Abstract

We describe our participation in the Word Segmentation and Morphological Parsing (WSMP) for Sanskrit hackathon. We approach the word segmentation task as a sequence labelling task by predicting edit operations from which segmentations are derived. We approach the morphological analysis task by predicting morphological tags and rules that transform inflected words into their corresponding stems. Also, we propose an end-to-end trainable pipeline model for joint segmentation and morphological analysis. Our model performed best in the joint segmentation and analysis subtask (80.018 F1 score) and performed second best in the individual subtasks (segmentation: 96.189 F1 score / analysis: 69.180 F1 score).

Finally, we analyse errors made by our models and suggest future work and possible improvements regarding data and evaluation.

Our code can be accessed from https://github.com/cicl-iscl/TueSan.

1 Introduction

This paper describes our efforts to solve the tasks proposed by the WSMP hackathon\(^1\) associated with Forum for Information Retrieval Evaluation (FIRE) 2021.\(^2\) The hackathon focuses on two Sanskrit NLP problems, namely word segmentation and morphological analysis. These tasks are precursors to downstream NLP tasks like dependency parsing. However, characteristics of the Sanskrit language make them non-trivial and even challenging on their own.

Sanskrit had a rich oral tradition, knowledge and wisdom were passed on through recitation long before the scriptures could be consolidated in written form. It is then not surprising that its orthography faithfully reflects the euphonic modifications made to reduce articulatory efforts during continuous speech or chanting, which requires stricter rhythm. This phonetic smoothing process over consecutive words, termed sandhi, comes at the cost of easy and unambiguous identification of word boundaries and word forms. Just by looking at the resulting phonetic stream, which can be written in arbitrary syllabic script or transliteration scheme, it is not enough to directly identify the sequence of words as there exist many lexically and phonetically valid parses.

Consider the following simple example taken from Guhe (2008), transliterated from Devanāgarī script to IAST:\(^3\)

\[
\begin{align*}
(1) & \quad \text{kena pathā bhavānsakhyā sahāgacchat} \\
& \quad \text{kena pathā bhavān sakhyā saha āgacchat} \\
& \quad \text{kena pathā bhavān sakhyā saha āgacchat}
\end{align*}
\]

The italicised first line in (1) is the sandhied sequence, followed by two segmentations with their difference underlined. Note that the spaces (hiatus) in the original sentence do not fully correspond to word boundaries.

For someone with knowledge of the language, these are the first two segmentations that come into mind because a human can look at this sentence holistically and first determine the inflected verb hidden in the sandhied chunk sahāgacchat. To do so, this person must already have an idea what the sentence is about, combining knowledge of a lexicon, possible sound changes and a grammar. However, a machine can only process the sequence linearly and operate without such linguistic knowledge unless explicitly provided or learned through training. Therefore, determining where sandhi has occurred is already a challenge. But suppose we know for sahāgacchat, ā is the result of applying the sandhi rule of merging similar word-initial and -final vowels into its long form. Here it is still

\(^1\)https://sanskritpanini.github.io/
\(^2\)http://fire.irsres.in/fire/2021/home
\(^3\)International Alphabet of Sanskrit Transliteration
non-trivial to obtain the word forms because in the reverse process, ā can be split into any combination of ā and a. The possible positions where sandhi could have occurred and the different ways of splitting amount to a multitude of phonetically possible segmentations. Hence the word segmentation task is reformulated as identifying the semantically most valid segmentation for the given sentence (Krishna et al., 2017).

For simplicity, here we only discuss the semantically valid splits for the long vowel ā, in particular, ā + a and a + a. From the segmentations we obtain the following morphological analyses:

(2) a. ... sakhyā saha agacchat
   a-āgam-t
   PST-come-3SG.PST.PAR
   'came with (a male/female) friend'

b. ... sakhyā saha agacchat
   a-gam-t
   PST-go-3SG.PST.PAR
   'went with (a male/female) friend'

Without any context, 2a and 2b are equally valid. Example 2 also shows that if word segmentation and morphological parsing are performed separately and sequentially, errors from segmentation will propagate into analysis.

