USR Builder: Tool for Automatic Generation of Universal Semantic Representation

Anonymous ACL submission

Abstract

This paper presents an integrated software **USR Builder** that automatically creates the multilayered Universal Semantic Representation (**USR**) from Hindi texts. The proposed software applies a set of heuristics on the outputs of various NLP tools to produce the multilayered semantic representation, USR, for a given discourse. Since manual annotation of any text data is always a labor-intensive, time-consuming, error-prone and expensive task, it is never a feasible option to manually annotate a text from scratch. USR Builder provides an automated generation option for USRs. It generates USR automatically, which the annotators can validate and correct to obtain the final version. The tool evaluation scores validate the claim that this tool saves both time and effort compared to starting the manual annotation process from scratch and improves the quality of annotation by reducing the chances of manual error.

1 Introduction

Semantic Representation is a foundational component in natural language processing (NLP) (Reiter, 2025), enabling machines to interpret and reason over human languages. One such representation, Universal Semantic Representation (USR), aims to abstract away from surface-level linguistic features and capture the underlying meaning in a language-agnostic manner. This can serve as an Interlingua for multilingual generation, a knowledge-rich resource for question-answering systems, and cross-lingual understanding. Manual semantic annotation requires a high level of linguistic expertise and is often time-intensive and costly. These limitations have led to a scarcity of large-scale, richly annotated semantic resources in many non-English languages.

To address this challenge, we present the USR Builder, a software system designed to assist in the semi-automated creation of USR datasets. The software integrates multiple NLP tools, including a Concept Identifier, Dependency Parser, Named Entity Recognizer, Morphological Analyzer, and Discourse Connective Marker, to automatically generate draft semantic representations. Annotators can then validate and refine these drafts, thus streamlining the annotation process and significantly reducing manual workload. This paper outlines the architecture and functionality of the USR Builder Tool, evaluates its performance for Hindi texts, and discusses its applicability in creating semantically annotated corpora.

2 Related Work

There exist web-based annotation tools specifically designed for annotating semantic resources. For example, Charon is used to annotate FrameNet data (Belcavello et al., 2022). UMR writer supports the creation of the graphical representation of texts in UMR (Uniform Meaning Representation) format (Zhao et al., 2021). UCCAApp (Abend et al., 2017) provides a web-application for syntactic and semantic phrase-based annotation for UCCA (Abend and Rappoport, 2013). To mitigate the cost of manual annotation, several systems have explored semi-automated annotation workflows. For example, the PMB pipeline consists of a sequence of NLP tools each serving for a specific annotation layer (Abzianidze et al., 2020). Studies have shown that such tools can significantly accelerate the annotation process while maintaining quality. Our work builds on this body of research by introducing a novel, domain-independent tool tailored for USR creation. By combining automatic NLP modules with human-in-the-loop validation, the USR Builder Tool bridges the gap between automation and accuracy in semantic dataset development.

3 Model Description

This section presents the architecture of the USR Builder followed by a comprehensive overview of the individual NLP modules integrated into the system. A complete workflow of the model is illustrated in Figure 1.

The workflow begins with the input sentences being processed by the Sentence Segmentor, a custom rule-based tool developed to break down complex sentences into simple, more manageable segments. Both automated parsers and human beings analyze simple sentences more accurately than complex ones. These segments are then simultaneously fed into four NLP modules: (a) the Dependency Parser and Mapper that determines syntactico-semantic structures by identifying POS tags, head words and generating dependency relations between the head and its children; (b) Morphological analyzer that provides detailed morphological information such as root forms, tense-aspect-modality (TAM), gender, number, and person (c) the Named Entity Recognition (NER) tool that identifies and classifies named entities present in each segment; and (d) the Discourse Connective Marker Tool that operates on the whole input text to detect discourse connectives and establish relationships between different segments.

All linguistic information obtained from the aforementioned NLP tools is then fed to two concept identifier modules: (a) the Complex Concept Identifier tool, which detects phrases with complex semantic information (see 3.4 for details) and (b) the Simple Concept Identifier tool that identifies atomic concepts and their associated grammatical features.

