Accessible Sanskrit: A Cascading System for Text Analysis and Dictionary Access

Giacomo De Luca
University of Tor Vergata, Via Cracovia 50, 00133 Roma, Italy
University of Tuscia, Via Santa Maria in Gradi 4, 01100 Viterbo, Italy
giacomo.deluca@unitus.it

Abstract

Sanskrit text processing presents unique computational challenges due to its complex morphology, frequent compound formation, and the phenomenon of Sandhi. While several approaches to Sanskrit word segmentation exist, the field lacks integrated tools that make texts accessible while maintaining high accuracy. We present a hybrid approach combining rule-based and statistical methods that achieves reliable Sanskrit text analysis through a cascade mechanism in which a deterministic matching using inflection tables is used for simple cases and statistical approaches are used for the more complex ones. The goal of the system is to provide automatic text annotation and inflected dictionary search, returning for each word root forms, comprehensive grammatical analysis, inflection tables, and dictionary entries from multiple sources. The system is evaluated on 300 randomly selected compounds from the GRETIL corpus across different length categories and maintains 90% accuracy regardless of compound length, with 91% accuracy on the 40+ characters long compounds. The approach is also tested on the complete text of the Yoga Sūtra, demonstrating 96% accuracy in the practical use case. This approach is implemented both as an open-source Python library and a web application, making Sanskrit text analysis accessible to scholars and interested readers while retaining state of the art accuracy.

1 Introduction

Sanskrit, additionally to the difficulties shared with other Morphologically Rich Languages (MRL) (Tsarfaty et al., 2020), presents the unique computationally challenge of Sandhi. Sandhi is defined in (Matthews, 2014) as the written modification and fusion of sounds at or across the boundaries of grammatical units and is used to represent words exactly as they will be pronounced. While the Sandhi application rules are deterministic, the parsing rules are sometimes not (Hellwig and Nehrdich, 2018). The Sandhi phenomenon makes Sanskrit inherently hard to parse for Large Language Models (LLM): the same nominative singular "yogaḥ", may appear as: "yogaś", "yoga", or, as "yogā", when merged with the initial 'a' of the next word. In this last worst case scenario, the word is indistinguishable with the nominative plural, and can only be parsed looking at the context. Without pre-splitting of Sandhi and compounds, the model has to learn multiple representations of the same words in an already scarcely digitalized literature. The ambiguity generated by the multiple parsing solutions of compounds and word blocks agglutinated by Sandhi were known since antiquity: for teaching purpose, alongside the Veda we find the *Padapatha*: a didactic version in which the words are restored to the non morphed grammatical version (Pillai, 1941). If the challenge of parsing created such interpretive complexity that multiple versions of the same poetic text emerged, why was this difficulty deliberately preserved rather than simplified? Before delving in how current computational approaches try to handle this difficulty, it is important to understand the historical reasons for this peculiar phenomenon. Sanskrit, whose name suggests a 'well made' language, is not a naturally arisen language, but a highly refined one which was formalized by the grammarians, starting from Pāṇini's seminal *Aṣṭādhyāyī* (Gillon, 2007) (Cardona, 1988). But what was the ideal leading to keep such a complexity in terms of reading? The reason becomes clear when reading the motivations for the study of language provided by Patañjali the grammarian: "preservation, modification, injunction, brevity and certainty" (Dasgupta, 1991). Those motivations are all related to the preservation of the Vedas and the performance of the sacrifices. From the correct execution of the sacrifice, soteriological immortality was believed to be attainable, as is stated in the

Rigveda(Jamison and Brereton, 2014)[1]. Patañjali presents multiple examples on how just a slight pronunciation error is enough to make the entire sacrifice backfires: the wrong pronunciation of the word "helayaḥ" is imputed as the reason for defeat of the Asuras (Dasgupta, 1991); again the mispelling of the word "Indra—śatru" changes is meaning from 'slayer of Indra' to 'slayed by Indra', resulting in the death of the son of Tvaṣṭṛ. From the correct execution of the sacrifices liberation was expected, and grammar was a mean, if not the primary mean to the right execution. It is easy to see how the performative aspect of language was prioritized over the communicative one. In consequence of this early focus, the language eloquently tells how it should be pronounced, not what words are underneath the pronunciation. To tackle this complexity, multiple approaches to the task of Sanskrit Word Splitting (SWP, splitting Sandhi and compounds) (Hellwig and Nehrdich, 2018) have been proposed, started from the pioneering grammatical based works of Huet (Huet, 2005, 2009). Several approaches to sandhi and compound parsing have been proposed, using both data driven approaches (Nehrdich et al., 2024) and mixed ones (Krishna et al., 2021). This development has not yet translated into improved accessibility to the original texts or the dictionaries. Without previous knowledge, is hard if not outright impossible to search in the dictionaries the words that appears in the texts: most words appear morphed by Sandhi or aggregated in compounds. The Digital Corpus of Sanskrit (Hellwig, 2010–2021) provides access on click to the stemmed and parsed text, with minimal entries derived from the Monnier Williams (MW) dictionary (Monier-Williams, 1899). Yet it works just on a manually annotated corpus of texts.