Moreover, as Krishna et al. (2020) pointed out, Sanskrit is a language characterised by a high degree of syncretism and homonymy, therefore the morphological analysis for a given surface word form is usually not unique. The inflected noun sakhyā in our example sentence, for instance, obtains the following analyses:

(3) a. ... sakhyā saha ... sakhi-ā
   friend.M-INS.SG
   'with (a male) friend'

b. ... sakhyā saha ... sakhi-ā
   friend.F-INS.SG
   'with (a female) friend'

Since the same word is analysed in 3a and 3b with two stems, sakhi and sakhi, we have a case of homonymy. A more extreme case of homonymy is demonstrated by sīṣyāḥ, which is the nominative plural of the masculine noun sīṣya (meaning student) and also the second person singular optative parasmaipada of the verb sās (meaning to command). From the feminine noun stem nau (meaning ship, vessel), nāvah is the resulting form for: (a) ablative singular (b) genitive singular (c) nominative plural (d) accusative plural and (e) vocative plural, hence an example of syncretism.

Notice that the above morphological analyses consider just the individual word without contextual information. It does not realise that bhavān is actually a form of the honorific pronoun bhavat in the second person, so the verb in 2 is actually intended for the second person, conforming to the inflection paradigm for the third person. This may also pose challenges to downstream tasks if morphological tags were to be used as inputs.

For the simple example sentence in 1, ambiguity at both segmentation and analysis level already produces four distinct readings. More ambiguities are expected as the length of the input sentence increases. Using this toy example, we also showed that neither word segmentation nor morphological analysis is to be seen as separated from semantics. Although generation of a sentence is deterministic with a given sequence of stems, their associated morphological properties and a set of sandhi rules, the reverse is non-deterministic at every step. This observation probably encourages postponing decisions at task-level and combining more contextual information to jointly resolve ambiguities.

The rest of this paper is structured as follows: Section 2 gives general descriptions of the three tasks of the hackathon, Section 3 introduces the dataset provided, Section 4 introduces our experiment with transliteration schemes, Section 5 describes our approaches to each of the three tasks in terms of data preprocessing and model structure, Section 6 outlines our training and tuning process and the results we obtained. Also, Section 6 contains an extensive error analysis. Finally, in Section 7 we discuss our findings and conclude this paper.

2 Task Descriptions

We briefly introduce the three tasks here, for more detailed descriptions please visit the official

---

4Glossings used here deviates slightly from the Leipzig Glossing Rules, accessible from: https://www.eva.mpg.de/lingua/resources/glossing-rules.php. PAR refers to Parasmaipada

5See translations in Example 2
2.1 Word segmentation

For the first task, a sandhied sentence in the form of a single string is given as input, with unambiguous word boundaries marked by spaces. The desired output is a list of words in the sentence in *unsandhied* form, as shown in Ex. 1, taken from the training dataset. Different from other languages where tokenization is also non-trivial, like Japanese, in Sanskrit, another layer of complexity is added to retrieve the correct segmentation: Tokenized outputs also have to undergo a reverse process of the euphonic changes. The task, therefore, is two-fold: finding correct word boundaries and retrieving unsandhied word forms.

**Input:** bhavati cātra
**Output:** ['bhavati', 'ca', 'atra']

Example 1: Input-output pair for task 1

2.2 Morphological parsing

In the second task, the segmentation ground-truth is given as input. Our goal is to predict the stem and morphological tag for each word. The morphological tags used in the provided dataset follow the standards of the Sanskrit Heritage Reader (Goyal and Huet, 2016), where depending on the grammatical category of a word, different numbers of labels are combined to form a single tag string. A finite verb receives a morphological tag with at most six labels, corresponding to 1) mood 2) tense 3) conjugation class 4) voice 5) number and 6) person, while a noun receives a triple consisting of 1) case 2) number and 3) gender as tag. Other indeclinables (avyaya) receive their own POS tags as morphological tags.