In the final stage, the outputs from all previous modules are passed to the Rule Applicator, which applies a predefined set of heuristics to integrate the linguistic and semantic information into the final USR format. The resulting USR captures the underlying semantics of the input text in a language-independent, human-readable and machine-interpretable format.

We now provide a detailed explanation of each individual NLP component integrated into the USR Builder

3.1 Dependency Parser and Mapper

3.1.1 Dependency Parser

Dependency Parser identifies grammatical relationships between words in a sentence by linking each word (dependent) to its syntactic head. For this work, we employed the Hindi ISC Parser, developed under the Indian Language Treebanking initiative (Begum et al., 2008). This parser is built on the Pāṇinian Dependency Grammar (PDG) framework, which is particularly suitable for morphologically rich and free word-order languages like Hindi (Bharati et al., 2006).

3.1.2 Dependency Mapper

Since some dependency labels (such as VMOD) attested by the ISC parser are underspecified, we have developed a mapper to map these relations to more semantically grounded labels.

3.2 Named Entity Recognition (NER) Tool

To identify named entities within each segment, we use IndicNER developed by AI4Bharat[1]. This model is specifically trained to perform named entity recognition for Indian languages, including Hindi. IndicNER has been fine-tuned on data from 11 Indian languages and benchmarked against both a human-annotated test set and several publicly available Indian NER datasets. On the Hindi test set, the model achieves an F1 score of 82.33 percent, demonstrating its effectiveness in accurately identifying named entities.

3.3 Discourse Relation Marker Tool

We developed a custom Discourse Relation Marker Tool to identify discourse connectives within the text segments and assign corresponding discourse relations. This tool plays a crucial role in the USR Builder by capturing the discourse-level connections between two or more segments, thus preserving coherence in the representation.
The tool has demonstrated a high accuracy of 94 percent on internal evaluation datasets, indicating its reliability in detecting and labeling discourse markers. By extracting these markers and their associated relations, the tool enhances the USR Builder's ability to reflect how different segments of a sentence—or multiple sentences—are semantically and logically connected.

3.4 Morph Analyzer

To extract detailed morphological information such as Tense-Aspect-Modality (TAM), root form, gender, number, and person, we employ the Apertium

[1]AI4Bharat is a research lab at IIT Madras dedicated to advancing AI technology for Indian languages and contributing to the field through open-source initiatives.

