IndiAnn: A Web-based Annotation Platform for Indic Languages

Bandaru Lavadeep, Ritwik Raghav, Abhik Jana
IIT Bhubaneswar, India
{22cs01049,a23cs09001,abhikjana}@iitbbs.ac.in

Abstract

Linguistic annotation tools that work well for non-Indic languages (e.g. English, German, Spanish, etc.) often fail with Indic scripts due to complex Unicode properties, including visual reordering of vowel matras, conjunct characters, and grapheme clusters spanning multiple code points. In this paper, we present a web-based annotation platform *IndiAnn*, designed for low-resource Indic languages, which uses native browser Unicode rendering, offset-based storage that preserves grapheme clusters, and no forced tokenization in the user interface. The tool supports annotation for tasks such as part-of-speech (POS) tagging, named entity recognition (NER), dependency relation annotation, and semantic role labelling (SRL), that maintain correct character boundaries and enable seamless interoperability with standard NLP pipelines and tools. The framework is designed for Indic languages and has been tested on Telugu, Hindi, Tamil, Malayalam, Bengali, Odia, Marathi, and Kannada, with no script breakage during annotation. To the best of our knowledge, this is the first ever attempt at building a unified annotation framework (IndiAnn), which covers annotation for such varieties of key NLP tasks, having provision for eight Indic languages. The code repository is made publicly available[1].

1 Introduction

Linguistic annotation is essential for building high-quality datasets for natural language processing, especially for low-resource languages. Large annotated resources such as Universal Dependencies (Nivre et al., 2016) and shared tasks like CoNLL-2003 (Sang and Meulder, 2003) highlight the importance of structured annotation for downstream NLP tasks. However, many languages, particularly Indic languages, still lack sufficient annotated resources (Joshi et al., 2020; Maji et al., 2025).

Most existing annotation tools are designed primarily for languages such as English, German, Spanish, etc. and assume a simple, linear character model with fixed tokenization. These assumptions do not hold for Indic languages, which have more complex writing systems. Indic scripts present several challenges due to their Unicode properties. In many cases, the storage order of characters differs from their visual display order, particularly because of vowel matras and combining marks (The Unicode Consortium, 2022b; Ansary et al., 2024). Moreover, what appears as a single character to the annotator is often composed of multiple code points forming a grapheme cluster, which is the more appropriate unit for segmentation in these scripts (Ansary et al., 2024). As a result, tools that rely on byte-level or code-point indexing often produce incorrect offsets, leading to incorrect span boundaries, cursor misalignment, and broken annotation highlighting. These issues directly impact annotation quality and make it difficult to build reliable datasets. Widely used annotation tools such as WebAnno (Yimam et al., 2013), INCEPTION (Klie et al., 2018), and Brat (Stenetorp et al., 2012) do not explicitly address these challenges, as they are primarily designed for languages with simpler orthographic structures. In addition, many of these systems rely on fixed tokenization schemes, which can incorrectly split or merge units in Indic languages, especially in the presence of conjunct characters or multi-word expressions.

Existing efforts for Indian languages, such as the Paninian dependency annotation scheme by Begum et al. (2008) and the ILCIANN tool by Kumar et al. (2021), are typically designed for specific tasks and operate over pre-tokenized or structured text. While effective for large-scale annotation and linguistic analysis, they do not support direct span selection over raw text as in tools like WebAnno, and therefore do not explicitly address character-level challenges in Indic scripts, such as grapheme

[1] https://github.com/Lavadeep/INDIANN

Proceedings of the 20th Linguistic Annotation Workshop (LAW XX), pages 130–145
July 3, 2026 ©2026 Association for Computational Linguistics

clusters and rendering-order mismatches.

In this paper, we present a web-based annotation platform - **IndiAnn**, designed specifically for Indic languages. The system is built around native Unicode rendering and uses a single canonical text with offset-based annotation that preserves grapheme clusters. Unlike traditional tools, it does not enforce tokenization in the user interface, allowing annotators to select natural linguistic units directly. The proposed system supports multiple annotation layers, including span-based annotations like POS tagging and NER, relations between spans, dependency relation annotation, and semantic role labeling. It accepts input in a variety of formats, including Plain text, Word documents, PDF files, and CoNLL-U, enabling flexible data ingestion. For interoperability, the system provides export functionality in JSON format, as well as CoNLL-U for POS and dependency annotations. The platform is designed for Indic languages and has been tested on Telugu, Hindi, Tamil, Malayalam, Bengali, Odia, Marathi, and Kannada, ensuring compatibility with diverse scripts and linguistic structures.

2 Related Work

Several annotation platforms have been developed for linguistic annotation, including WebAnno (Yimam et al., 2013), INCEPTION (Klie et al., 2018), and Brat (Stenetorp et al., 2012). These systems support multiple annotation layers and collaborative workflows, and have been widely used to annotate documents written in European languages. More recent tools such as doccano (Nakayama et al., 2018), and Label Studio (Tkachenko et al., 2020-2025) provide flexible web-based annotation interfaces.

Despite their capabilities, these tools are largely designed with assumptions that align with European languages and simpler writing systems. Prior work has shown that many NLP tools implicitly rely on whitespace-based tokenization and linear character representations, which do not generalize well to languages with complex scripts (Bird et al., 2009). This limitation becomes more pronounced in multilingual settings, where language-specific properties are often not adequately handled (Bender, 2011).

In the context of Indic languages, several efforts have been made to develop resources and tools tailored to local linguistic phenomena. Begum et al. (2008) propose a dependency annotation scheme based on the Paninian framework, introducing syntactic-semantic relations suitable for morphologically rich, free-word-order languages such as Hindi. Their approach operates over pre-tokenized and chunked text, where annotations are defined between predefined units rather than directly on raw text spans. Similarly, Kumar et al. (2021) presents ILCIANN, a web-based annotation tool designed to create and manage large-scale parallel corpora across multiple Indian languages. The system supports distributed annotation and project management, but restricts annotation to pre-tokenized word-level units.

Initiatives such as the Indian Languages Corpora Initiative (ILCI) (Jha, 2012) and related efforts have contributed significantly to the development of annotated datasets and guidelines for Indian languages. However, these approaches are often dataset- or task-specific, and their reliance on token- or chunk-level annotation abstracts away character-level complexities inherent in Indic scripts.