To improve the accessibility of the original texts, we propose an approach to Sanskrit word splitting that retrieves grammatical information and entries from multiple dictionaries. This approach allows for both text annotation and for a dictionary search allowing words to be queried as they appear in text, – inflected, compounded and morphed by Sandhi. This approach has been implemented as an open source Python library which includes a REST API built using Flask and a web application, which is accessible at `https://www.sanskritvoyager.com`. As it can be seen in Figure 1a, the application allows access to the text of the GRETIL[2] library. Words throughout the text become clickable, allowing users to access grammatical analysis and dictionary entries with a single click. Alternatively, text can be provided by the user, either by pasting or typing, receiving the same on demand interactive analysis.

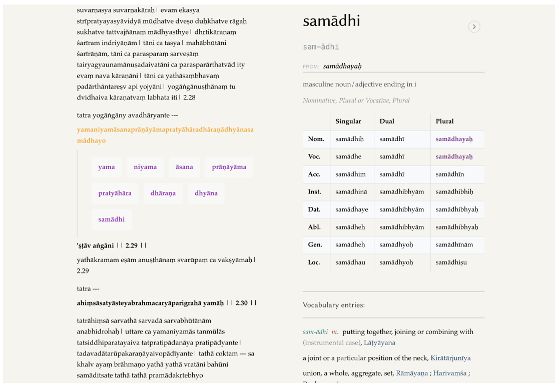
Alternatively, the web application can be used as a more accessible engine to query Sanskrit dictionaries, allowing for inflected form search and multi-dictionary lookup. To check for the capabilities of the current approach to handle complex compounds and Sandhi-blocks, the underlying system has been tested with a random selection of 300 compounds of various length from the GRETIL corpus, and with a practical use case of annotating the entire Yoga Sūtra. The system performed effectively in both tasks, maintaining an accuracy of 92% for all the compounds categories, which increases to an accuracy of the 96% for the Yoga Sūtra task.

2 Previous Literature

Sanskrit word processing has seen considerable development over the past few decades, moving from rule-based systems to more data-driven approaches, though it continues to present unique challenges due to the language's complex morphology and phonology. Early approaches often combined Pāṇini's phonetic and morphological rules with lexical resources, using either formal methods or statistical approaches(Huet, 2005) (Huet, 2009). Finite state transducers were employed for automatic segmentation, with the aim of splitting a Sanskrit string into its constituent words (Mittal, 2010). However, a major hurdle is the availability of annotated datasets, which are crucial for training data driven models, particularly when compared to the resources available for other languages. The Digital Corpus of Sanskrit (DCS) has been a significant effort, providing over 650,000 lexically and morphologically tagged sentences. Datasets for word segmentation have also been created, though these often come with their own limitations (Krishnan et al., 2024). The central challenge to Sanskrit computational linguistics remains the handling of sandhi, which obscures word boundaries due to the phonetic merging of words (Krishna et al., 2021). Recent work has explored neural sequence labeling tasks, using recurrent and convolutional neural net-

[1]VIII.48.3: "We have drunk the soma; we have become immortal; we have gone to the light; we have found the gods"

[2]https://gretil.sub.uni-goettingen.de/gretil.html


suvarṇasya suvarṇakāraḥ। evam ekasya strīpratyayasyāvidyā mūḍhatve dveṣo duḥkhatve rāgaḥ sukhatve tattvajñānaṃ mādhyasthye। dhṛtikāraṇaṃ śarīram indriyāṇām। tāni ca tasya। mahābhūtāni śarīrāṇām, tāni ca parasparaṃ sarveṣāṃ tairyagyaunamānuṣadaivatāni ca parasparārthatvād ity evaṃ nava kāraṇāni। tāni ca yathāsaṃbhavaṃ padārthāntareṣv api yojyāni। yogāṅgānuṣṭhānaṃ tu dvidhaiva kāraṇatvaṃ labhata iti। 2.28

tatra yogāṅgāny avadhāryante ---

yamaniyamāsanaprāṇāyāmapratyāhāradhāraṇādhyānasamādhayo

yama niyama āsana prāṇāyāma

pratyāhāra dhāraṇa dhyāna

samādhi

'ṣṭāv aṅgāni ।। 2.29 ।।

yathākramam eṣām anuṣṭhānaṃ svarūpaṃ ca vakṣyāmaḥ। 2.29

tatra ---

ahiṃsāsatyāsteyabrahmacaryāparigrahā yamāḥ ।। 2.30 ।।

tatrāhiṃsā sarvathā sarvadā sarvabhūtānām anabhidrohaḥ। uttare ca yamaniyamās tanmūlās tatsiddhiparatayaiva tatpratipādanāya pratipādyante। tadavadātarūpakaraṇāyaivopādīyante। tathā coktam --- sa khalv ayaṃ brāhmaṇo yathā yathā vratāni bahūni samāditsate tathā tathā pramādakṛtebhyo