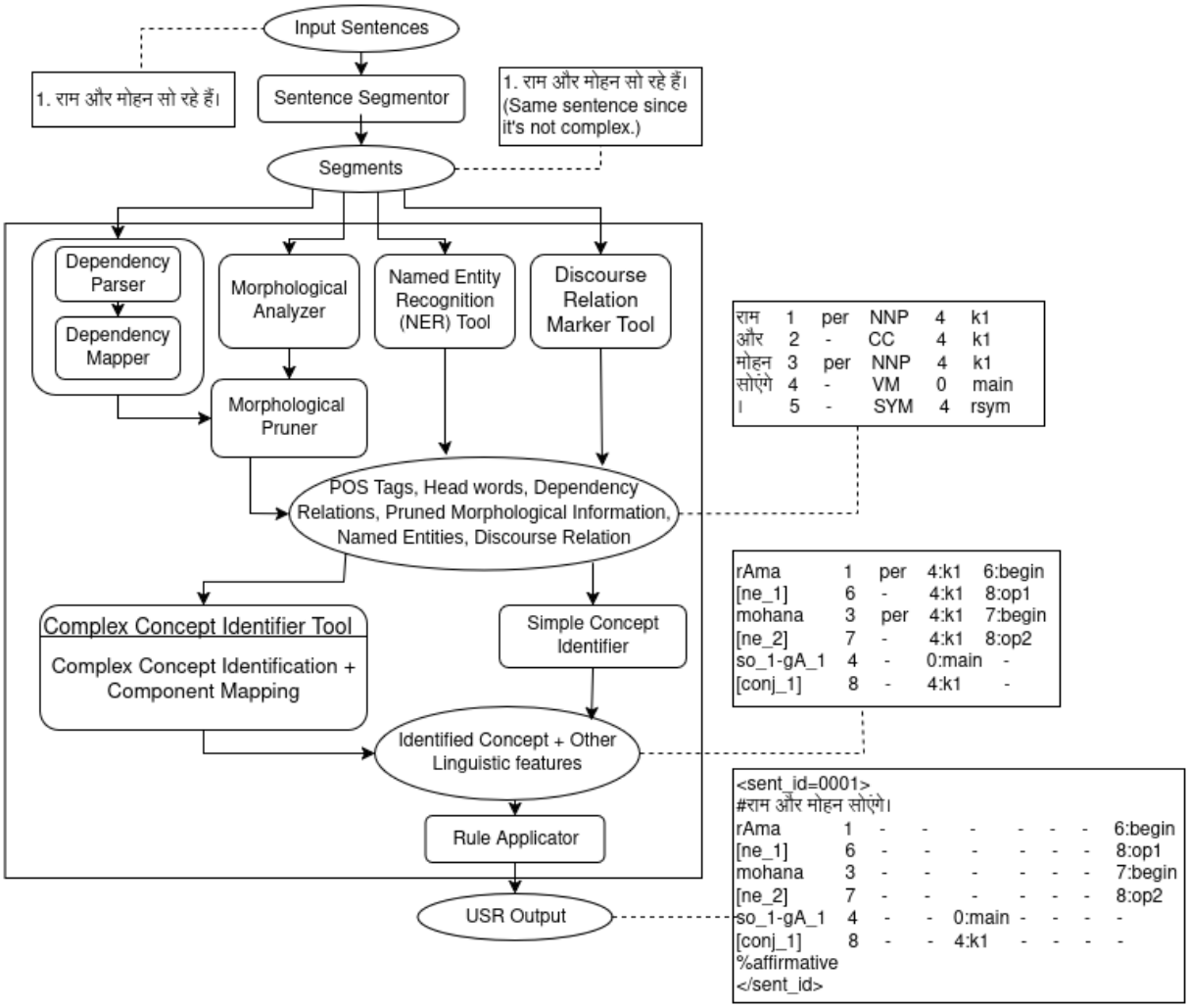

Figure 1: Complete Workflow of the USR Builder Tool

Hindi Morph tool. This is a monolingual language package specifically designed for Hindi, available as part of the open-source Apertium platform. It performs morphological analysis using a finite-state transducer (FST) approach, implemented via Lttoolbox (Forcada et al., 2011).

Due to its high accuracy, Hindi-specific design, and open-source availability, Apertium Hindi Morph is a key component of the USR Builder, ensuring reliable morphological analysis for generating semantically rich representations.

3.5 Complex Concept Identifier

To automatically identify complex concepts within the Hindi corpus, we have developed a rule-based Complex Concept Identifier Tool. This tool leverages the outputs from the Dependency Parser, Dependency Mapper, and Named Entity Recognition (NER) modules. By applying a set of predefined linguistic and syntactic rules to these outputs, the tool extracts instances of complex concepts such as rates (10 hours per day), measuring expressions (10 inches, 5 kg, 4 lt), calendric expressions (10th January 2025; 10th day of January, 2025), temporal (at 7 pm evening) along with their relevant components.

The Complex Concept Identifier has been evaluated on a manually annotated dataset and achieves an F1 score of 73.42%, demonstrating its effectiveness in capturing semantically complex constructs.

