g \cap S_p|}{|S_p|}$; $R = \frac{|S_g \cap S_p|}{|S_g|}$, $F1 = \frac{2PR}{P+R}$, (S_g) : Set of locations where the rule occurs in gold output, (S_p) : Set of locations where the rule is predicted.

	rcNN-SS			TransLIST		
Rules	**P**	**R**	**F**	**P**	**R**	**F**
ā	99.3	99.3	99.3	99.7	99.6	99.6
a-a	95.4	96.6	96.0	96.6	97.8	97.2
ā-a	88.4	83.1	86.5	90.5	83.8	87.0
āḥ	76.7	70.1	73.7	77.2	80.1	78.0
ā-ā	50.1	42.1	45.7	80.0	40.9	53.3

Table 3: The comparison (on SIGHUM-test set) in terms of P, R and F metrics between rcNN-SS and the TransLIST for ambiguous *sandhi* rules leading to the same surface character *ā*. The proposed model consistently outperforms rcNN-SS in all the metrics.

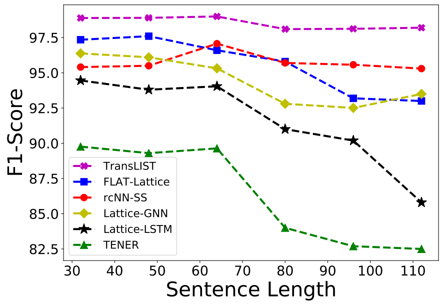


Figure 5: F1-score against sentence length (no. of characters) over the SIGHUM dataset

(5) How robust is the system when sentence length is varied? In Figure 5, we analyze the performance of the baselines with different sentence lengths. We plot the F1-score against sentence length. Clearly, while all the systems show superior performance for shorter sentences, TransLIST is much more robust for longer sentences compared to other baselines. The lattice-structured baselines give competing F1-scores over short sentences but relatively sub-par performance over long sentences.

(6) Illustration of PRCP with an example: Table 4 illustrates an example that probes the effectiveness of PRCP in TransLIST. We compare TransLIST with rcNN-SS and observe that TransLIST also predicts words out of candidate solution space when PRCP module is not activated. However, the degree of such mistakes in TransLIST is comparatively less due to effective modelling of inductive bias for tokenization using LIST and SMA modules. In Table 4, rcNN-SS predicts three words which are not part of candidate space, namely, *vāmbike, yakṣavapuḥ* and *caka*. These are mistakes that can be rectified with the help of available candidate space. Interestingly, TransLIST commits only a single mistake in this

	Sentence	F-score
Input sentence	kimetadīśe bahuśobhamāne vāṃbike yakṣavapuścakāsti *Translation:* What is this body resembling a *Yaksha* that glows, oh Ambika! You who lord over! You who shine!	-
Correct segmentation	kim etat īśe bahu śobhamāne vā ambike yakṣa vapuḥ cakāsti	-
SHR candidate space	kim, etat, īśe, bahu, śobhamāne, śobham, āne, śobha, māne, mā, vā, ambike, yakṣa, vapuḥ, cakāsti, ca, kā, asti *Word-word meaning:* what, this, the one who lord, very much, the one who shine, bright, mouth, I respect, never, or, Parvati, a kind of celestial being, body, glows, and, who (female), is there (be).	-
rcNN-SS	kim etat īśe bahu śobhamāne **vāmbike yakṣavapuḥ caka asti**	52.60
TransLIST-PRCP	kim etat īśe bahu śobhamāne vā **aambike** yakṣa vapuḥ cakāsti	90.00
TransLIST	kim etat īśe bahu śobhamāne vā ambike yakṣa vapuḥ cakāsti	100.00

Table 4: An example to illustrate the effectiveness of PRCP module of TransLIST. Bold represents incorrect segmentation for the input sequence.

category by predicting out of solution space word *aambike*. PRCP aids in mitigating such mistake by appropriately substituting suitable candidates.