**Input:** bhavati ca atra
**Output:** [('bhú', 'pr. [1] ac. sg. 3'), ('ca', 'ca', 'conj.'), ('atra', 'atra', 'adv.')]  

Example 2: Input-output pair for task 2

2.3 Combined segmentation and analysis

The third task combines task 1 and task 2: Given a sandhied sentence as a single string, we predict the segmented unsandhied words with their full morphological analysis: stem and morphological tag. The output is a list of triples (word, stem, tag) for each sentence.

**Input:** bhavati cātra
**Output:** [('bhavati', 'bhú', 'pr. [1] ac. sg. 3'), ('ca', 'ca', 'conj.'), ('atra', 'atra', 'adv.')]  

Example 3: Input-output pair for task 3

3 Dataset

The WSMP dataset provided by the task organizers is separated into training and development sets. The organizers do not release the test set. The training set consists of 90 000 sentences and the development set contains 10 332 sentences. Ground truths for all three tasks are provided alongside the input sentences. Apart from the main datasets, each sentence is also paired with a graphML file which stores all valid parses of the input sentence generated by Sanskrit Heritage Reader (Goyal and Huet, 2016). These sentences, as described by the organizers, are taken from the Digital Corpus of Sanskrit (DCS) (Hellwig, 2010–2021), a corpus for digitalised Sanskrit manuscripts of various domains of knowledge and styles.

4 Transliteration

There have been opposing views on the effect of using different unit representations in language modeling. Huet (2013) argues that transliteration is *completely irrelevant* to Sanskrit text processing and the segmentation problem, while Adiga et al. (2021), in a pioneer large-scale study of Sanskrit ASR (Automatic Speech Recognition), favors *phonetic based graphemic representations* over scripts without one-to-one correspondence to phonemes.

For our own investigation, we experimented with two transliteration schemes for all three tasks to see whether the use of different transliteration schemes has influence over model performance. The first one is the given input in IAST, the other is the internal representation \(^9\) used by Oliver Hellwig for

\(^6\)https://sanskritpanini.github.io/tasks.html

\(^7\)cf. Krishna et al. (2017) for structural specifications of the graphML files

\(^8\)Accessible from: https://github.com/OliverHellwig/sanskrit/tree/master/dcs/data

\(^9\)Character mapping from IAST is defined in: https://github.com/OliverHellwig/sanskrit/blob/master/papers/2018emnlp/code/data_loader.py
the DCS, which is similar to SLP1. Different from IAST, which uses two Unicode characters to represent diphthongs and aspirated consonants, the second transliteration scheme uses one Unicode character, establishing one-to-one correspondence between transliterated character and phonemic unit.

5 Method

5.1 Word segmentation

We formulate the word segmentation task as a sequence labelling problem: Given a character sequence \( s = c_1c_2 \ldots c_n \), predict exactly 1 edit operation for each character. An edit operation locally modifies the input sequence. By applying the predicted edit operations to each character of the source sentence, we arrive at the predicted sequence \( s' = c'_1c'_2 \ldots c'_n \). Then, we retrieve the predicted segmentation of the input sentence by splitting \( s' \) at predicted spaces. In this way, word boundaries and unsandhied word form are determined at the same time.

Data preprocessing Our data preprocessing has to achieve two goals: (a) extract the necessary edit operations and (b) align the edit operations with the characters.

First, we remove all datapoints where the source sentence is longer than the target sequences, as these represent corrupted datapoints. This affects \( \approx 1\% \) of the dataset. Then, we align each source sequence with the respective target sentence by minimising Levenshtein distance. The aligned sequences decompose into aligned and non-aligned chunks. Aligned chunks are transformed into each other by simple copy operations. Non-overlapping chunks are kept as edit rules, which are represented as tuples of ngrams \((\text{SOURCE}, \text{TARGET})\). These rules denote what source ngram is transformed into which target ngram. Each source character in \text{SOURCE} gets the corresponding rule as label. For inference, we contract consecutively predicted equal rules. To ensure \text{SOURCE} contains at least 1 character, we concatenate \text{TARGET} of rules with empty \text{SOURCE} to the \text{TARGET} of the left rule. Finally, we introduce a special rule for the case where we simply insert a space after a character. \text{COPY} and \text{INSERT}-SPACE rules abstract away from the concrete character, thereby reducing the total number of rules. The sequences of edit operations generated for the training set are used as ground-truth labels to train our model.

