INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

CL-1 Project

Building a Morphological Analyzer-Generator for Tamil,Hindi and English

Venya Velmurugan (2024114002)
Nikhita Ravi (2024114003)
Ananya Agarwal (2024114007)
IIIT Hyderabad

May 2024

Contents

Acknowledgments

We would like to extend our sincere gratitude to Parameswari Ma'am for her invaluable guidance and wisdom throughout the course of this project. Her expertise and insightful feedback have played a crucial role in shaping the development of this Tamil, Hindi, and English morphological analyzer and generator.

We also thank the International Institute of Information Technology, Hyderabad, for providing us with this opportunity to work on such an engaging and impactful project.

We are also incredibly thankful to Saketh Sir for his continuous support and dedication in helping us navigate challenges and deepen our understanding of computational linguistics. His guidance has been instrumental in enhancing our knowledge, especially in the area of morphological analysis.

A special note of thanks goes to all our friends and family for their unwavering encouragement, patience, and belief in us. Their constant support has been a great source of strength, making this journey both possible and enjoyable.

1 Introduction

1.1 Background and Motivation

Natural Language Processing (NLP) is a core subfield of Artificial Intelligence (AI) that enables machines to interpret, analyze, generate, and interact with human languages in a meaningful way. As NLP applications like machine translation, question answering, speech recognition, and sentiment analysis become integral to our digital ecosystem, the need for linguistically rich and computationally robust language tools has never been greater.

Morphological analysis is a foundational task in NLP that deals with the structure of words and how they can be broken down into morphemes—the smallest units of meaning. Morphological analysis involves identifying the root of a word, its part of speech, and its inflectional and derivational features (e.g., tense, case, number, gender). Morphological generation, on the other hand, reverses this process by synthesizing valid word forms from grammatical and lexical information. This process is particularly essential for morphologically rich languages, where a single root can give rise to hundreds or thousands of surface forms through affixation, compounding, and phonological alternation.

While English has received considerable attention in computational linguistics, Indian languages remain underrepresented due to their morphological complexity and the scarcity of annotated linguistic resources. This project seeks to bridge this gap by building a multilingual morphological analyzer and generator for Tamil, Hindi, and English—three linguistically and morphologically diverse languages spoken by billions of people across the Indian subcontinent and beyond.

1.2 Linguistic Diversity and Morphological Challenges

Tamil, an agglutinative language from the Dravidian family, forms words through the linear concatenation of morphemes, often resulting in long and morphologically dense word forms. For example, verbal morphology in Tamil encodes intricate distinctions of person, number, gender, tense, mood, and honorificity. Tamil also exhibits morphophonemic changes like sandhi and vowel harmony, making rule formulation and parsing particularly challenging.

Hindi, a highly inflectional language from the Indo-Aryan family, marks grammatical categories like case, number, gender, and tense through a combination of suffixes and postpositions. Unlike Tamil, Hindi features internal stem alternations and agreement between verbs and their subjects or objects, which must be systematically captured in any morphological framework. Furthermore, Hindi employs compound verbs, ergative constructions, and split auxiliary systems that pose additional hurdles to accurate morphological analysis.

English, while relatively morphologically poor, presents its own set of challenges. Irregular plural forms (e.g., "mice" vs. "cats"), irregular verb conjugations (e.g., "go" $\rightarrow$ "went"), and phrasal verbs contribute to the need for systematic morphological processing. Although many high-resource NLP tools already exist for English, incorporating it into a multilingual system highlights the contrast in complexity and enables the development of generalized approaches that can adapt to different morphological typologies.

1.3 Objectives and Contributions

The overarching aim of this project is to develop a robust, scalable, and linguistically grounded morphological analyzer and generator for Tamil, Hindi, and English. The system will be capable of:

- Parsing surface forms into root words and associated morphological features such as case, number, gender, person, tense, mood, and aspect.
- Generating valid surface forms from root lexemes and grammatical specifications.
- Modeling language-specific morphotactics (rules governing morpheme ordering) and morphophonemics (sound changes triggered by affixation).
- Handling out-of-vocabulary (OOV) and unseen word forms using data-driven techniques.
- Integrating both rule-based finite-state transducers (FSTs) and statistical approaches to ensure precision and coverage.