5 Related Work

Earlier approaches on SWS focused on rule-based Finite State Transducer systems (Gérard, 2003; Mittal, 2010). Natarajan and Charniak (2011) attempted to solve the SWS task for sentences with one or two splits using the Bayesian approach. Recently, Goyal and Huet (2016a, SHR) proposed a *lexicon driven* shallow parser. This, along with the recent upsurge in segmentation datasets (Krishna et al., 2017; Hellwig and Nehrdich, 2018; Krishnan et al., 2020) led to two categories of approaches, namely, *lexicon driven* (Krishna et al., 2016a, 2018, 2020b) and *purely engineering* (Hellwig, 2015; Hellwig and Nehrdich, 2018; Aralikatte et al., 2018; Reddy et al., 2018). These existing approaches for SWS are either brittle in realistic scenarios or do not consider the potentially useful/available information. Thus, TransLIST bridges the shortcomings exhibited by each family and gives a win-win solution that marks a new state-of-the-art results.

6 Conclusion and Discussion

In this work, we focused on Sanskrit word segmentation task. To address the shortcomings of existing *purely engineering* and *lexicon driven* approaches, we demonstrate the efficacy of TransLIST as a win-win solution over drawbacks of the individual lines of approaches. TransLIST induces inductive bias for tokenization in a character input sequence using the LIST module, and prioritizes the relevant candidate words with the help of soft-masked attention (SMA module). Further, we propose a novel path ranking algorithm to rectify corrupted predictions using linguistic resources on availability (PRCP module). Our experiments showed that TransLIST provides a significant boost with an average 7.2 points (PM) absolute gain compared to the best baselines, rcNN-SS (SIGHUM) and FLAT-Lattice (Hackathon). We have also showcased fine-grained analysis on TransLIST's inner working. We plan to extend this work for morphological tagging in standalone mode (Gupta et al., 2020) and multi-task setting (Krishna et al., 2018) with the SWS task.

Limitations

The preliminary requirement to extend TransLIST for the languages which also exhibit *sandhi* phenomenon is *lexicon-driven* shallow parser similar to Sanskrit Heritage Reader (SHR). Otherwise, the natural choice is the proposed purely engineering variant $\text{TransLIST}_{\text{ngram}}$. It would be interesting to check if TransLIST and $\text{TransLIST}_{\text{ngram}}$ can be used together.

Ethics Statement

We do not foresee any ethical concerns with the work presented in this manuscript.

Acknowledgements

We are grateful to Oliver Hellwig for providing the DCS Corpus and Gerard Huet for providing the Sanskrit Heritage Engine. We thank Sriram Krishnan, University of Hyderabad and Hackathon organizers[10] for providing Hackathon dataset. We

[10] https://sanskritpanini.github.io/index.html

thank Amrith Krishna, University of Cambridge for clarifying our queries related to SIGHUM dataset and evaluation metrics. We are grateful to Rishabh Kumar, IIT Bombay for helping us with evaluation of Cliq-EBM baseline. We would like to thank the anonymous reviewers for their constructive feedback towards improving this work. The work of the first author is supported by the TCS Fellowship under the Project TCS/EE/2011191P.

References

Rahul Aralikatte, Neelamadhav Gantayat, Naveen Panwar, Anush Sankaran, and Senthil Mani. 2018. Sanskrit sandhi splitting using seq2(seq)2. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4909–4914, Brussels, Belgium. Association for Computational Linguistics.

Huet Gérard. 2003. Lexicon-directed segmentation and tagging of sanskrit. In *XIIth World Sanskrit Conference, Helsinki, Finland, Aug*, pages 307–325. Citeseer.

Pawan Goyal and Gérard Huet. 2016a. Design and analysis of a lean interface for sanskrit corpus annotation. *Journal of Language Modelling*, 4:145.

Pawan Goyal and Gérard Huet. 2016b. Design and analysis of a lean interface for sanskrit corpus annotation. *Journal of Language Modelling*, 4:145.

Tao Gui, Yicheng Zou, Qi Zhang, Minlong Peng, Jinlan Fu, Zhongyu Wei, and Xuanjing Huang. 2019. A lexicon-based graph neural network for Chinese NER. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1040–1050, Hong Kong, China. Association for Computational Linguistics.

Ashim Gupta, Amrith Krishna, Pawan Goyal, and Oliver Hellwig. 2020. Evaluating neural morphological taggers for Sanskrit. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 198–203, Online. Association for Computational Linguistics.

Oliver Hellwig. 2010. Dcs-the digital corpus of sanskrit. *Heidelberg (2010-2021). URL http://www.sanskrit-linguistics.org/dcs/index.php*.

Oliver Hellwig. 2015. Using recurrent neural networks for joint compound splitting and sandhi resolution in sanskrit. In *4th Biennial Workshop on Less-Resourced Languages*.