In total, we extract 166 rules from the dataset without transliteration and 187 rules when transliterating Sanskrit inputs. An example of the result of our data preprocessing is in Tab. 4.

Model For predicting the label sequence, we use a model that is very similar to the model proposed by Hellwig and Nehrdich (2018): First, we extract ngram features from the input character embeddings by 1d-convolutions with filter sizes 2 to 8. Inputs are 0-padded to ensure that sequence lengths remain the same after convolution. Then, we apply a residual block to the ngram features. We concatenate all ngram features and project them to a lower dimension. Finally, we run a 2-layer BiLSTM with residual connection on the sequence and predict the labels for each character. A visualisation of the model is in Fig 1.

5.2 Morphological parsing

We also formulate the morphological parsing task as a sequence-labeling task with labels being rules extracted from the train data. These rules describe how to transform a word into the word’s stem. Mor-

\[ e_1 \quad e_2 \quad e_3 \quad \ldots \]

\[ c_1 \quad c_2 \quad c_3 \quad \ldots \]

Figure 1: Visualisation of our word segmentation model (Sec. 5.1): Character convolutions of different filter sizes are followed by downsampling and BiLSTM. Finally, rules are predicted for all input characters. Also note the residual connections.
For example, predict Sandhi rules at word boundaries, if necessary. “C” means copy and “I” means insert space (after character). For example, we segment “dināntara” by predicting the Sandhi rule (ā → a a) for the first “ā”.

Dataset preprocessing To extract rules, we process the (WORD, STEM) pairs individually. We align WORD and STEM by matching their longest common infix. Then, for both WORD and STEM, we extract a prefix and a suffix. The prefix corresponds to the substring preceding the matched infix. The suffix corresponds to the substring following the matched infix. Both prefix and suffix can be empty. The final rule is thus a quadruple (prefix WORD, suffix WORD, prefix STEM, suffix STEM). Given a rule and a word, we can apply the rule if the word starts with prefix WORD and ends with suffix WORD. If this is the case, we can generate the predicted stem by substituting prefix WORD with prefix STEM and substituting suffix WORD with suffix STEM.

An example of alignment and extraction of suffix and prefix is in Fig. 2.

Model For predicting rules (and optionally morphological tags), we first encode each token separately by character convolutions with filter sizes 2 to 6. We 0-pad tokens so that their lengths remain unchanged after convolution. The extracted features are combined into a single token embedding by feature-wise max pooling. We also experimented with using the concatenated final hidden states of a BiLSTM run on the character features as token embeddings. However, we found this to perform worse.

After having thus calculated an embedding for each token, we run a sentence level BiLSTM on the token embeddings. The reason is that we want to include context information. However, we include a residual connection by adding each token embedding to the respective BiLSTM output. In this way, the resulting token embeddings encode both subword and contextual information. Rules and morphological tags are predicted from these contextualised token embeddings.

A visualisation of the model is in Fig 3. Note that our model can be seen as a modified version of ELMo (Peters et al., 2018).

For inference, we check which rules are applicable to a given token. A rule is applicable to a token iff the token starts with prefix WORD and ends with suffix WORD. If no rules are applicable, we predict the unaltered word.

5.3 Combined segmentation and analysis

We propose a pipeline approach to combined segmentation and analysis that is end-to-end trainable for both predicted segmentations and predicted stems and morphological tags: First, we apply the same model as for word segmentation to predict the words (see Sec. 5.1). Then, we convert the predicted labels into token boundaries (i, j), where i
Figure 3: Visualisation of our morphological parsing model (Sec. 5.2): We encode each word by character convolutions of different filter sizes, which are max-pooled to arrive at embeddings for each word. Then, we capture context information by running a BiLSTM on the token embeddings. Finally, rules are predicted for all input words. Note that residual connections are not shown.

denotes the index of the first character of the corresponding word in the unsegmented input string and j denotes the index of the last character.