When general-purpose annotation tools are applied to Indic data, several issues arise. Offset calculations based on bytes or code points can become inconsistent with the visually rendered text, leading to incorrect span boundaries, cursor misalignment during selection, and broken highlighting. These issues are closely related to Unicode text segmentation and grapheme cluster handling (The Unicode Consortium, 2022b,a). Similar challenges have also been observed in multilingual NLP settings where script-specific properties are not properly handled (Joshi et al., 2020).

Conjunct characters and ligatures may be incorrectly split, and tokenization methods designed for Latin scripts often produce inconsistent segmentation for Indic languages. This limitation has been noted in prior work on Indian language processing, where language-specific tokenization and normalization are required for accurate analysis (Kunchukuttan, 2020; Bharati et al., 1996). In addition, many annotation tools enforce predefined tokenization schemes that do not adapt well to morphologically rich languages (Tsarfaty et al., 2010), further reducing annotation flexibility.

Overall, existing tools and efforts do not provide a unified solution that simultaneously handles Unicode-aware rendering, grapheme-consistent offsets, and flexible annotation workflows for Indic languages. This gap makes it difficult to cre-

ate high-quality annotated corpora in such settings. Our work addresses these challenges by designing an annotation platform that aligns with the properties of Indic scripts. By using native Unicode rendering, maintaining a single canonical text, and avoiding forced tokenization, the system ensures accurate selection, stable offsets, and consistent annotation across multiple layers.

3 Challenges of Indic Script

Indic scripts are composed of grapheme clusters, where a single visible character may consist of multiple Unicode code points. Proper segmentation, therefore, requires grapheme-level processing rather than code-point-based indexing (The Unicode Consortium, 2022b,a; Ansary et al., 2024). Indic scripts pose inherent challenges that generic tools do not account for:

Characters visually reorder: Storage order of the graphemes may not be the same as the display order. Mostly, the vowel signs and combining marks are stored after the base consonant, but they can be rendered before, above, or below it. Hence, byte or code-point indices do not always correspond to the visually perceived character sequence. For example, in Bengali the grapheme ড়া is internally stored as consonant + nukta + vowel sign, while visually rendered as a single grapheme cluster with reordered components.

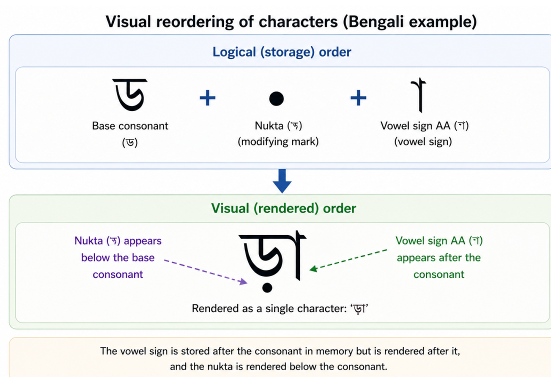

Figure 1: Example of visual reordering in Bengali grapheme rendering.

Many glyphs are formed by multiple Unicode code points: what appears as one "character" (one grapheme cluster) may be several code points (e.g. base consonant + nukta + vowel sign, such as Bengali ড়া = ড + ◌় + ◌া). Treating each code point as a unit breaks selection and alignment.

Vowels attach in multiple positions: Vowel signs can appear before, after, above, or below the consonant, so a single "character" in the user's mind corresponds to a variable-length sequence in memory.

Segmentation of grapheme clusters is non-trivial: Identifying user-perceived character boundaries requires Unicode grapheme cluster rules; naïve substring or tokenization often splits or merges them incorrectly.

An annotation tool for Indic languages must therefore: (1) use a rendering model that respects Unicode (e.g. native browser rendering); (2) store and interpret offsets in a single, stable string that matches what is displayed; (3) avoid forced tokenization that could split grapheme clusters or words incorrectly; and (4) support standard interchange formats like CoNLL-U (Nivre et al., 2016) while preserving correct character boundaries.

4 System Features

Our tool is designed so that Indic text is rendered, selected, and annotated without the issues described above. The following points summarize the main design choices.

Native Unicode Rendering The tool uses native browser Unicode rendering, which fully supports Indic scripts. It does not rely on a custom or simplified rendering engine; instead, it leverages the same rendering stack used by modern browsers for complex web scripts. As a result, issues related to reordering and ligatures that arise when display order is handled separately from storage are naturally mitigated.

Accurate Selection and Highlighting The tool provides accurate text selection and highlighting for Telugu, Hindi, Tamil, Malayalam, Bengali, Odia, Marathi, and Kannada across multiple annotation views (POS, span-rel, and SRL). After fetching `doc.content`, the frontend constructs a single client-side representation by collapsing consecutive whitespace and trimming leading/trailing spaces, and then derives sentence boundaries using a punctuation-based sentence segmentation strategy after normalization (with a punctuation-based fallback otherwise, including Indic sentence terminators). Annotation start/end offsets are computed against the same representation that is used to render sentence containers and highlight spans. As a result, the browser selection ranges align with the stored offsets used for highlighting, enabling stable and correct span rendering for Indic text in all supported layers.

5 Implementation & Features

Section 4 described the design principles that guide IndiAnn, particularly its Unicode-aware rendering and offset-preserving annotation model. In this section, we describe the core implementation choices that realize these principles in practice, the annotation layers currently supported by the platform, and the architectural flexibility that allows the framework to be extended to new annotation tasks while maintaining consistent alignment with the underlying text.

5.1 Core Implementation Design

Offset Storage and Grapheme Clusters The tool stores start/end offsets in a single offset space. All annotations (POS, spans, relations, dependencies) refer to this same string. During rendering, the client derives sentence containers and highlight spans using a whitespace-normalized representation of the stored content, so offset computations are performed against the same representation used for display. Offsets are computed in a manner that preserves grapheme clusters. Consequently, the substring boundaries used for highlighting match the mapped offset ranges, preventing script breakage due to unintended splitting or merging of grapheme clusters.

No Forced Tokenization The tool does not impose forced tokenization in the frontend or display pipeline. Instead of enforcing a fixed token grid that could split conjuncts or combine unrelated units, sentence boundaries are provided by the backend, but the frontend reconstructs sentence spans using a lightweight punctuation-based segmentation strategy. Within this structure, users can select spans corresponding to natural linguistic units (words, morphemes, or multi-word expressions), thereby avoiding the boundary errors typical of tokenizers designed for Latin scripts.