The overall workflow of the tool is illustrated in Figure 2

3.6 Simple Concept Identifier

The Simple Concept Identifier module is responsible for detecting atomic concepts within each segment, which is generally represented by a single word. This module processes the pruned output from the Morphological Analyzer and identifies relevant linguistic features for individual words, in-

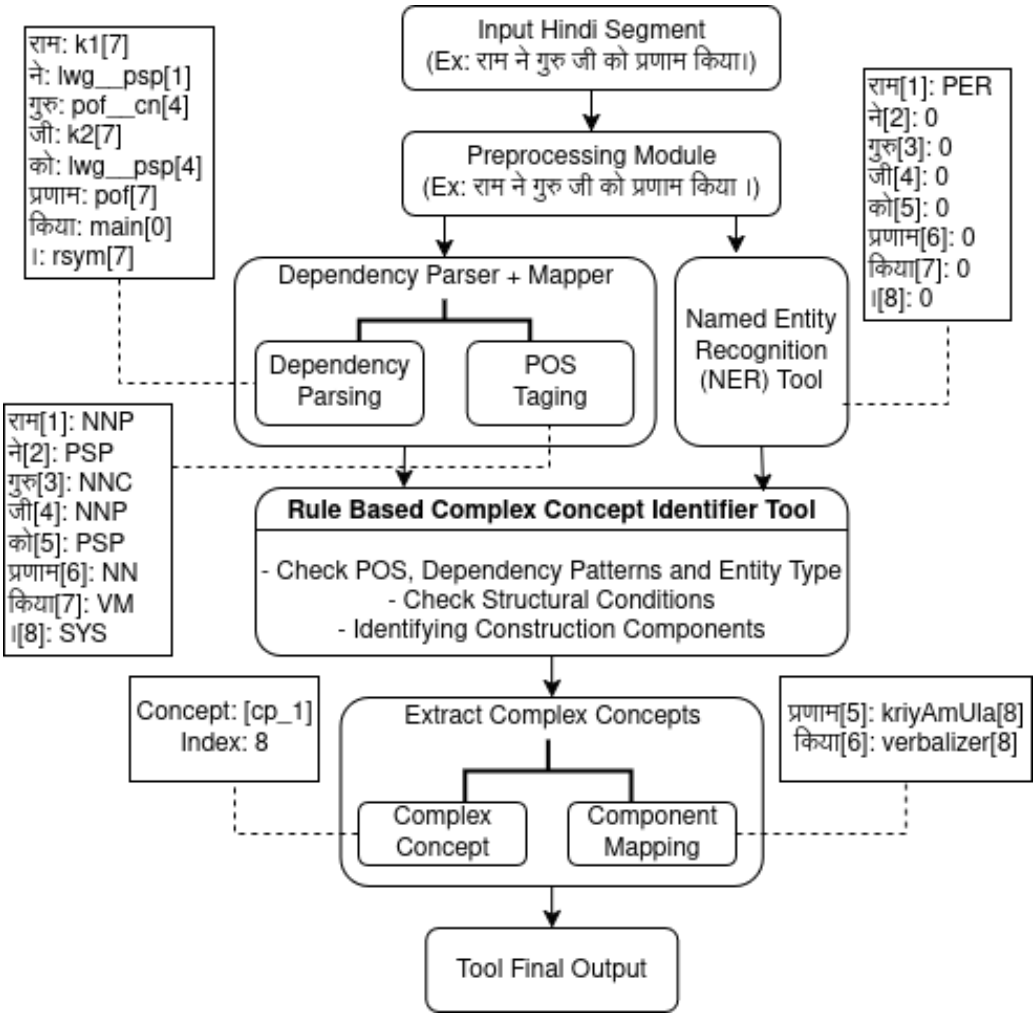

Figure 2: Workflow of Complex Concept Identifier Tool

cluding the Tense-Aspect-Modality (TAM) of finite verbs present in the segment. Accurate identification of TAM is essential for capturing the grammatical structure and temporal characteristics of the sentence within the USR framework. The identified simple concepts are then passed to the Rule Applicator for further processing.