For each predicted token, we calculate a token embedding by feature-wise max pooling the corresponding outputs of the BiLSTM model as described in Sec. 5.1. The corresponding outputs are all timesteps of the input sequence starting with index i and ending with index j. Finally, we predict stem rules and morphological tags from the token embeddings.

In this case, we always predict stems and morphological tags separately, because this yields better performance in our setup. As noted in Sec. 5.2, stem rules and morphological tags can be predicted jointly.

Note in particular that we need to predict the segmentation first in order to proceed with the calculation of token embeddings. Therefore, our approach implements a pipeline. However, by reusing hidden states of the BiLSTM, the predictions for stems and morphological tags are differentiable w.r.t. the input characters. Therefore, the whole model remains end-to-end trainable.

Dataset preprocessing is the same as for task 1 (see Sec. 5.1) and task 2 (see Sec. 5.2).

6 Results

6.1 Training

All models are trained by minimising the cross-entropy-loss between prediction probabilities and one-hot encoded target labels. We use a vanilla SGD optimizer without weight decay or momentum. The learning rate is set according to a one cycle learning rate scheduler (Smith and Topin, 2019). For our models, we found the combination of SGD and one cycle learning rate scheduler to perform better than Adam (Kingma and Ba, 2015). All models are implemented in PyTorch (Paszke et al., 2019).

6.2 Hyperparameter tuning

In order to find the best configuration for our models, we perform extensive hyperparameter tuning using the Ray Tune library (Liaw et al., 2018). An overview over the tuned hyperparameters and their possible values is in the appendix.

We make the following observations:

- Larger embedding sizes always work better. This is surprising, as we even see improvements when the embedding dimension is already larger than the vocabulary size.
- Wider models perform better in all evaluated cases. However, this is not surprising given the abundance of training data. We do not evaluate whether deeper models also perform better.
- The performance of our models already peaks after 15 epochs. We do not observe improved performance when training longer.
- Both the use of transliteration and larger vs. smaller batch sizes do not show a clear trend as to which is better. Instead, this decision is dependent on the particular subtask.
- For predicting stem rules for morphological parsing, including all rules generated from the training data (i.e. not discarding infrequent rules) and predicting morphological stems separately from tags yields better performance.

6.3 Hackathon results

Our approaches obtained the second place at the FIRE 2021 hackathon in both the word segment-
tation track and the morphological parsing track. Our pipeline model described in Sec. 5.3 obtained the first place in the combined segmentation and analysis track. Since the test set is not released, we report official results from the hackathon’s codalab page. Results for all three tracks are in Tab. 5a, Tab. 5b, and Tab. 5c.

Generally, the top results are very close to each other. Due to the black-box nature of the evaluation, we cannot assess whether all differences are significant. However, we observe that our end-to-end trainable pipeline yields best results for the third track, despite our models do not yield best performance on the individual tasks. One possible conclusion is that end-to-end training in this case makes the pipeline more robust. However, the pipeline can still benefit from our distinct approaches to the two individual tasks.

6.4 Error analysis

Since the test set is not released, we evaluate our model performance on dev data. The scoring script we use is taken from the starting kit provided by the task organisers, which differs from the actual script used for the hackathon.

Segmentation errors For word segmentation we report errors of the following types:

• Where multiple splits are possible, the model does not pick the correct split. This includes not splitting where it should and oversplitting with the INSERT-SPACE operation, which is the most common error type. Errors of this type are expected since not all ambiguities can be resolved.

• The edit operations extracted are themselves problematic due to incorrect ground-truths. For instance, we extracted rules that map single characters to the empty string, which sometimes causes weird vanishing characters in the prediction. Upon inspection, we found that some of the ground-truths are marked with this problem, so we may have extracted such rules from corrupt datapoints in the training data.

• We observe false errors due to incorrect ground-truths. The gold segmentations can sometimes be incomplete, either characters or entire chunks of the input sentence are missing.

Analysis errors For predicting stems and morphological tagging, our T2 model evaluated on dev set produces 65.51% full match for the sentences. Out of the 68 094 words to be predicted, 92.51% are fully correctly predicted. And 87.69% of the erroneous cases are partially correct, which means either the stem or the tag is correctly predicted. Note that the tag is a single string combining multiple morphological properties and the evaluation we choose to report here uses strict comparison. The hackathon, however, uses a different way to compare the strings, which will be discussed later in 6.5.