CoNLL-U Import/Export and Character Boundaries CoNLL-U import and export are supported while maintaining correct character boundaries. During import, the CoNLL-U file is converted into a canonical plain-text representation (e.g., one sentence per line), and token offsets are recomputed in this representation so that POS and dependency annotations align with the same underlying text. During export, annotations are merged back or generated from this canonical form. Consequently, CoNLL-U round-trips do not introduce offset drift or misaligned spans.

Alignment of All Annotation Types All annotation types (POS, spans, relations, dependencies) are stored using the same start/end offset model and align precisely with the underlying Indic text. A single offset space is maintained for the document, eliminating the possibility of inconsistencies that arise from separate token-index spaces. This unified representation ensures that POS tags, span labels, and dependency arcs remain correctly aligned with their corresponding character ranges.

Backend: FastAPI (Python) with PostgreSQL for persistence. Documents are stored with a single canonical content string; annotations store `start_offset` and `end_offset` into this string.

Frontend: Static HTML/JavaScript/CSS. The document content is fetched from the API and rendered in the browser. Before computing sentence containers and offsets for highlighting, the frontend collapses consecutive whitespace and trims leading/trailing spaces to ensure a consistent representation. Sentence boundaries are derived using a lightweight punctuation-based segmentation strategy, which includes Indic sentence terminators.

Document Upload: For plain text, the backend applies language-aware sentence tokenization (e.g. NLTK (Loper and Bird, 2002) for general languages, indicnlp (Kunchukuttan, 2020) for Indic languages: Telugu, Hindi, Tamil, Malayalam, Bengali, Odia, Marathi, and Kannada). The result is stored as one sentence per line. For CoNLL-U, the file is parsed; a canonical plain-text form is derived; and token offsets are recomputed in that form so that all annotations refer to the same stored content.

Table 1 summarizes the design choices of **IndiAnn**.

5.2 Supported Annotation Tasks

All annotation layers are represented using a unified offset-based scheme over a single canonical text, where each annotation is stored with `start_offset` and `end_offset`, ensuring consistent alignment across layers.

Part-of-speech (POS) Tagging: annotation of tokens or spans with grammatical categories such as nouns, verbs, and adjectives, based on configurable tagsets.

NER Tagging and Relation Annotation:

- **Spans**: generic span annotations just like NER with configurable labels.

Choice	Description
Canonical text	All annotations are defined using character offsets with respect to a single, consistent document representation.
Content Immutability	The document text is not modified after upload; the same content is retrieved without any transformation.
Frontend whitespace normalization	The frontend collapses consecutive whitespace and trims leading/trailing spaces before rendering and computing sentence/offset positions, ensuring offsets remain consistent with the displayed representation.
Backend sentence segmentation	Sentence segmentation is performed during upload or import; the frontend derives sentence structure using a lightweight punctuation-based segmentation strategy over normalized content.
CoNLL-U Alignment	During CoNLL-U import, token offsets are recomputed within the canonical text to ensure alignment of POS and dependency annotations.

Table 1: Design choices of IndiAnn.

- **Relations**: links between spans or tokens, supporting relation extraction tasks.

Dependency Relation Tagging: annotation of syntactic head–dependent relations, where each token is linked to a governing word (head) and assigned a dependency label; compatible with CoNLL-U through token and head indices. This task is integrated within the POS layer (See Figure C.12 of Appendix C).
Semantic Role Labelling (SRL): annotation of predicate–argument structures, where arguments are associated with predicates and represented using character offsets.

All of these use the same document content and offset space, so they stay aligned with Indic text when the pipeline respects grapheme clusters and avoids forced tokenization.

5.3 Extensibility of Annotation Layers

IndiAnn follows a modular layer-oriented architecture in which each annotation layer is defined by a task-specific label schema, an offset-based representation over canonical text, and lightweight frontend interaction logic. Since all layers share the same `start_offset`/`end_offset` alignment model, new annotation layers can be introduced without modifying the core rendering, storage, or curation pipeline.

Beyond the currently supported layers the framework can be extended to additional span-level classification, event annotation, discourse relation labeling, coreference annotation, and other structured linguistic tasks through project-level label configuration and minor interface extensions.

At present, IndiAnn is best suited for text-centric annotation tasks grounded in character-offset spans or relations over canonical text. Tasks requiring multimodal alignment, deeply nested graph structures, or specialized preprocessing may require additional modeling beyond the current implementation.

5.4 Sentence Segmentation and Scalability

Sentence segmentation is performed once at upload time. For plain text documents, the backend applies language-aware sentence tokenization (e.g., NLTK for non-Indic languages and the Indic NLP Library for Indic languages), then joins the resulting sentences using newline characters to form a canonical string. For CoNLL-U documents, each sentence block is converted into a canonical plain-text form by representing it as a sequence of whitespace-separated tokens. Token offsets are then recomputed over this representation to ensure alignment with the underlying text.

Upon retrieval, the API returns this canonical content; when it is already sentence-separated, the frontend derives sentence containers using punctuation-based segmentation over the normalized text. If newline segmentation is not available, the frontend uses a lightweight punctuation-based heuristic to derive sentence spans for rendering and offset mapping.

Because sentence boundaries are encoded as lightweight newline delimiters and annotation offsets are defined over a single immutable string, the system scales linearly with document length and annotation count. Rendering and interaction primarily involve substring operations and span insertions, which maintain responsiveness even for relatively large documents.

5.5 Sparse and Incremental Annotation

The tool supports sparse and incremental annotation. Annotators are not required to label every token or span in a document; instead, they can focus

on selected regions (e.g., complex constructions or specific phenomena) and extend coverage in subsequent passes. All annotation layers (POS, spans, relations, dependencies, SRL) are stored independently as records keyed by document ID and character offsets into the canonical text. As a result, additional annotations can be introduced without modifying or recomputing existing ones, enabling efficient incremental annotation workflows while preserving offset consistency.