Oliver Hellwig and Sebastian Nehrdich. 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, Brussels, Belgium. Association for Computational Linguistics.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Amrith Krishna, Ashim Gupta, Deepak Garasangi, Pavankumar Satuluri, and Pawan Goyal. 2020a. Keep it surprisingly simple: A simple first order graph based parsing model for joint morphosyntactic parsing in Sanskrit. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4791–4797, Online. Association for Computational Linguistics.

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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2550–2561, Brussels, Belgium. Association for Computational Linguistics.

Amrith Krishna, Bishal Santra, Ashim Gupta, Pavankumar Satuluri, and Pawan Goyal. 2020b. A graph based framework for structured prediction tasks in sanskrit. *Computational Linguistics*, 46(4):1–63.

Amrith Krishna, Bishal Santra, Pavankumar Satuluri, Sasi Prasanth Bandaru, Bhumi Faldu, Yajuvendra Singh, and Pawan Goyal. 2016a. 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, Osaka, Japan. The COLING 2016 Organizing Committee.

Amrith Krishna, Pavan Kumar Satuluri, and Pawan Goyal. 2017. A dataset for Sanskrit word segmentation. In *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 105–114, Vancouver, Canada. Association for Computational Linguistics.

Amrith Krishna, Pavankumar Satuluri, Shubham Sharma, Apurv Kumar, and Pawan Goyal. 2016b. Compound type identification in Sanskrit: What roles do the corpus and grammar play? In *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*, pages 1–10, Osaka, Japan. The COLING 2016 Organizing Committee.

Sriram Krishnan, Amba Kulkarni, and Gérard Huet. 2020. Validation and normalization of dcs corpus using sanskrit heritage tools to build a tagged gold corpus.

Rishabh Kumar, Devaraja Adiga, Rishav Ranjan, Amrith Krishna, Ganesh Ramakrishnan, Pawan Goyal, and Preethi Jyothi. 2022. Linguistically informed post-processing for asr error correction in sanskrit. *Proc. Interspeech 2022*, pages 2293–2297.

Xiaonan Li, Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2020. FLAT: Chinese NER using flat-lattice transformer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6836–6842, Online. Association for Computational Linguistics.

Vipul Mittal. 2010. Automatic Sanskrit segmentizer using finite state transducers. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 85–90, Uppsala, Sweden. Association for Computational Linguistics.

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

Vikas Reddy, Amrith Krishna, Vishnu Sharma, Prateek Gupta, Vineeth M R, and Pawan Goyal. 2018. Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Jivnesh Sandhan, Amrith Krishna, Pawan Goyal, and Laxmidhar Behera. 2019. Revisiting the role of feature engineering for compound type identification in Sanskrit. In *Proceedings of the 6th International Sanskrit Computational Linguistics Symposium*, pages 28–44, IIT Kharagpur, India. Association for Computational Linguistics.

Jivnesh Sandhan, Amrith Krishna, Ashim Gupta, Laxmidhar Behera, and Pawan Goyal. 2021. A little pretraining goes a long way: A case study on dependency parsing task for low-resource morphologically rich languages. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 111–120, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz 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.

Hang Yan, Bocao Deng, Xiaonan Li, and Xipeng Qiu. 2019. Tener: Adapting transformer encoder for named entity recognition.

Yue Zhang and Jie Yang. 2018. Chinese NER using lattice LSTM. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1554–1564, Melbourne, Australia. Association for Computational Linguistics.

A Appendix

Average run times: Table 5 shows the average training time in hours and inference time in milliseconds for all competing baselines. We find that pure engineering-based techniques (TENER, rcNN-SS) outperform lattice-structured architectures (Lattice-LSTM, Lattice-GNN, FLAT-Lattice) in terms of run time. When the inference times of TransLIST and TransLIST$_{\text{ngrams}}$ are compared, TransLIST takes longer owing to the PRCP module. It would be interesting to explore approaches to optimise the inference time of the PRCP module.

System	Train (Hours)	Test (ms)
TENER	4 H	7 ms
Lattice-LSTM	16 H	110 ms
Lattice-GNN	64 H	95 ms
FLAT-Lattice	5 H	14 ms
rcNN-SS	4 H	5 ms
Cliq-EBM	10.5 H	750 ms
TransLIST$_{\text{ngrams}}$	8 H	14ms
TransLIST	8 H	105 ms

Table 5: Average training time (in hours) and inference time (in milliseconds) for all the competing baselines.