samādhi

sam–ādhi

FROM: *samādhayaḥ*

masculine noun / adjective ending in i

Nominative, Plural or Vocative, Plural

	Singular	Dual	Plural
Nom.	samādhiḥ	samādhī	**samādhayaḥ**
Voc.	samādhe	samādhī	**samādhayaḥ**
Acc.	samādhim	samādhī	samādhīn
Inst.	samādhinā	samādhibhyām	samādhibhiḥ
Dat.	samādhaye	samādhibhyām	samādhibhyaḥ
Abl.	samādheḥ	samādhibhyām	samādhibhyaḥ
Gen.	samādheḥ	samādhyoḥ	samādhīnām
Loc.	samādhau	samādhyoḥ	samādhiṣu

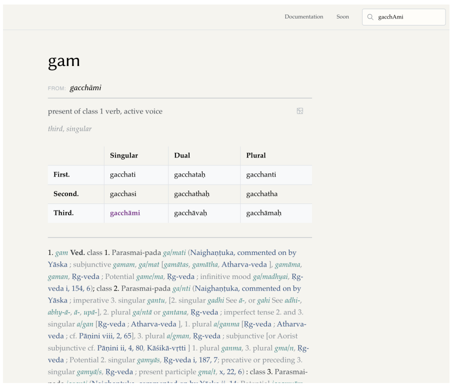
Vocabulary entries:

sam-ādhi *m.* putting together, joining or combining with (instrumental case), Lāṭyāyana

a joint or a particular position of the neck, Kirātārjunīya

union, a whole, aggregate, set, Rāmāyaṇa ; Harivaṃśa ;

(a) Enhanced text reader with parsing capability


Documentation Soon gacchAmi

gam

FROM: *gacchāmi*

present of class 1 verb, active voice

third, singular

	Singular	Dual	Plural
First.	gacchati	gacchataḥ	gacchanti
Second.	gacchasi	gacchathaḥ	gacchatha
Third.	**gacchāmi**	gacchāvaḥ	gacchāmaḥ

1. *gam* **Ved.** class 1. Parasmai-pada *ga/mati* (Naighaṇṭuka, commented on by Yāska ; subjunctive *gamam, ga/mat* [*gamātas, gamātha,* Atharva-veda], *gamāma, gaman,* Ṛg-veda ; Potential *game/ma,* Ṛg-veda ; infinitive mood *ga/madhyai,* Ṛg-veda i, 154, 6); class 2. Parasmai-pada *ga/nti* (Naighaṇṭuka, commented on by Yāska ; imperative 3. singular *gantu,* [2. singular *gadhi* See *ā-,* or *gahi* See *adhi-, abhy-ā-, ā-, upā-*], 2. plural *ga/ntā* or *gantana,* Ṛg-veda ; imperfect tense 2. and 3. singular *a/gan* [Ṛg-veda ; Atharva-veda], 1. plural *a/ganma* [Ṛg-veda ; Atharva-veda ; cf. Pāṇini viii, 2, 65], 3. plural *a/gman,* Ṛg-veda ; subjunctive [or Aorist subjunctive cf. Pāṇini ii, 4, 80, Kāśikā-vṛtti] 1. plural *ganma,* 3. plural *gma/n,* Ṛg-veda ; Potential 2. singular *gamyās,* Ṛg-veda i, 187, 7; precative or preceding 3. singular *gamyā/s,* Ṛg-veda ; present participle *gma/t,* x, 22, 6) : class 3. Parasmai-

(b) Dictionary interface for inflected forms

Figure 1: Web interface of the Sanskrit analysis system. (a) shows the dictionary lookup for inflected forms, and (b) displays the Yoga Sūtra text from GRETIL with commentary and on-demand word parsing.

works, and seq2seq models for word segmentation (Aralikatte et al., 2018). A graph based framework has been developed for structured prediction tasks including word segmentation and morphological parsing (Krishna et al., 2021). The goal of current computational approaches is developing an unified models capable of handling multiple tasks such as word segmentation, lemmatization and morphological tagging jointly. The approach proposed in (Nehrdich et al., 2024) promise to be handle all those tasks at once, and is the most significant breakthrough in Sanskrit computational linguistic in the recent years. In (Nehrdich et al., 2024) a Byt5 model (Xue et al., 2022) was trained to handle many downstream Sanskrit analysis tasks maintaining state of the art performance.