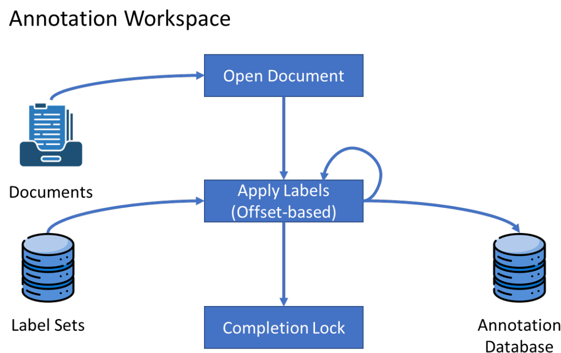

Figure 2: Workflow Diagram for Annotator

5.6 Three-Tier Management

The platform supports a collaborative annotation workflow with distinct roles. The graphical user interface can be described as three role-oriented views: (i) an annotation view for annotators, (ii) a curation view for curators to resolve conflicts and consolidate decisions, and (iii) an administration view to configure projects, labels, and membership.

Annotator Annotators perform primary annotation on assigned documents. In the annotation view, the user is presented with the document text rendered using native Unicode support and a compact control panel for labeling actions. Annotation is performed directly on the text: the annotator selects a span and assigns a label; the tool stores start/end offsets relative to the canonical document content, ensuring that selections remain stable for Indic grapheme clusters. The annotator's workflow is visualized in the Figure 2.

Typical annotator actions include:

- **Labeling**: Apply labels by selecting text and choosing a label from the available tagset.
- **Deletion and correction**: Delete incorrect annotations (including, where applicable, associated dependent structures such as SRL roles attached to a deleted predicate) and re-annotate the corrected span. The system does not currently support direct in-place editing of annotations; corrections require manual re-selection of the span (Figure 3).
- **Multi-layer annotation**: Work across multiple layers (POS, spans/entities, relations, dependencies, SRL) over the same canonical text, depending on the project configuration. It is displayed in Figure 4.
- **Export**: After annotations are done, the annotated file can be exported using standard formats like JSON (See Figure B.5 of Appendix B) and CoNLL-U (See Figure B.6 of of Appendix B) for dependency/POS workflows where applicable.
- **Progress control**: There is a provision of progress control to mark documents as complete to reduce accidental edits.

Curator Curators validate, resolve conflicts, and build a higher-quality consensus layer over annotator submissions. The curation view aggregates all annotations that share the same $(start, end)$ span and summarizes them *by label*. For example, the same highlighted word or span might have been annotated as *noun* by five annotators and as *verb* by three annotators. For each competing label, the interface provides explicit curation actions (approve/reject), allowing curators to record a decision at the level of label groups rather than inspecting annotations one by one. Curation itself is only made available once all assigned annotators have *locked* (marked as completed) their work for a document; this gating is configured and monitored by the admin role. The curation workflow is visualized in the Figure 5. This figure shows a curation tab of an NER layer (Figure 6)

Typical curator actions include:

- **Conflict inspection**: view all labels proposed by different annotators for the same $(start, end)$ span.
- **By-label breakdown**: inspect per-label vote counts for a span (e.g., *noun: 5*, *verb: 3*), making disagreement patterns immediately visible.
- **Approve/reject per label**: for each competing label group (e.g., *noun* and *verb*), use dedicated **Approve** and **Reject** buttons to accept

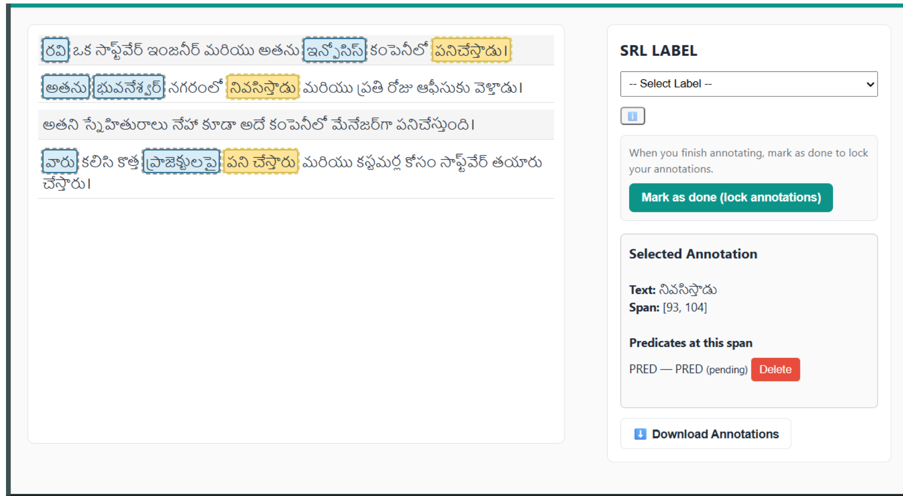

Figure 3: SRL Annotation tab of an Annotator. The document is written in Telugu. The orange coloured box is a predicate while the blue boxes are roles.

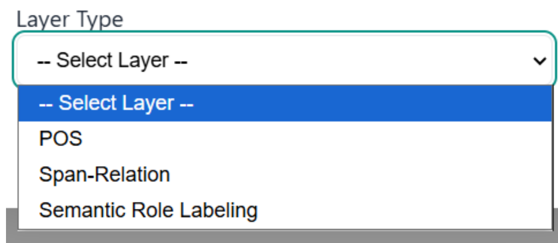

Figure 4: A Snapshot showing provision of multi-layer annotation

or discard that label for the current span; this yields a curated consensus decision.

- **Quality control**: identify systematic problems (e.g. recurrent boundary mistakes) and correct them while keeping the canonical text fixed.
- **Error-focused refinement**: operationalize script-induced issues (offset/boundary errors) as explicit curation decisions.

Admin Admins configure projects and manage users and resources needed to run annotation campaigns. The administration view supports creating projects, defining label sets, assigning user roles, and managing documents and exports. Figure A.1 (Appendix A) shows the admin dashboard of **Indi-Ann**. Typical admin actions include:

- **Project setup**: create projects and configure which annotation layers are enabled (e.g. POS vs. Span-Relation vs. SRL). See Figure 4.
- **Label management**: define the permitted labels/tagsets per project and update them as guidelines evolve. See Figure A.2 of Appendix A.