This hybrid approach not only enables fine-grained control through handcrafted linguistic rules but also leverages the generalizability and scalability of modern NLP models, making the system resilient across formal and informal language domains.

1.4 Scope and Applications

The system's scope extends to the creation of extensive lexicons for all three languages. Each lexicon will be annotated with grammatical categories and transformation rules that define the morphological behavior of root words. In Tamil, the system must cover complex verb inflections, nominal declensions, and extensive honorific and rationality features. For Hindi, it must handle gendered agreement, compound verb constructions, and postpositional syntactic dependencies. In English, attention will be paid to irregular morphology, auxiliary constructions, and periphrastic expressions.

Applications of the analyzer and generator include but are not limited to:

- **Machine Translation:** Accurate morphological analysis ensures correct alignment and transfer of grammatical features between source and target languages.
- **Information Retrieval:** Stemming and lemmatization based on morphological analysis improve search relevance.
- **Speech and Text Processing:** Text-to-speech (TTS), speech-to-text (STT), and grammatical error correction systems require morphological synthesis.
- **Educational Technology:** Language learning applications can use morphological generation to create personalized drills, inflection tables, and exercises.

1.5 Data Resources and Feasibility

To support the development and evaluation of the system, the following linguistic datasets and frameworks are utilized. These resources not only provide rich morphological information but also ensure compatibility with widely accepted standards in computational linguistics.

- **Universal Dependencies (UD) Treebanks:** The Universal Dependencies project offers standardized, cross-linguistic treebanks that include detailed linguistic annotations such as part-of-speech (POS) tags, morphological features, and syntactic dependencies. The data is formatted in the `CoNLL-U` (Conference on Natural Language Learning - Universal) format, a tab-separated plain-text structure that facilitates easy parsing and programmatic access. Each line in a CoNLL-U file represents a word or token and includes ten fields:
    1. **ID:** Token index in the sentence.
    2. **FORM:** The surface word form.
    3. **LEMMA:** The base or dictionary form of the word.

4. **UPOS:** Universal part-of-speech tag (e.g., NOUN, VERB).
5. **XPOS:** Language-specific POS tag.
6. **FEATS:** Morphological features (e.g., Gender=Masc—Number=Sing).
7. **HEAD:** Syntactic head of the current token.
8. **DEPREL:** Dependency relation to the head.
9. **DEPS:** Enhanced dependency graph (optional).
10. **MISC:** Miscellaneous information (e.g., SpaceAfter=No).

The key aspects of Universal Dependencies (UD) datasets which make it useful include:

1. **Cross-linguistic consistency:** UD establishes a standard that works across languages with very different grammatical structures.
2. **Three-layer annotation:** Tokenization and word segmentation, Morphological features and part-of-speech tags and Syntactic dependencies between words are all covered in a UD dataset.
3. **Common format:** All treebanks use the CoNLL-U format, a standardized tab-separated text format.
4. **Support and inclusion for multiple languages:** UD encompasses over 200 treebanks in more than 100 languages, making it the largest multilingual collection of annotated texts with a consistent annotation scheme.

These treebanks serve as a gold-standard benchmark for evaluation and provide the necessary morphological diversity to extract lemmas, features, and surface forms for building and testing the analyzers.

The UD datasets used for our project are:

1. **UD_Hindi-HDTB:** Hindi Dependency Treebank, one of the larger Hindi resources
2. **UD_Hindi-PUD:** Parallel Universal Dependencies corpus for Hindi (contains parallel texts across many languages)
3. **UD_Tamil-MWTT:** Tamil Modern Written Treebank
4. **UD_Tamil-TTB:** Another UD Tamil Treebank
5. **UD_English-EWT:** English Web Treebank, one of the foundational English UD resources
6. **UD_English-GUM:** Georgetown University Multilayer corpus
7. **UD_English-PUD:** Parallel Universal Dependencies corpus for English
8. **UD_English-Atis:** Based on the ATIS (Airline Travel Information System) dataset

9. **UD_English-CTeTex:** Clinical Trial EndNote Texton corpus
10. **UD_English-ESLSpok:** English as a Second Language spoken corpus
11. **UD_English-GENTLE:** Likely a simplified or "gentle" version for learning purposes
12. **UD_English-GUMReddit:** Reddit data subset from the Georgetown University Multilayer corpus
13. **UD_English-LinES:** English side of a parallel English-Swedish corpus
14. **UD_English-ParTUT:** Parallel TUT corpus (from Turin University)
15. **UD_English-Pronouns:** Specialized treebank focusing on pronoun usage

- **Custom Corpora:** In addition to UD datasets, we use domain-specific corpora such as Wikipedia dumps, educational textbooks, and journalistic articles to improve the system's robustness and generalizability. These corpora are annotated similar to the CoNLL-U format but with only the word, lemma, pos tag and morphological features.