3.7 Rule Applicator

The Rule Applicator filters information from the output of all aforementioned tools and organize them in the USR format for human evaluation and correction through a validation interface. USR is a 9-row feature-value matrix format with an XML tag delimiting the USR block. The features are specified below. The values of each feature are determined from the analysis performed on a given input sentence:

```
<sent_id=...>
concept
index
semantic_cat
morpho_semantic
dependency
discourse
speaker's view
scope
CC component
%sentence_type
</sent_id>
```

The most important task of the rule applicator is to postulate the legitimate concepts in the USR, both simple and complex. Root form represents the concept in USR. Not all words in a sentence are valid concepts in USR. For example, post-positions, auxiliaries, connectives, comparative and superlative degree markers, and discourse particles do not occur as concepts in USR. The semantics of these words are captured in other rows against the appropriate concept. For example, the tense-aspect-modality is marked on the verb. The causative and

degree of adjective (for -er and -est) are marked on the morphophonemics row for the verb and adjective, respectively. Dependency relations are attested on the dependent in the dependency row along with the head index. The semantics of the coordinate conjunction is assigned to the main verb of the sentence that has the connective along with the main verb index of the other conjoined sentence; whereas the semantics of the subordinate conjunction is assigned to the main verb of the subordinate clause along with the head index of the main clause. This is done to keep consistency in marking relations both at the dependency layer and the discourse layer. Here are two USRs generated by the Rule Applicator for the given small discourse:

(1) a. rAma aura mohana so rahe hEM.
Ram and Mohan sleep-3pr cont
"Ram and Mohan are sleeping."

b. lekina mili jagI huI hE.
but Mili awake be-pres
"But Mili is awake."

```
<sent_id=1>
#rAma Ora mohana so rahe hEM.
rAma                1  male  -  -  -  -  -  2:begin
[ne_1]              2  per   -  -  -  -  -  6:op1
mohana              3  male  -  -  -  -  -  4:begin
[ne_2]              4  per   -  -  -  -  -  6:op2
so_1-0_rahA_hE_1    5  -     -  0:main  -  -  -  -
[conj_1]            6  -     -  5:k1    -  -  -  -
%affirmative
</sent_id>
```

Figure 3: USR for segment 1a

```
<sent_id=2>
lekina mili jagI huI hE
mili          1  female  -  -  -  -  -  2:begin
[ne_1]        2  per     -  4:k1  -  -  -  -
jaga_1        3  -       krt  4:k1s  -  -  -  -
hE_1-pres     4  -       -  0:main  1.5:contrast  -  -  -
%affirmative
</sent_id>
```

Figure 4: USR for segment 1b

4 Experiments

To evaluate the effectiveness of the USR Builder Tool, we conducted experiments using datasets from the health domain. As part of the experimental setup, we first applied the workflow described in Section 3 to a set of input texts. After processing these texts through the Sentence Segmentor, we obtained a total of 260 segments, which served as the input for the USR Builder Tool.

The tool was then used to automatically generate USRs for these segments. Once the representations were generated, they were reviewed by human annotators, who were tasked with validating the semantic correctness and completeness of the output. These validated representations formed our gold-standard data, which we used to assess the tool's accuracy.

The evaluation methodology and results, based on comparison with the gold data, are discussed in the next section.

5 Evaluation

5.1 Layer-wise Evaluation

In this section, we perform a layer-wise evaluation of the USR Builder. A total of 260 segments were processed, and the automatically generated USRs were compared against a manually annotated gold standard. The evaluation was conducted across four core layers of the USR: Concept, Dependency Relation, Discourse, and Construction. The results are summarized in Table 1

Layer	Precision	Recall	F1-Score	Accuracy
Concept	1.000	1.000	1.000	1.000
Dependency	0.570	0.545	0.532	0.797
Discourse	0.627	0.523	0.560	0.667
Complex Concept	0.688	0.556	0.603	0.728

Table 1: Evaluation metrics across different layers

From the results, we observe perfect performance on the Concept layer, with 100% precision, recall, F1-score, and accuracy. This indicates that the USR Builder is highly effective in accurately identifying and representing both simple and complex concepts within each segment.

For the Dependency Relation layer, the system achieved a moderate F1-score of 0.532 and an accuracy of 79.7%. Although not perfect, these results suggest that the tool correctly captures a significant portion of syntactico-semantic relationships. This performance could potentially be improved by integrating more accurate or robust dependency parsers.

In the Discourse layer, the tool attained an F1-score of 0.560 and accuracy of 66.7%. Given the inherent complexity and variability of discourse connectives in natural language, these scores reflect a reasonable ability to detect discourse relations.