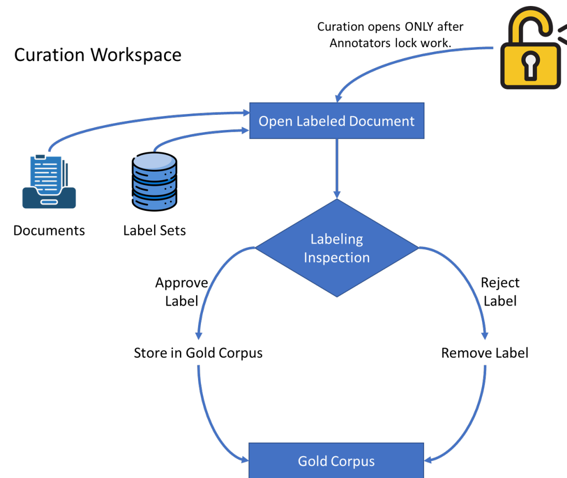

Figure 5: Workflow Diagram for Curator

- **User and role management**: add users to projects and assign roles (annotator/curator/admin). See Figure A.3 of Appendix A.
- **Document management**: upload documents (Plain Text/ CoNLL-U/Word document/PDF), monitor progress, and manage exports.
- **Completion locks**: monitor whether annotators have locked (completed) a document, and if needed, *reverse/unlock* a lock to allow further edits (e.g., after guideline updates or detected errors). See Figure A.4 of Appendix A.
- **Curation gating**: enforce or configure the policy that curation starts only once all assigned annotators have locked their work, so

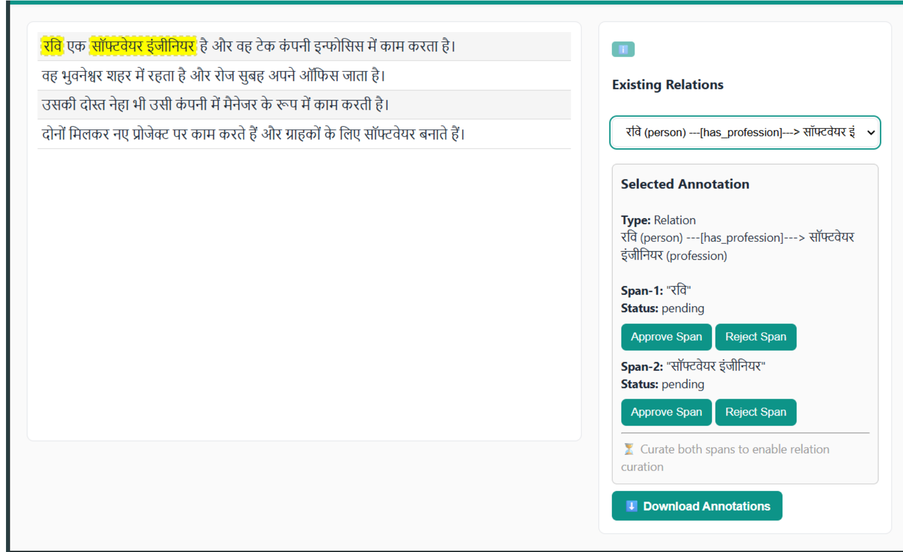

Figure 6: NER Curation tab of a Curator. The document is written in Hindi Language.

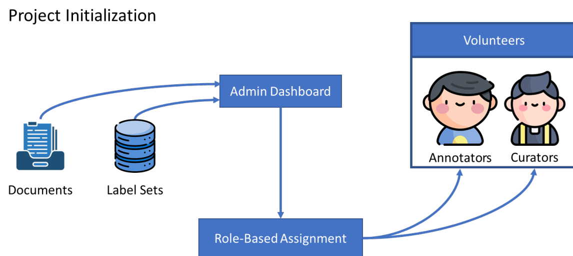

Figure 7: Workflow Diagram for Admin

curators can curate a stable snapshot of annotations.

- **Governance**: enforce workflow policies appropriate for a dataset (e.g. completion locks, curation requirements) and maintain traceability of campaign progress.

The typical workflow for admins is presented in Figure 7.

6 Linguistic Coverage

The tool is designed for low-resource Indic languages, including Telugu, Hindi, Tamil, Malayalam, Bengali, Odia, Marathi, and Kannada. More generally, it supports scripts that follow standard Unicode encoding and use explicit whitespace for word delimitation. This design makes the platform applicable to most Indic scripts used in Indo-Aryan and Dravidian languages, in which grapheme clusters align with user-perceived character units and can be handled reliably via native rendering and offset-based annotation. However, linguistic phenomena that obscure word boundaries (e.g., sandhi or compounding) may require additional handling beyond the current design.

Although IndiAnn was developed primarily for Indic languages, its design is not restricted to Indic scripts. By relying on native Unicode rendering, grapheme-consistent offset storage, and direct span selection over canonical text, the framework can support other complex scripts as well. In preliminary experiments, we observed stable rendering and annotation behavior for Arabic and Urdu, suggesting that extending IndiAnn to additional Unicode-compliant scripts would require minimal changes beyond language-specific preprocessing such as sentence segmentation.

7 Error Mitigation

IndiAnn is designed to reduce annotation errors that arise when software assumptions about text representation do not align with the properties of Unicode-complex scripts. By relying on native Unicode rendering, preserving annotations in a single canonical offset space, and avoiding forced tokenization in the annotation interface, the framework minimizes cursor-offset mismatch, broken span boundaries, and annotation inconsistencies caused by grapheme fragmentation

8 Conclusion

We have proposed an annotation platform (**IndiAnn**), built for low-resource Indic languages. The key features of this framework are: (1) reliance on native Unicode rendering and a single canonical text with offset-based annotations that preserve grapheme clusters; (2) no forced tokenization in the user interface, avoiding script breakage. Together, these properties allow accu-

rate selection, highlighting, and export for Telugu, Hindi, Tamil, Malayalam, Bengali, Odia, Marathi, and Kannada, addressing the limitations of tools that are built mainly for European scripts and that suffer from vowel matra reordering, conjunct handling, multi-byte Unicode, and tokenization errors. **IndiAnn** supports annotation for four different tasks such as POS tagging, NER, Dependency Relation extraction and Semantic Role Labeling. This framework would be the stepping stone towards seamless annotation experience for low-resource Indic Languages.

Limitations

While the proposed platform is designed to support annotation for low-resource Indic languages, our framework has a few limitations. First, the proposed framework supports multiple annotation layers, but they are handled within a shared workspace, which may increase interface complexity and cognitive load in multi-layer projects. Second, the framework assumes a stable preprocessing pipeline for sentence segmentation and offset alignment. Changes in preprocessing may affect consistency and require careful version control. Third, while the platform handles Unicode-level complexities such as grapheme clusters and rendering across Indic scripts, it does not explicitly address linguistic phenomena such as sandhi and compounding (samās) yet, where multiple lexical units are realized as a single orthographic word. In such cases, boundaries required for annotation may not align with whitespace-delimited tokens, potentially limiting fine-grained annotation. Handling such cases is our immediate future work.