By leveraging both standardized treebanks and custom-curated resources, our system balances linguistic accuracy with practical applicability, ensuring that the morphological analyzer performs well across diverse registers and real-world usage scenarios.

1.6 Significance

This project aims to fill a critical void in multilingual NLP by addressing the need for robust morphological analysis tools tailored to Indian languages. While English-centric NLP tools have seen widespread adoption, the lack of high-quality, open-source resources for Tamil and Hindi hampers progress in inclusive technology development. By combining linguistic rigor with computational efficiency, this system lays the groundwork for future research in cross-lingual morphological modeling and opens doors to more equitable language technologies.

The broader impact includes enhancing digital inclusion for native speakers of underrepresented languages, supporting government and educational language initiatives, and contributing to the global repository of open-source NLP tools.

2 Literature Review

2.1 Finite-State Transducers in Morphological Analysis

Finite-State Transducers (FSTs) have been widely adopted for morphological analysis in morphologically rich languages due to their efficiency in handling concatenative and non-concatenative processes. For Tamil, multiple implementations demonstrate the versatility of FST frameworks. [4] introduced *ThamizhiMorph*, an open-source morphological analyzer and generator (MAG) built using Foma. Their approach leverages FSTs to model Tamil's complex morphophonological rules, enabling bidirectional analysis and generation. The system incorporates Meta-Morph rules, a high-level abstraction layer, to simplify the encoding of inflectional paradigms and orthographic variations, thereby enhancing maintainability.

In contrast, [3] utilized the Apertium platform, which employs a "Word and Paradigm" model. This approach compiles lexical databases into FSTs using XML-based specifications. While effective, the Apertium implementation faced challenges in handling external Sandhi and colloquial forms, necessitating pre-processing steps like Sandhi splitting. Similarly, [2] adopted the Xerox Finite-State Toolkit (XFST) to model Tamil nouns and verbs. Their transliteration scheme addressed Unicode limitations in XFST, but the system's reliance on ASCII encoding introduced scalability issues for modern applications.

Cross-linguistic insights from Hindi morphology further validate FST-based approaches. [1] developed a paradigm-driven MAG using a database-backed FST, prioritizing speed and accuracy over storage efficiency. This aligns with ThamizhiMorph's design philosophy, where finite-state operations are augmented with extensible lexicons to balance coverage and performance.

2.2 Challenges in Tamil Morphology

Tamil's agglutinative structure and morphophonemic alterations pose significant challenges for computational models. A primary issue is *Sandhi*, which governs phonological changes at morpheme boundaries. [3] highlighted the need for Sandhi splitters to handle conjoined words like *veNumAnYAlum* ("need+though"), which are pervasive in written texts. ThamizhiMorph addressed this by integrating internal Sandhi rules directly into its FST, eliminating the need for external modules. For instance, euphonic increments (*an/in*) and consonantal glides are modeled through orthographic rules applied during transducer compilation [4].

Orthographic variations and dialectal differences further complicate analysis. [2] reported inconsistencies in handling verb forms like *porYAnY* (colloquial "he is going") versus *pokirYAnY* (standard form). ThamizhiMorph's evaluation on the Universal Dependencies (UD) Tamil treebank revealed similar issues, with 14% of errors stemming from spelling variations in Indian vs. Sri Lankan Tamil [4]. These findings underscore the necessity of dialect-aware rules and robust normalizers, as implemented in ThamizhiMorph's pre-processing pipeline.

Low-resource constraints exacerbate these challenges. While [3] relied on corpora like EMILLI CIIL (4.8 million tokens), data scarcity persists for specialized domains. ThamizhiMorph mitigated this by curating lexicons from diverse sources, including classical literature and modern textbooks, ensuring broader coverage [4].