3 Methodology

Our methodology takes a fundamentally different approach from existing solutions by recognizing that not all Sanskrit words require complex processing. Instead of applying sophisticated analysis techniques universally, we implement a cascading system that starts with simple, deterministic methods and progressively moves to more complex approaches only when necessary. The foundation of this approach lies in the observation that many Sanskrit words can be analyzed through straightforward methods with complete certainty. For instance, regular inflected forms like "eṣyāmi" can be directly mapped to their root form through inflection tables, – in this case identifying it as the third person future of the verb "i" (to go). Common Sandhi cases follow predictable patterns: "yogaś" can be restored to its base form "yogaḥ" through simple substitution rules. Additionally, frequently used inflected forms such as "yogena" (the instrumental case of "yoga") often appear in dictionaries as standalone entries, allowing direct lookup without complex analysis. Our system implements this insight through a three tiered processing pipeline, which is shown in the flowchart at Figure 2. The first tier employs computationally inexpensive methods: dictionary lookup and basic substitution rules. This provides deterministic results for a significant portion of Sanskrit vocabulary. When these methods fail to produce a result, the system employs a statistical approach. The quality of this result is evaluated through a scoring system. If the confidence score falls below a predetermined threshold, the system tries again with a quasi brute force compound splitter that tries all possible combination using. The system then retains the highest scoring result from the second or third approach. Finally, for each recovered entry, the system retrieves grammatical information and entries from multiple Sanskrit dictionaries.

3.1 Preprocessing

The transliteration scheme of the input is automatically detected using an adaptation of Indic Transliteration Detect [3], and transliterated to IAST through the Indic Transliteration package. The system also supports special character handling for advanced search capabilities. Wildcard searches can be performed inserting underscore ('_') characters, which act as single character wildcards within words. When a word ends with an asterisk ('*'), the system switches to exact dictionary matching mode. If the input is a single word with no diacrit-

[3]https://github.com/indic-transliteration/detect.py/blob/master/detect.py

ics there is first an attempt to match it directly in the UTF-8 decomposed list of words, and it returns all the entries for the possible words with diacritics. This allows for searches without diacritics. For example the term "śiva" can be matched writing "śiva", "siva" and "shiva".

3.2 Matching using inflection tables and dictionaries

The rule-based approach draws from the inflected form lookup of the University of Koeln [4]. Through this approach are built inflection tables for the non-indeclinable entries of the Monnier Williams dictionaries. The inflection tables are stored in a SQLite database. Associate to the inflection tables are the grammatical informations relative to the type of the word. The code has been rewritten from php to python, using SQLalchemy as ORM. As was done in (Nehrdich et al., 2024), the tables have been converted from SLP1 to IAST for readability, as it causes minimal storage increase. A multi index increased drastically the speed of the lookup, and future version may benefit from the hash indexes offered by PostgreSQL. The original approach suffers from overgeneration in case of particles (such as "ca") and curious lack of common words such as "vṛtti". To handle those case a post processing cleanup was added.

To match words, the system first tries to match them using the inflection table. Words with uncommon prefixes were a common cause of failure, since they are outside the dictionary. To handle those case, the system looks in a list of prefixes, if the words has one of them it tries again while removing the prefix. If the word initials and endings are inside a list of common sandhi rules, it tries to replace them and to match using again the inflection table. When all of the previous fails, it tries to check if the word is directly inside the hashed list of words of all dictionaries. If a word was matched during any of those steps, the entry (or entries) are retrieved and the function ends. In case the function failed, it means that there are probably multiple words inside and is sent to the multi word processing.

3.3 Dictionaries

For the word entries, the digitalized Sanskrit dictionaries from the university of Cologne were employed (Cologne University, 2024). To provide a clean interface, dictionaries were manually selected to provide non redundant quality output. The *Concise Pali Dictionary*[5] was later added as well since many terms employed by Vasubandhu could be found there and not in the Sanskrit dictionaries. In table 1 there is the list of the dictionaries and the number of unique entries in each dictionary. The total number of unique words contained in the database is 246,955. Since the system is for web reading and not for print, the majority of the abbreviations (both references and in text) were removed to increase clarity. To increase usability, in the online interface all the Sanskrit words in the entries were made clickable, returning the clicked entry. This way, if "rājapuruṣa" is searched, it returns the entries for the word, but also the split version "rāja—pūruṣa". Selecting one of the splits, the sub-entry is opened. The dictionary lookup accepts a list of dictionary abbreviation as argument, and tries to return the entries from the dictionaries. If the entry is not present in the selected dictionaries, it tries to look in the hashed dictionary with entries and dictionaries that have it, and returns it.