References

Nazmuddoha Ansary, Quazi Adibur Rahman Adib, Tahsin Reasat, Asif Shahriyar Sushmit, Ahmed Imtiaz Humayun, Sazia Mehnaz, Kanij Fatema, Mohammad Mamun Or Rashid, and Farig Sadeque. 2024. Unicode normalization and grapheme parsing of indic languages. In *Proceedings of the 2024 joint international conference on computational linguistics, language resources and evaluation (LREC-COLING 2024)*, pages 17019–17030.

Rafiya Begum, Samar Husain, Arun Dhwaj, Dipti Misra Sharma, Lakshmi Bai, and Rajeev Sangal. 2008. Dependency annotation scheme for indian languages. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*.

Emily M Bender. 2011. On achieving and evaluating language-independence in nlp. *Linguistic Issues in Language Technology*, 6.

Akshar Bharati, Vineet Chaitanya, and Rajeev Sangal. 1996. *Natural Language Processing: A Paninian Perspective*. Prentice-Hall of India.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Girish Nath Jha. 2012. The tdil program and the indian language corpora initiative. In *Language resources and evaluation conference*.

Pratik Joshi, Sebastin Santy, Amar Budhiraja, Kalika Bali, and Monojit Choudhury. 2020. The state and fate of linguistic diversity and inclusion in the nlp world. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 6282–6293.

Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart De Castilho, and Iryna Gurevych. 2018. The inception platform: Machine-assisted and knowledge-oriented interactive annotation. In *Proceedings of the 27th international conference on computational linguistics: system demonstrations*, pages 5–9.

Ritesh Kumar, Shiv Bhusan Kaushik, Pinkey Nainwani, and Girish Nath Jha. 2021. Creating and managing a large annotated parallel corpora of indian languages. *Preprint*, arXiv:2112.01764.

Anoop Kunchukuttan. 2020. The IndicNLP Library. `https://github.com/anoopkunchukuttan/indic_nlp_library/blob/master/docs/indicnlp.pdf`.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, pages 63–70.

Arijit Maji, Raghvendra Kumar, Akash Ghosh, Nemil Shah, Abhilekh Borah, Vanshika Shah, Nishant Mishra, Sriparna Saha, and 1 others. 2025. Drishtikon: A multimodal multilingual benchmark for testing language models' understanding on indian culture. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1289–1313.

Hiroki Nakayama, Takahiro Kubo, Junya Kamura, Yasufumi Taniguchi, and Xu Liang. 2018. doccano: Text annotation tool for human. Software available from https://github.com/doccano/doccano.

Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, and 1 others. 2016. Universal

dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666.

Tjong Kim Sang and De Meulder. 2003. Introduction to the conll-2003 shared task. In *CoNLL*.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107.

The Unicode Consortium. 2022a. Unicode standard annex #29: Unicode text segmentation, version 15.0.

The Unicode Consortium. 2022b. *The Unicode Standard, Version 15.0*. The Unicode Consortium, Mountain View, CA.

Maxim Tkachenko, Mikhail Malyuk, Andrey Holmanyuk, and Nikolai Liubimov. 2020-2025. Label Studio: Data labeling software. Open source software available from https://github.com/HumanSignal/label-studio.

Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kübler, Yannick Versley, Marie Candito, Jennifer Foster, Ines Rehbein, and Lamia Tounsi. 2010. Statistical parsing of morphologically rich languages (spmrl) what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart De Castilho, and Chris Biemann. 2013. Webanno: A flexible, web-based and visually supported system for distributed annotations. In *Proceedings of the 51st annual meeting of the Association for Computational Linguistics: system demonstrations*, pages 1–6.