However, further refinements, particularly in identifying implicit or less frequent connectives, could improve both precision and recall.

The Complex Concept Component layer, which captures structural components of complex concepts (e.g., complex predicates and noun compounds), achieved an F1-score of 0.603 and accuracy of 72.8%. These results are promising and indicate the tool's effectiveness in recognizing common syntactic constructions, though improvements may still be made by expanding the rule base or refining feature extraction.

Overall, the evaluation demonstrates that the USR Builder performs exceptionally well in concept identification and delivers moderate to good performance across other semantic and structural layers. These results validate its practical utility for large-scale USR generation, particularly in low-resource language contexts, where manual annotation from scratch is both expensive and time consuming.

5.2 Complete USR Evaluation

We evaluated the performance of the USR Builder with a focus on two primary aspects: format consistency and efficiency in reducing manual annotation effort. Human annotators, however experienced, are prone to certain types of errors during the annotation process, especially in large-scale, complex datasets and using text editors. In contrast, the USR Builder consistently avoids such issues, delivering 100% accuracy in format-related tasks. Below are the key areas where the USR Builder outperforms manual annotation:

Format Consistency

- **Missing or Incorrect Cell Values:** Human annotators often omit required values in the USR's 9-column structure. In cases where a linguistic value is not applicable, a hyphen (-) must be used. Annotators occasionally forget to insert this, resulting in format violations. The USR Builder, however, enforces this rule strictly, ensuring every cell is properly populated or explicitly marked.

- **Index Duplication Errors:** Each concept in the USR must have a unique index. Manual annotations sometimes contain duplicate indices, leading to semantic ambiguity. The USR Builder automatically assigns and verifies unique indices, eliminating this type of error.

- **Incorrect Feature Formatting:** When multiple linguistic features are present in a single cell, they must be separated by a forward slash (/). Human annotators sometimes use incorrect separators such as hyphens or spaces, violating the USR format. The USR Builder follows the correct formatting standard consistently.

- **Incorrect Root Identification:** Annotators may struggle to correctly identify the root form of a word, especially in morphologically rich languages like Hindi. In contrast, the USR Builder ensures 100% accuracy by automatically generating concepts in proper WX notation and retrieving their correct root forms directly from the Morphological Analyzer, thereby eliminating human error in this aspect.

Reduction in Manual Effort

In addition to improving accuracy, the USR Builder significantly reduces annotation time and effort. A controlled experiment on 100 Hindi segments, with an average of 9 concepts per segment revealed the following:

- The USR Builder generated the complete USR representations in approximately 20 minutes.

- When the same task was performed manually from scratch, annotators required nearly 8.3 hours.

- When human annotators were asked to validate the system-generated output instead of annotating from scratch, it took only 2.5 hours.

This demonstrates a 5.8-hour time savings, highlighting the tool's potential for scaling dataset creation—especially in low-resource language contexts where manual annotation is expensive and time-consuming.

6 Error Analysis

While the USR Builder shows strong performance—particularly in concept identification—it exhibits some limitations in more linguistically complex layers, such as dependency relations, discourse marking, and Complex concept component detection. Below, we outline the key sources of errors observed during evaluation:

- **Dependency Layer Errors:** Errors in the dependency layer primarily stem from parser limitations. The dependency parser occasionally mislabels head–child relations in sentences with free word order or long-distance dependencies, which are common in Hindi. These inaccuracies propagate into the USR structure, affecting the semantic layer. Additionally, coordination and nested clauses often confuse the parser, resulting in incorrect dependency mappings.

- **Discourse layer Errors:** The identification of discourse relations showed moderate performance due to the implicit nature of many discourse connectives. The tool currently focuses on explicit markers, and may fail to capture coherence when connectives are inferred rather than stated. Additionally, the absence of a coreference resolution module limits the system's ability to fully annotate discourse-level phenomena, as coreference information is also crucial in the discourse layer.