For morphological tagging, our predictions are wrong most of the time only by one or two subtags. For instance the model predicts “m. sg. acc.” instead of “n. sg. acc.” for the form deham (meaning body), only the gender is incorrect. Since gender is an inherent property of the noun, and according to the declension paradigm of masculine and neuter nouns ending in short vowel -a, both analyses are valid without contextual information or a lexicon. Similarly, for verbs, the conjugate class is associated with the root, we observe cases where only the class is incorrectly predicted. For example the verbs in one particular conjugation class, the tenth, are often confused with causatives. They are sometimes indistinguishable for human as well, hence our model predicted “ca. opt. ac. sg. 3” where the correct analysis would be “opt. [10] ac. sg. 3”.

For recovering word stems, the model does not perform well when sound changes are involved, in particular the retroflexion of n and s to ň and ň, or one induced by an insertion of a past tense marker a between a preverb and the stem, like for abhidhāv-, when a is inserted between the preverb abhi- and the stem dhāv, it changes to abhyadhāv-, our model prediction fails to remove the tense marker a and restore i from y.

6.5 Reevaluating evaluation

Since the hackathon is essentially a competition, it is necessary to look at how the scores are calculated, a more fine-grained way of evaluation may change the final results depending on the models used by the teams. As mentioned before, the hackathon
| T1 | F1 Score | T1 Precision | T1 Recall | T1 Task Score |
|----|----------|--------------|-----------|---------------|
| 1  | hrishikesht | 97.478 (1)   | 97.580 (1) | 97.439 (1)    |
| 2  | Ours      | 96.189 (2)   | 96.439 (2) | 96.061 (2)    |
| 3  | jivnesh   | 95.436 (3)   | 95.793 (3) | 95.237 (3)    |
| 4  | maheshak  | 82.752 (4)   | 82.223 (4) | 83.646 (4)    |
| 5  | Asha      | 51.470 (5)   | 52.177 (5) | 51.525 (5)    |

(a) Results for T1

| T2 | F1-Score | T2 Precision | T2 Recall | T2 Task Score |
|----|----------|--------------|-----------|---------------|
| 1  | jivnesh  | 69.327 (1)   | 60.410 (1) | 85.467 (1)    |
| 2  | Ours     | 69.180 (2)   | 60.259 (2) | 85.317 (2)    |
| 3  | hrishikesht | 42.644 (3)   | 52.513 (3) | 39.234 (5)    |
| 4  | maheshak | 42.644 (3)   | 39.234 (4) | 52.513 (3)    |
| 5  | Asha     | 38.190 (4)   | 33.187 (5) | 47.310 (4)    |
| 6  | Manas_P_P | 27.950 (5)   | 24.017 (6) | 35.384 (6)    |

(b) Results for T2

| T3 | F1 Score | T3 Precision | T3 Recall | T3 Task Score |
|----|----------|--------------|-----------|---------------|
| 1  | Ours     | 80.018 (1)   | 80.252 (1) | 79.906 (1)    |
| 2  | jivnesh  | 78.762 (2)   | 79.065 (2) | 78.591 (2)    |
| 3  | hrishikesht | 63.657 (3)   | 63.603 (3) | 63.781 (3)    |
| 4  | maheshak | 24.026 (4)   | 16.579 (4) | 46.403 (4)    |

(c) Results for T3

Uses a different way to compare strings. Instead of checking for string equality, they used a counter-based method to count the characters that occurred in both the ground-truth and the model prediction. Although this measure seems more lenient, it is not very accurate and can create more confusion.