Appendix

A Provision of Admin

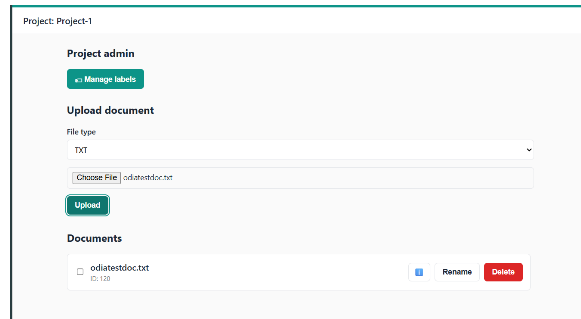

Figure A.1: A snapshot of Admin Dashboard

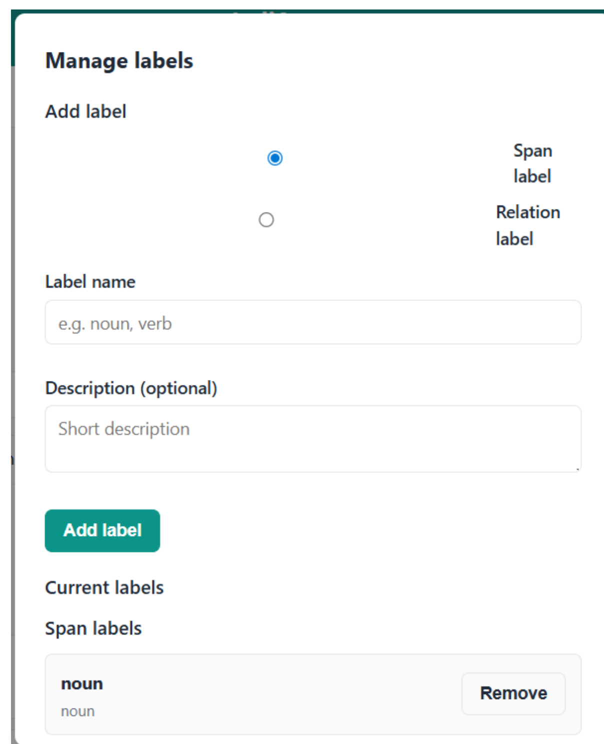

Figure A.2: A snapshot of label management provision

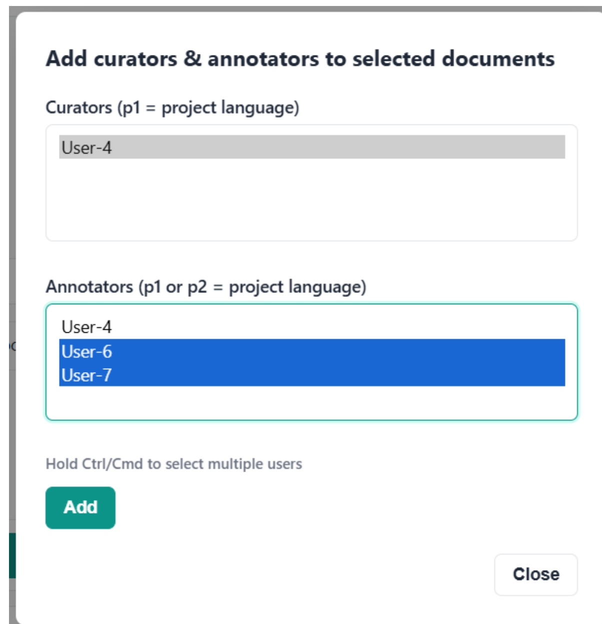

Figure A.3: A snapshot of role management provision

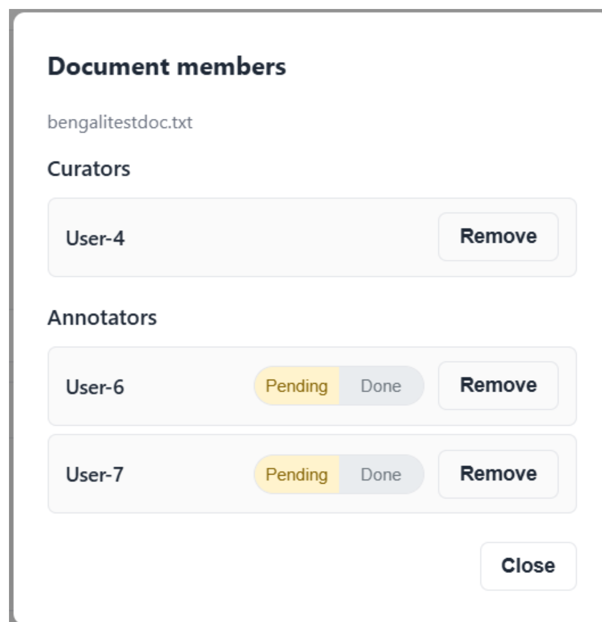

Figure A.4: A snapshot of lock management provision

B File Export Format


```
{
  "document_id": 10,
  "project_id": 4,
  "filename": "telugutestdoc.txt",
  "text": "రవి ఒక సాఫ్ట్వేర్ ఇంజనీర్ మరియు అతను ఇన్ఫోసిస్ కంపెనీలో పనిచేస్తాడు।\nఅతను భువనేశ్వర్
తయారు చేస్తారు।",
  "predicates": [
    {
      "id": 3,
      "predicate_label": "PRED",
      "predicate_text": "పనిచేస్తాడు।",
      "start_offset": 57,
      "end_offset": 69,
      "roles": [
        {
          "id": 5,
          "role_label": "arg0",
          "role_text": "రవి",
          "start_offset": 0,
          "end_offset": 3
        },
        {
          "id": 6,
          "role_label": "argm-loc",
          "role_text": "ఇన్ఫోసిస్",
          "start_offset": 38,
          "end_offset": 47
        }
      ]
    },
```


Figure B.5: Exported JSON file of an SRL-annotated document. The document is written in Telugu language.

```
1	রাহুল	রাহুল	propnoun	propnoun	_	_	_	_	start=0|end=5
2	একজন	একজন	_	_	_	_	_	_	start=6|end=10
3	সফটওয়্যার	সফটওয়্যার	noun	noun	_	_	_	_	start=11|end=21
4	ইঞ্জিনিয়ার	ইঞ্জিনিয়ার	noun	noun	_	_	_	_	start=22|end=33
5	এবং	এবং	conjuction	conjuction	_	_	_	_	start=34|end=37
6	সে	সে	pronoun	pronoun	_	_	_	_	start=38|end=40
7	একটি	একটি	_	_	_	_	_	_	start=41|end=45
8	বড়	বড়	_	_	_	_	_	_	start=46|end=49
9	কোম্পানিতে	কোম্পানিতে	_	_	_	_	_	_	start=50|end=60
10	কাজ	কাজ	verb	verb	_	_	_	_	start=61|end=64
11	করে।	করে।	verb	verb	_	_	_	_	start=65|end=69
12	সে	সে	pronoun	pronoun	_	_	_	_	start=70|end=72
13	কলকাতা	কলকাতা	_	_	_	_	_	_	start=73|end=79
14	শহরে	শহরে	_	_	_	_	_	_	start=80|end=84
15	থাকে	থাকে	_	_	_	_	_	_	start=85|end=89
16	এবং	এবং	_	_	_	_	_	_	start=90|end=93
17	প্রতিদিন	প্রতিদিন	adverb	adverb	_	_	_	_	start=94|end=102
18	অফিসে	অফিসে	noun	noun	_	_	_	_	start=103|end=108
19	যায়।	যায়।	_	_	_	_	_	_	start=109|end=114
20	তার	তার	_	_	_	_	_	_	start=115|end=118
21	বন্ধু	বন্ধু	_	_	_	_	_	_	start=119|end=124
22	অনিতা	অনিতা	_	_	_	_	_	_	start=125|end=130
23	একই	একই	_	_	_	_	_	_	start=131|end=134
24	কোম্পানিতে	কোম্পানিতে	_	_	_	_	_	_	start=135|end=145
25	ম্যানেজার	ম্যানেজার	_	_	_	_	_	_	start=146|end=155
26	হিসেবে	হিসেবে	_	_	_	_	_	_	start=156|end=162
27	কাজ	কাজ	_	_	_	_	_	_	start=163|end=166
28	করে।	করে।	_	_	_	_	_	_	start=167|end=171
29	তারা	তারা	_	_	_	_	_	_	start=172|end=176
30	একসাথে	একসাথে	_	_	_	_	_	_	start=177|end=183
31	নতুন	নতুন	_	_	_	_	_	_	start=184|end=188
32	প্রজেক্টে	প্রজেক্টে	_	_	_	_	_	_	start=189|end=198
33	কাজ	কাজ	_	_	_	_	_	_	start=199|end=202
34	করে	করে	_	_	_	_	_	_	start=203|end=206
35	এবং	এবং	_	_	_	_	_	_	start=207|end=210
36	গ্রাহকদের	গ্রাহকদের	noun	noun	_	_	_	_	start=211|end=220
37	জন্য	জন্য	_	_	_	_	_	_	start=221|end=225
38	সফটওয়্যার	সফটওয়্যার	_	_	_	_	_	_	start=226|end=236
39	তৈরি	তৈরি	verb	verb	_	_	_	_	start=237|end=241
40	করে।	করে।	verb	verb	_	_	_	_	start=242|end=246
```

Figure B.6: Exported CoNLL-U file of a POS annotated document. The document is written in Bengali language.