2.3 Paradigm-Based Approaches

Paradigm-driven methodologies are central to modeling Tamil's inflectional morphology. ThamizhiMorph classified verbs into 12 conjugational classes based on tense markers and irregular forms, while nouns were categorized into 38 declensional classes [4]. This granularity enabled precise handling of stem mutations, such as the insertion of *nt* in Class 12 verbs (e.g., *nata* $\rightarrow$ *natantaan* "he walked"). The system's Meta-Morph rules abstracted these patterns into JSON-based templates, automating the generation of FST lexicons.

Similarly, [2] grouped nouns into seven classes based on terminal phonemes (e.g., "m"-ending stems) and verbs into 11 classes. Their XFST implementation used regular expressions to encode suffixation rules, but manual intervention was required to resolve over-generation in complex verbs. In contrast, Apertium's paradigm linking [3] allowed derived forms (e.g., *patikkirYavanY* "one who reads") to inherit inflectional properties from base lexemes, reducing redundancy.

The Hindi MAG [1] adopted a similar paradigm approach, storing all inflected forms in a relational database. While this incurred storage overhead, it ensured rapid lookup—a trade-off mirrored in ThamizhiMorph's design, which prioritizes computational efficiency through FST optimization.

2.4 Evaluation and Performance

Evaluating morphological analyzers requires diverse datasets and rigorous metrics. ThamizhiMorph achieved 93.3% coverage on a textbook corpus and 84.7% accuracy on the UD Tamil treebank, with errors primarily due to out-of-vocabulary items [4]. The IIIT Hyderabad shallow parser, in contrast, achieved 84% coverage on the EMILLI corpus but struggled with orthographic variations [**?**]. [2] reported 78.3% accuracy for nouns in their XFST model, though verb analysis was limited by toolkit constraints.

Notably, ThamizhiMorph's open-source lexicons and rule sets facilitate community-driven improvements, addressing a key limitation of prior tools like Anna University's *Atcharam*, which lacked documentation [4]. The integration of a morphological guesser further enhanced robustness, enabling partial analyses for unseen words—a feature absent in earlier systems.

2.5 Open Source Tools and Resources

The proliferation of open-source tools has significantly advanced Tamil NLP. ThamizhiMorph's release of FST models, lexicons, and test datasets under a CC-BY license sets a precedent for collaborative development [4]. In contrast, proprietary tools like XFST limit extensibility, as noted by [2], who advocated migrating to open platforms like Foma. Apertium's modular architecture [3] also supports extensibility but requires XML expertise, posing barriers for non-technical users.

Community engagement remains critical. The Tamil UD treebank's annotation errors, identified during ThamizhiMorph's evaluation, highlight the need for standardized resources [4]. Similarly, [1] emphasized database-driven approaches to streamline Hindi-Punjabi machine translation, a strategy applicable to Tamil's low-resource context.

2.6 Cross-Language Insights

Lessons from Hindi morphology inform Tamil NLP. [1]'s paradigm-driven database, which stored all inflected forms, inspired ThamizhiMorph's lexicon design. However, Tamil's richer inflectional system (e.g., 260 verb forms vs. Hindi's 48) necessitates more sophisticated rule engines. The use of finite-state operations for Arabic [2] and Urdu [1] further validates FSTs' suitability for Dravidian languages, despite challenges in handling non-concatenative processes.

2.7 Conclusion

The reviewed works underscore FSTs' efficacy in modeling Tamil's morphological complexity. ThamizhiMorph's innovative use of Meta-Morph rules and open resources addresses longstanding gaps in Tamil NLP, while insights from Apertium and XFST implementations highlight the trade-offs between usability and power. Future efforts must prioritize dialect adaptation, standardized evaluation metrics, and community-driven resource creation to fully unlock Tamil's computational potential.

References

[1] Vishal Goyal and Gurpreet Singh Lehal. Hindi morphological analyzer and generator. In *First International Conference on Emerging Trends in Engineering and Technology*, pages 1156–1159. IEEE, 2008.

[2] S Lushanthan, A R Weerasinghe, and D L Herath. Morphological analyzer and generator for tamil language. In *2014 International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 190–196. IEEE, 2014.

[3] K Parameshwari. An implementation of apertium morphological analyzer and generator for tamil. *Language in India*, 11(5):41–44, 2011.

[4] Kengatharaiyer Sarveswaran, Gihan Dias, and Miriam Butt. Thamizhimorph: A morphological parser for the tamil language. *Machine Translation*, 35(3):37–70, 2021.