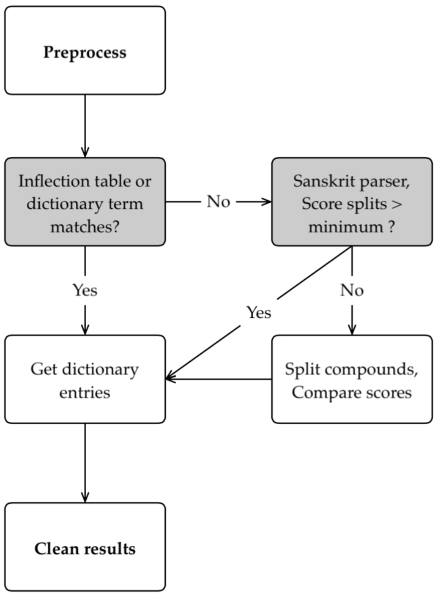

Preprocess
Inflection table or dictionary term matches?
No
Sanskrit parser, Score splits > minimum ?
Yes
Yes
No
Get dictionary entries
Split compounds, Compare scores
Clean results

Figure 2: Flowchart of the cascading system, simple words are directly matched using inflection tables, more complex cases are handled with the parser first, then the compound splitter in case the score if too low.

[4]https://github.com/sanskrit-lexicon/csl-inflect

[5]https://buddhistuniversity.net/content/reference/concise-pali-dictionary

Dictionary Name	Number of Entries
Monier-Williams	194,068
Grassmann	11,108
Apte Practical	31,703
Buddhist Hybrid	17,777
Concise Pali	23,849
Cappeller	38,484
MacDonnell	20,100
Total Unique Words	246,955

Table 1: Number of unique entry in each dictionary, and total of unique words

3.4 Sandhi Splitting

For Sandhi and compound splitting the Python library Sanskrit_parser is used as base [6]. The library places every possible split in a graph and attempts to find the most probable split. As stated in the documentation, the first result provided is often not accurate, but the correct one is usually to be found in the first ten splits.

The incorrect splits showed patterns that were incredibly easy to spot: multiple dual letter fragments such as "to" and "ta", non grammatical entries, incorrect sandhi usage. Any application intended for general public use and also for a non professional audience should be providing a single split for a sentence. Recovering the right split amid possible ones may be easy for someone that knows the language, but risks alienating further other kind of interested users.

To select the best split among the offered one, a simple scoring system was made that evaluates the splits on three dimensions: length, morphology and sandhi. The length score tries to predict the number of split, and rewards a number of splits close to the expected. Less splits to the expectation are preferred to more, since errors are usually words being broken up into multiple places. The morphology score punishes multiple very short words that are not in the list of the indeclinable or of the word suffixes (like "tva"). The sandhi score monitors that the sandhi rules were correctly applied. The split with the best score is then selected. If the split is under the confidence threshold, the word is sent to the last cascading fallback. It is to note that every compound with no presence of Sandhi returns none at this stage and is processed by the compound splitter.

[6]https://github.com/kmadathil/sanskrit_parser

Type	Total	Errors	Err%	Acc%
V.Long	654	58	8.87	91.13
Long	377	22	5.84	94.16
Medium	187	13	6.95	93.05
Total	1218	93	7.63	92.37
Y.Sūtra	665	27	4.06	95.94

Table 2: Text Accuracy Analysis. *Total excludes Y.Sūtra

3.5 Compound Splitter

The root compound function takes advantage of a characteristic of Sanskrit grammar: in compounds, only the rightmost word is declined. All the words on the left are then in their root form. Assuming it is a pure compound with no Sandhi, it's going to be possible to reach the first word on the left by simply erasing the rightmost letter one by one and trying to match it with the hashed vocabulary with all the dictionary entries. In the ideal pure compound, this returns the leftmost word in O(n) operations, where n is the number of the remainder. Since the rightmost word is inflected, after removing a word on the left, it should be checked if the remainder is in the inflection tables, using replacements in case there is possible sandhi at the endings. Since most compounds contain 2–8 roots, and each root requires O(n) operations to find, with n decreasing at each step, the practical performance remains efficient despite the theoretical $O(n^2)$ worst case.

The complexity of this operation actually increases since pure compounds are rare, and often there is the presence of Sandhi either in the initial position or in the middle. To handle these cases two dictionaries with replacements for the initial and ending position are used. This way if a word if a letter is found that could have been the result of Sandhi, it is tested with all the grammatical possibilities (usually less than two) before getting erased. To handle cases like "kleśakarmavipākāśayair", to avoid it to be split after "kleśaka", which is in the dictionary, when selected suffixes like "ka" are meet, the system tries to split again ignoring the suffix, and measures (in terms of length of words) the quality of the results and pick the best one.