To illustrate the point, consider the made-up example of “opt. [1] ac. sg. 3” and “opt. [3] ac. sg. 1”, according to their scoring script, this prediction is considered fully correct, when the third person is predicted as first person. The contrary, “ca. opt. ac. sg. 3” and “opt. [10] ac. sg. 3” are considered different when they are actually more similar to each other than the previous pair. For nouns, consider “f. du. abl.” and “m. pl. dat.”, none of the subtags are correctly predicted, but 8 out of 11 characters are counted as the same, merely because there happen to be overlapping characters in different morphological categories. This would imply that locative (loc.) and vocative (voc.) are considered more similar than nominative (nom.) and vocative (voc.), while the truth is the exact opposite. In this regard, this way of evaluation is not very informative.

When models achieve similar high scores, it is also interesting to know on what they fail and whether model performance is consistent over the “difficult” cases. To better understand individual model performance, we propose to subdivide the test dataset into multiple sets, each sharing some linguistically-oriented properties. For instance, an expert in Sanskrit can compose lists of hard-to-analyse forms categorised into forms with reduction, ambiguous forms, idiosyncratic forms, forms with nontransparent vowel gradation etc. Subsets of the test set can be automatically created from there on. It would also be useful to have a set testing the models on out-of-vocabulary (OOV) forms of known roots and unseen roots. In doing so, we have a more fine-grained qualitative evaluation of the models. The results would be more informative especially when model structures are not specified in the competition.

7 Discussion and Conclusion

In this paper, we have presented our approaches to Sanskrit word segmentation and morphological analysis. Our main contributions are proposing to
predict rules to generate stems from inflected forms and proposing an end-to-end trainable but pipelined model for joint segmentation and morphological parsing. Since our rules are applicable to any word with matching prefix and suffix, they can be applied to analyse OOV forms, transferring the inflection paradigm we learned to potentially unseen roots. An analogous idea, “data hallucination” (Anastasopoulos and Neubig, 2019), is already used for data augmentation in learning morphological inflection in a low-resource setting, where the stem region of a word is identified via alignment and characters in that region are replaced randomly. While they are hallucinating data, we are essentially doing root/stem interpolation.

Our approaches performed well in the WSMP hackathon associated with FIRE 2021. This shows that these ideas constitute interesting avenues which can be further explored in the future.

Predicting rules for segmentation and stemming offers the possibility of including linguistic information both into preprocessing (e.g. which rules to consider, the format of the rules) and decoding (often, there are only few rules that apply in to certain instance, which should be narrowed down as much as possible). This is confirmed by our error analysis, where we spot erroneous rules derived from corrupted data and wrong predictions that could be corrected by a dictionary lookup (e.g. wrongly predicted gender).

One line of future work therefore could enhance our approach by including more explicit linguistic information, for example by consulting a dictionary when decoding to rule out stemming rules or tags. Also, we do not make use of the segmentation lattices representing all valid segmentations. These could be used to improve alignments for extracting segmentation edit operations.

Our error analysis also shows that the models’ performance still can be improved. Therefore, investigating the impact of pretrained embeddings and better suited model architectures would be useful.

Finally, we discussed problems with the data quality and evaluation that we think could help improve NLP processing of Sanskrit in the future.

Acknowledgements
We thank Çağrı Çöltekin for support and helpful discussions. We also thank the NVIDIA corporation for donating GPUs to Çağrı, which we used to run our experiments. We thank Frank Köhler for his work in teaching Sanskrit. Finally, we extend our thank to the organisers of the FIRE 2021 hackathon for organising the event and providing data.

References
Devaraja Adiga, Rishabh Kumar, Amrith Krishna, Preethi Jyothi, Ganesh Ramakrishnan, and Pawan Goyal. 2021. Automatic Speech Recognition in Sanskrit: A New Speech Corpus and Modelling Insights.

Antonios Anastasopoulos and Graham Neubig. 2019. Pushing the Limits of Low-Resource Morphological Inflection. 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 984–996, Hong Kong, China. Association for Computational Linguistics.

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

E. Guhe. 2008. Einleitung in das klassische Sanskrit: ein Lehrbuch mit Übungen. Elementa Linguarum Orientis. Harrassowitz.

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. 2010–2021. The Digital Corpus of Sanskrit (DCS).

Oliver Hellwig and Sebastian Nehrdich. 2018. Sanskrit Word Segmentation Using Character-level Recursive 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.