C Sample Snapshots of IndiAnn

This section contains images of documents annotated using POS tags for the supported languages.

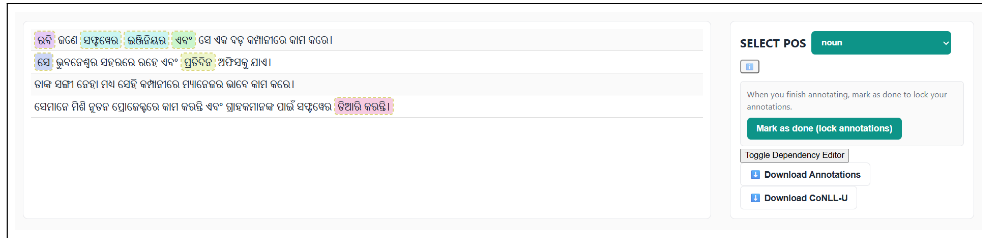

Figure C.7: POS Annotated Odia document

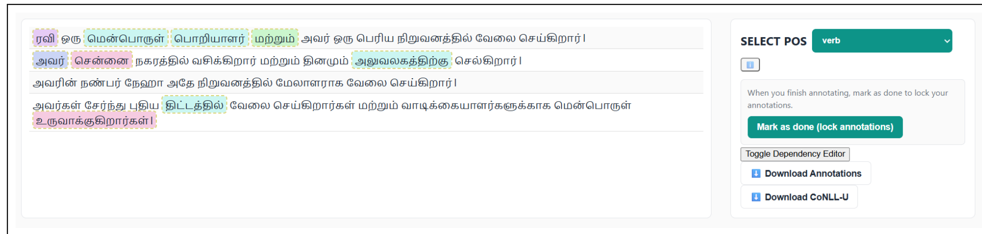

Figure C.8: POS Annotated Tamil document

Figure C.9: POS Annotated Bengali document

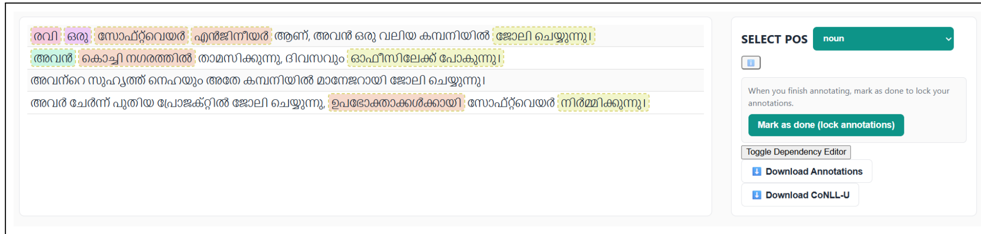

Figure C.10: POS Annotated Malayalam document

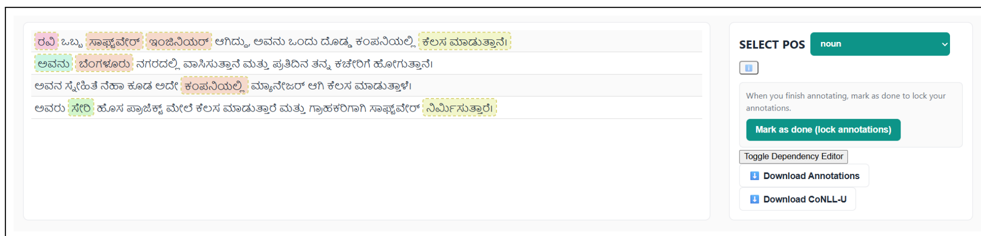

Figure C.11: POS Annotated Kannada document

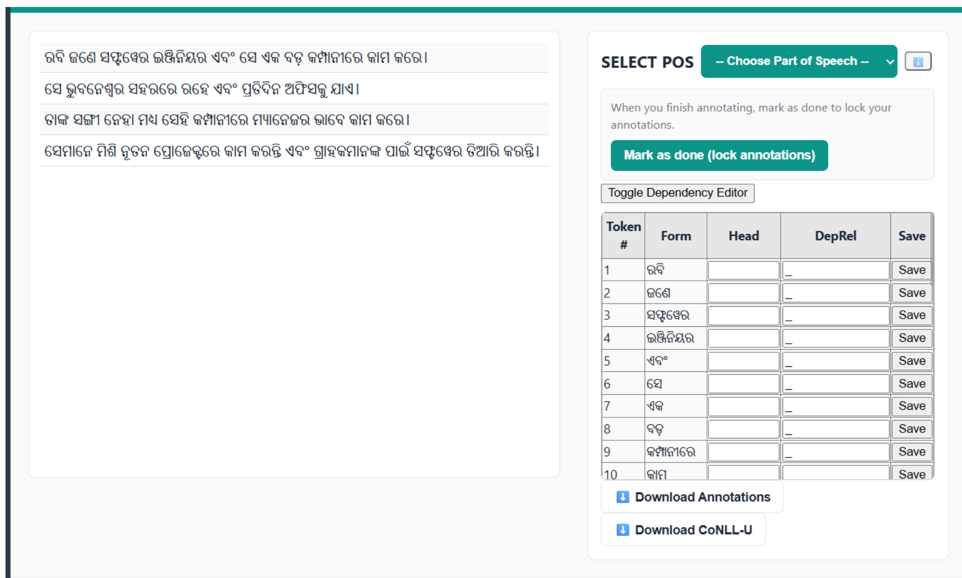

Figure C.12: Dependecy tagging of an odia Document