3.6 The problem with current Sanskrit Benchmarks

To test the accuracy of the computational approaches to Sanskrit, the Sandhikosh benchmark has been proposed (Bhardwaj et al., 2018) (Aralikatte et al., 2018) , which includes 13,930 annotated sentence splits. The sentence are split only for

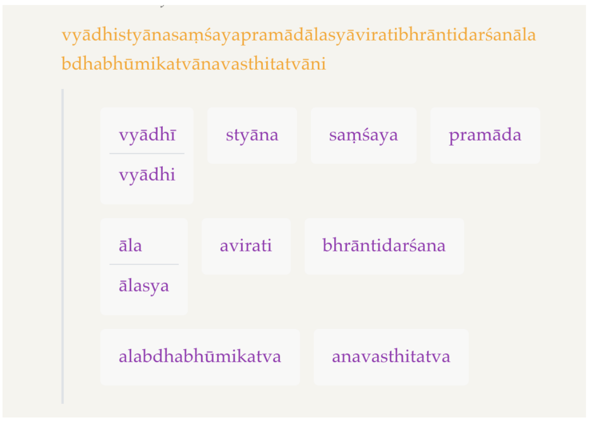
Sandhi and not for compounds, which remains agglutinated together. Since this system splits sandhi and compounds in the same pass, the benchmark is not usable to test the proposed approach. It should also be mentioned that the corpus used is extremely unbalanced in favor of Brahmanical text compared to Buddhist ones, drawing extensively from the online corpus of the University of Hyderabad [7]. A more interesting corpus is the Sighum one, presented in (Krishna et al., 2017). The corpus has been used as a benchmark in the inspiring (Nehrdich et al., 2024). The Sighum corpus, similarly from the Hackaton [8], is derived from the Digital Corpus of Sanskrit (Hellwig, 2010–2021). Those corpus provide the roots for all the sandhi and compound split words in the sentence, similarly to the approach proposed there.There is however a important methodological difference which should be considered. This difference can be explained with the parsing of the block "dagdhabījakalpān" which appears in the Yoga Sūtra Bhasya. In the proposed system the block is split in "dagdha", "bīja" and "kalpa". The word "dagdha" is indicated as coming from dah in the provided vocabulary entries (from Apte Practical Sanskrit-English:"dagdha Past passive participle. [dah-kta] 1 Burnt, consumed by fire"). In the DCS the word is directly described as the "PPP" of "dah". While both approaches are grammatically equivalent, the approach used here provides a more specific dictionary entry, with the possibility of accessing the primitive "dah" with an additional click. The DCS approach returns instead directly the primitive without an additional action. Since the current system is built with the explicit goal of vocabulary entry retrieval in mind, rather than stemming, for the current goal is preferable to keep it as it is. For the same reasons common compounds which are present in the dictionaries, such as "rājayoga" are not split. "Rājayoga" and other similarly common compounds have specific dictionary entries, and the entry offers also the detailed parsing "rāja—yoga". The two split parts can be accessed with a simple click on the online interface. This methodological difference makes it so that simply using the smaller corpus derived by the DCS would result in countless errors, derived by the different format of the output. In a benchmark of tens of thousands of sentences it would be impossible to parse manually all those errors. For those reasons is impossible to test the current approach on any benchmark directly derived from the DCS. It also highlight the problem with every current benchmark testing in Sanskrit: each system employs his own convention. A good benchmark should be able to return positive for both "dah" or "dagdha". Since no similar benchmark currently exist, we manually test the system on a random selection of the Gretil corpus and on a practical use case on the Yoga Sūtra text.

3.7 Testing

Owing to the problems with the current Sanskrit benchmarks highlighted in the last paragraph, an alternative testing approach was used. Since all the simple words are deterministically parsed, what needed to be tested is the capability of the fallback systems to handle complex compounds and the applicability on the automatic annotation of a real text. All the text of the Gretil corpus was extracted and split in four lists of words in the following categories: medium 10–20, long 20–40, very long 40+. To avoid English words mixed in, only words with diacritics were kept. From each of the four lists 100 random samples were taken. The testing was made to check if the system is resilient enough to handle tasks that cannot be handled by the deterministic matching. The system was applied to each one of those compounds, and manually reviewed. For the practical use case the system was tested on the Yoga Sūtra in the transcription by Philip Maas accessible through Gretil [9]. Both tests can be replicated with the testing module inside the python library.

Undecidable words such as "ālasya" are returned with both possible parsings: the uninflected "ālasya" and the genitive of "āla". Since the only way to decide between the two is looking at the context, both words are returned, and is not counted as an error as long as the correct parsing is there. Even in presence of those cases, the system tends to not overgenerate. Figure 3 present the parsing results from a complex text block from the Yoga Sūtra: some of the words are presented with two possible parsing, such as "ālasya" and "āla" for the "ālasya" in the text. Every non perfect parsing is counted as an error. Errors are counted on a root by root basis: if a compound has 10 roots and only one is incorrect, a single error out of then is counted.