Gérard Huet. 2013. Sanskrit Segmentation.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Amrith Krishna, Bishal Santra, Ashim Gupta, Pavankumar Satuluri, and Pawan Goyal. 2020. A Graph-Based Framework for Structured Prediction Tasks in Sanskrit. Computational Linguistics, 46(4):785–845.
As stated in Sec. 6.2, we perform extensive hyperparameter tuning to find good configurations for our models. The tuned hyperparameters and corresponding values are in Tab. 6.

Due to time and resource restrictions, we only evaluate hyperparameters that seem either interesting from a theoretical perspective (e.g. whether to use transliterated inputs) or seemed responsible for major improvements in performance when doing preliminary experiments. Furthermore, we optimised hyperparameters for task 1 first, then for task 2, and finally for task 3. Thereby, we used insights gained from previous optimisation studies for choosing the hyperparameters to tune for the current task. For example, we noticed that 15 epochs of training are best for task 1 and task 2. Therefore, we did not tune the number of epochs for task 3.

Also due to time and resource restrictions, we did not perform full grid search. Instead, we randomly sampled 20 hyperparameter configurations for task 1 and task 3, and 30 hyperparameter configurations for task 2, using functionality provided by the Ray Tune library.

In each case, we train the model on the train data and evaluate it on the development data (see Sec. 3). In Sec. 6.2, we report our main findings.

### Appendix

#### A.1 Hyperparameters

As stated in Sec. 6.2, we perform extensive hyperparameter tuning to find good configurations for our models. The tuned hyperparameters and corresponding values are in Tab. 6.

Due to time and resource restrictions, we only evaluate hyperparameters that seem either interesting from a theoretical perspective (e.g. whether to use transliterated inputs) or seemed responsible for major improvements in performance when doing preliminary experiments. Furthermore, we optimised hyperparameters for task 1 first, then for task 2, and finally for task 3. Thereby, we used insights gained from previous optimisation studies for choosing the hyperparameters to tune for the current task. For example, we noticed that 15 epochs of training are best for task 1 and task 2. Therefore, we did not tune the number of epochs for task 3.

Also due to time and resource restrictions, we did not perform full grid search. Instead, we randomly sampled 20 hyperparameter configurations for task 1 and task 3, and 30 hyperparameter configurations for task 2, using functionality provided by the Ray Tune library.

In each case, we train the model on the train data and evaluate it on the development data (see Sec. 3). In Sec. 6.2, we report our main findings.

### Table 6: Tuned hyperparameters and corresponding values for all tasks.

| Hyperparameter     | Values             | Best value |
|--------------------|--------------------|------------|
| Batch size         | {16, 64}           | 16         |
| Epochs             | {15, 20, 25}       | 15         |
| Dropout            | {0.0, 0.1}         | 0.1        |
| Use LSTM           | {TRUE, FALSE}      | TRUE       |
| Hidden dim         | {128, 256, 512}    | 512        |
| Embedding dim      | {32, 64, 128}      | 128        |
| Transliteration    | {TRUE, FALSE}      | TRUE       |

(a) Hyperparameters for T1

| Hyperparameter     | Values             | Best value |
|--------------------|--------------------|------------|
| Epochs             | {15, 20}           | 15         |
| Hidden dim         | {256, 512}         | 512        |
| Embedding dim      | {32, 64, 128}      | 128        |
| Transliteration    | {TRUE, FALSE}      | FALSE      |
| Rule freq. cutoff  | {1, 5, 50}         | 1          |
| Char2Token         | {LSTM, MAX}        | MAX        |
| Tag rules          | {TRUE, FALSE}      | FALSE      |

(b) Hyperparameters for T2

| Hyperparameter     | Values             | Best value |
|--------------------|--------------------|------------|
| Batch size         | {16, 64}           | 16         |
| Hidden dim         | {256, 512}         | 512        |
| Dropout            | {0.0, 0.1}         | 0.1        |
| Transliteration    | {TRUE, FALSE}      | TRUE       |
| Char2Token         | {LSTM, MAX}        | MAX        |

(c) Hyperparameters for T3