- **Complex Concept Component Layer Challenges:** The construction layer includes complex predicates, noun compounds, and other multi-word expressions. Mistakes in this layer often occur when such expressions are non-contiguous or not easily distinguishable based on surface features alone. Furthermore, the rule-based nature of the Complex Concept Identifier means that edge cases—especially idiomatic or less frequent constructions—are not always captured correctly.

- **Manual Annotation Discrepancies:** Some of the performance discrepancies between system-generated and gold-standard data arise from inconsistencies at the level of decision-making during the preparation of the gold data. Variability in annotators' decisions regarding compound boundaries, complex predicate identification, and compound analysis may not always indicate system failure but rather subjectivity in the reference data.

7 Conclusion and Future Work

In this paper, we presented the USR Builder Tool, an automated system designed to generate high-quality Universal Semantic Representation (USR) datasets, particularly for low-resource languages such as Hindi. The tool integrates several natural language processing tools, including the sentence segmentor, dependency parser, morphological analyzer, named entity recognizer, and discourse analyzer, into a unified pipeline. It further incorporates custom modules like the Complex Concept Identifier, Simple Concept Identifier, and Rule Applicator to transform linguistic features into a structured, multi-layered USR format.

Through detailed layer-wise evaluation, we demonstrated that the USR Builder achieves perfect performance in concept identification and performs reasonably well in identifying dependency relations, discourse markers, and construction-level features. Compared to manual annotation, the system not only improves consistency and adherence to the USR format but also significantly reduces annotation time and human effort, making it a practical and scalable solution for building semantic datasets in linguistically diverse contexts.

The error analysis revealed that the primary limitations of the system lie in parsing complex syntactic structures, detecting implicit discourse relations, and handling non-standard or idiomatic expressions.

Overall, the USR Builder represents a significant step toward automating semantic annotation in low-resource settings, offering both speed and reliability. It can serve as a foundational tool for various downstream NLP tasks such as machine translation, information extraction, and question answering in Indian languages.

In the future, we aim to integrate a coreference resolution tool for Hindi into the USR Builder and replace the current parser with a more accurate dependency parser to further improve the overall performance and accuracy of the generated representations.

Limitations

Despite the effectiveness of the USR Builder Tool in generating Universal Semantic Representations, several limitations remain. One major challenge is the reliance on an existing dependency parser, which does not always yield accurate syntactic structures for complex or non-canonical sentences, thereby affecting the accuracy of downstream components like the Complex Concept Identifier. Additionally, the system lacks a coreference resolution module, which limits its ability to fully capture discourse-level phenomena, especially in texts where entity references span multiple segments.

The Discourse Marker Tool currently focuses only on explicit discourse connectives and struggles to handle implicit or inferred relations, which are common in natural language. The Construction Layer and many other components are based on manually defined heuristic rules, which, although effective in many cases, may not generalize well to unseen data or diverse linguistic structures.

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (ucca). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–238.

Omri Abend, Shai Yerushalmi, and Ari Rappoport. 2017. Uccaapp: Web-application for syntactic and semantic phrase-based annotation. In *Proceedings of ACL 2017, System Demonstrations*, pages 109–114.

Lasha Abzianidze, Rik Van Noord, Chunliu Wang, and Johan Bos. 2020. The parallel meaning bank: A framework for semantically annotating multiple languages. *arXiv preprint arXiv:2012.14854*.

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*.

Frederico Belcavello, Marcelo Viridiano, Ely Edison Matos, and Tiago Timponi Torrent. 2022. Charon: A framenet annotation tool for multimodal corpora. *arXiv preprint arXiv:2205.11836*.

Akshar Bharati, Rajeev Sangal, Dipti Misra Sharma, and Lakshmi Bai. 2006. Anncorra: Annotating corpora guidelines for pos and chunk annotation for indian languages. *LTRC-TR31*, pages 1–38.

Mikel L Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. 2011. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25(2):127–144.

Ehud Reiter. 2025. *Natural Language Generation*. Springer Nature Switzerland.

Jin Zhao, Nianwen Xue, Jens Van Gysel, and Jinho D Choi. 2021. Umr-writer: A web application for annotating uniform meaning representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 160–167.