[7]https://sanskrit.uohyd.ac.in/Corpus/
[8]https://sanskritpanini.github.io/
[9]https://gretil.sub.uni-goettingen.de/gretil/1_sanskr/6_sastra/3_phil/yoga/patyogbu.htm



vyādhistyānasaṃśayapramādālasyāviratibhrāntidarśanāla
bdhabhūmikatvānavasthitatvāni

vyādhī
vyādhi
styāna
saṃśaya
pramāda
āla
ālasya
avirati
bhrāntidarśana
alabdhabhūmikatva
anavasthitatva

Figure 3: Parsing of one of the Yoga Sūtra verses with multiple possible roots.

In Table 2 are presented the result of the testing, maintaining 90%+ accuracy for all categories. The most surprising result is the high accuracy rate with long compounds considering that the longest one was 283 character long. The practical example of the Yoga Sūtra shows that the accuracy increases with a normal text in which the inflection tables can be used to automatically parse single words such as "atha", "ca" or "iti" or the multiple variations of "yoga".

3.8 Error discussion

The majority of the errors come from words which are outside the dictionaries or the inflection tables, and are then unrecognized. The Monnier-Williams dictionary is from 1899 and, while being an incredible work, is oddly missing some reasonably common compounds like "dvandva" (which is instead present in the Macdonnel dictionary). In particular, about 20% of the overall errors come from abstract words produced with the suffix *tva*, which are sometimes recorded (such as "śūnyatva", from "śūnya", emptiness, which is even recorded as the even rarer "sarvaśūnyatva"), but more often than not outside the dictionary entries. The system at the moment returns the root word, the suffix "tva" and the inflected ending as another word, which is clearly not optimal. Since the inflection tables are based entirely on the Monnier Williams entries, all the inflected forms of words outside the Monnier Williams that are not listed as entries may provoke errors. The shape of those errors is typically the word being rightfully recognized plus the inflectional suffix being marked as another word. Less common but still present are the cases in which the word root is morphed. In that case the word is split in small morphemes. Less common verbal forms, like causatives, are often not listed in the inflection tables; the inflection tables are also missing many irregular forms. A possible solution would be to use LLMs to generate the missing inflection tables, and to also use LLMs to search inside the Monnier Williams and other Dictionaries for mentions of irregular forms, and to apply them to the tables.

The sandhi splitter is, even with the scoring, the weakest part of the pipeline. Further versions of this approach could try replacing it entirely with the model developed in (Nehrdich et al., 2024) to increase the accuracy. The other alternative avenue explored was fine-tuning using a Llora (Hu et al., 2021) a middle sized LLM. Since the errors produced by the system are easily identifiable, it is possible to use the output of this system to train a transformer that replaces it. This replacement should be assessed with respect to the increased computational cost and the scalability of the online application. In the best case scenario the current approach resolves annotation with just a few SQL queries. There is no reason to replace the inflection table lookup, as it is deterministically correct and computationally inexpensive.

4 Limitations

The cascading system is highly modular; consequently, most limitations stem from the current implementation rather than from the architecture itself.

The system relies heavily on dictionary entries, with the majority derived from the Monier-Williams dictionary (1899), as illustrated in Table 1. While the Monier-Williams dictionary provides a comprehensive foundation, it exhibits notable deficiencies regarding abstract words formed with the "-tva" suffix and numerous compounds of moderate frequency. Although these dictionary limitations are partially mitigated through the integration of multiple dictionaries, the inflection tables are currently constructed solely from the Monier-Williams dictionary. Furthermore, certain common terms, such as "vṛtti" (vortex, mental fluctuation), appear in the dictionary but are notable absent from the inflection tables. The next version of the implementation should take care in reconstructing the inflection table using the correct list of all words as a basis. A possible approach for adding all the irregular forms would be using language models to extract them from dictionaries and grammar books.

Even with the scoring improvements, the Sandhi splitter remains the weakest component in our pipeline. While it works well for most cases, complex Sandhi patterns can still lead to incorrect splits. Future versions can replace this component with neural models such as the one described in (Nehrdich et al., 2024), although this would increase computational costs. The computational increase would still be limited for just the complex cases, since for most words the inflection tables are going to still be enough.

Finally, our approach prioritizes dictionary entry retrieval over stemming, which creates methodological differences when compared to existing benchmarks. This system prioritizes keeping compound and inflected forms intact when the dictionary entry is there: "yogānuśāsanam" (the instruction on yoga) is matched directly with the entry for "yogānuśāsana" that contextualizes the term within Patañjali's framework, rather than being decomposed into the components "yoga" and "anuśāsana". While this is an advantage for a text annotation tool, since the results are more context aware and usually present the split in the dictionary entries, it is a severe limitation when used as a pure stemming tool.

5 Conclusions

This work demonstrates that by taking a progressive approach to Sanskrit text processing, starting with simple, deterministic methods and escalating to more complex analysis only when necessary, it is possible to achieve both high accuracy and practical usability. The system's 90%+ accuracy on the long compounds drawn from the GRETIL corpus and 96% accuracy on the Yoga Sūtra validates this approach, showing that it performs reliably across different text types and compound complexities. The system's capability lies in the deterministic lookup for inflected and Sandhi-ed single words, returning entries from multiple dictionaries with accurate grammatical information with minimal computational cost. The approach peculiarity is in keeping compounds and inflected forms which have entries in the dictionaries, returning more contextualised entries than direct stemming. This approach has been implemented as an open source Python library which includes a REST API built using Flask and a web application, which is accessible at `https://www.sanskritvoyager.com`. The system was designed with practical performance considerations in mind. The entire backend requires only 2GB of RAM to operate effectively, making it deployable on modest hardware. Response times vary based on word complexity: simple inflected forms that can be resolved through table lookups are processed in milliseconds, while the most complex compound and Sandhi cases require at most 3 seconds to resolve on a 4GB RAM virtual machine. This performance profile makes the system suitable for both interactive web applications and batch processing of larger texts. The current approach allows dictionary searches for Sandhi-ed inflected and compounded words, without specifying the transliteration scheme and retrieving entries in multiple dictionaries. Future versions may employ a ByT5 based model (Nehrdich et al., 2024) as the last step in the cascading system to handle the most complex cases. The hope of this approach is to open up the treasury of the original Sanskrit literature to any interested reader, regardless of their previous linguistic skills.

References

Rahul Aralikatte, Neelamadhav Gantayat, Naveen Panwar, Anush Sankaran, and Senthil Mani. 2018. Sanskrit sandhi splitting using seq2 (seq)ˆ 2. *arXiv preprint arXiv:1801.00428*.

Shubham Bhardwaj, Neelamadhav Gantayat, Nikhil Chaturvedi, Rahul Garg, and Sumeet Agarwal. 2018. SandhiKosh: A benchmark corpus for evaluating Sanskrit sandhi tools. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

George R Cardona. 1988. *Pāṇini : his work and its traditions*. Motilal Banarsidass, Delhi, India.

Cologne University. 2024. Cologne digital Sanskrit dictionaries. Online resource. Accessed on February 19, 2025.

Surendranath Dasgupta. 1991. *The Mahabhasya of Patanjali (Ahnikas I-IV)*. INDIAN COUNCIL OF PHILOSOPHICAL RESEARCH, New Delhi.

Brendan S Gillon. 2007. Pāṇini's" aṣṭādhyāyī" and linguistic theory. *Journal of Indian philosophy*, 35(5/6):445–468.

Oliver Hellwig. 2010–2021. Dcs - the digital corpus of sanskrit.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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

Gérard Huet. 2009. Sanskrit segmentation. In *Proceedings of the South Asian Languages Analysis Roundtable XXVIII*, Denton, Ohio.

Stephanie W Jamison and Joel P Brereton. 2014. *The Rigveda: 3-Volume Set*. Oxford University Press.

Amrith Krishna, Bishal Santra, Ashim Gupta, Pavankumar Satuluri, and Pawan Goyal. 2021. A graph-based framework for structured prediction tasks in sanskrit. *Computational Linguistics*, 46(4):785–845.

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.

Sriram Krishnan, Amba Kulkarni, and Gérard Huet. 2024. Normalized dataset for sanskrit word segmentation and morphological parsing. *Language Resources and Evaluation*, pages 1–52.

P H Matthews. 2014. *The concise oxford dictionary of linguistics*, 3 edition. Oxford Quick Reference. Oxford University Press.

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

M. Monier-Williams. 1899. *A Sanskrit-English Dictionary: Etymologically and Philologically Arranged with Special Reference to Cognate Indo-European Languages*. The Clarendon Press, Oxford.

Sebastian Nehrdich, Oliver Hellwig, and Kurt Keutzer. 2024. One model is all you need: Byt5-sanskrit, a unified model for sanskrit nlp tasks. *arXiv preprint arXiv:2409.13920*.

PK Narayana Pillai. 1941. The gveda padapāha—a study with special reference to the gveda prātiśākhya. *Bulletin of the Deccan College Research Institute*, 2(3/4):247–257.

Reut Tsarfaty, Dan Bareket, Stav Klein, and Amit Seker. 2020. From SPMRL to NMRL: What did we learn (and unlearn) in a decade of parsing morphologically-rich languages (MRLs)? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7396–7408, Online. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.