3 Design and Implementation

3.1 Corpus Preparation and Preprocessing

The system takes as input Universal Dependencies (UD) formatted treebanks in the `.conllu` format. These annotated corpora consist of syntactically parsed sentences with rich linguistic annotations, including tokenized words, lemmas, universal part-of-speech (UPOS) tags, morphological features (stored in the `FEATS` column), and dependency information.

To ensure systematic and reliable parsing of these files, we employ the Python `conllu` library, which provides an object-oriented interface to extract structured linguistic data from `.conllu` files. Each sentence is parsed into a list of token dictionaries, with each token containing keys such as `‘‘form’’`, `‘‘lemma’’`, `‘‘upos’’`, and `‘‘feats’’`. This allows us to access word-level annotations programmatically and uniformly across languages.

The data pipeline performs the following steps:

- Recursively loads all `.conllu` files from a dataset directory, filtering by language name (e.g., “Hindi”, “Tamil”, or “English”).
- Parses each file sentence-by-sentence using `parse_incr()` from the `conllu` module to avoid memory overload.
- Extracts token-level tuples of the form (`form`, `lemma`, `feats`), which are then used for lexicon and paradigm construction.

To eliminate noisy tokens and non-linguistic artifacts, we apply several normalization steps:

- **Removal of zero-width joiner characters** (`\u200d`): These invisible Unicode characters often occur in complex scripts like Tamil and Devanagari and can cause false mismatches in string comparisons.
- **Filtering out tokens containing special symbols or punctuation:** Tokens that consist purely of punctuation marks, numerals, or special characters (e.g., `#`, `%`, `$`) are discarded to focus on morphologically relevant units.
- **Unicode normalization:** Tokens are optionally normalized using standard Unicode forms (e.g., NFC or NFD) to ensure consistency in encoding, especially for scripts involving diacritics and ligatures.

These preprocessing steps ensure that only clean, linguistically meaningful data is used to derive morphological patterns and build finite-state transducers (FSTs). By leveraging the standardized CoNLL-U format and the robust parsing capabilities of the `conllu` library, the system maintains consistency, extensibility, and language-independence throughout the training pipeline.

3.2 Lexicon Extraction

After parsing the `.conllu` files, we extract a structured lexicon that maps lemmas to their observed surface forms and associated morphological features. The lexicon is organized hierarchically by part-of-speech (POS) tags and is stored as a nested Python dictionary of the form:

```
lexicon[POS][lemma] = [(form, FEATS)]
```

This structure supports efficient access to all inflectional forms of a lemma within a specific grammatical category.

The extraction pipeline performs the following tasks:

- **Symbol filtering:** Tokens containing non-linguistic symbols (e.g., punctuation, special characters) are excluded using an explicit character blacklist. This ensures that only morphologically valid forms contribute to paradigm formation.
- **Lexicon population:** For each valid token, we record its part of speech, lemma, surface form, and feature bundle. Multiple distinct surface forms (e.g., tense or number variants) may map to the same lemma and are stored as tuples.
- **Duplicate suppression:** Redundant (`form`, `feats`) pairs are avoided by checking for existing entries before insertion.

Finally, the resulting lexicon is serialized into a CSV file named `lexicon.csv`, which includes three columns: **POS**, **Lemma**, and **Forms with Feats**. Each row captures all observed morphological variants of a given lemma within a part-of-speech category. This output serves as the foundation for paradigm induction and morphological rule extraction in the subsequent stages of the pipeline.

3.3 Paradigm Induction

The next stage involves transforming this lexicon into a set of morphological paradigms. For each pair of `(form, lemma)`, we compute a pair of strings:

- **Remove**: the suffix in the surface form not present in the lemma.
- **Add**: the suffix in the lemma not present in the surface form.

This character-level transformation is computed by finding the longest common prefix between the form and the lemma. The remainder is then encoded as the transformation rule for inflection.

Each transformation is associated with the lemma's POS tag and morphosyntactic features (e.g., `Number=Sing`, `Gender=Masc`, `Tense=Pres`). These are stored in a dictionary where each unique transformation group (set of rules) is mapped to a list of lemmas that share it. We then invert this dictionary to produce a mapping from lemma sets to transformation groups. This results in a reduced and generalized inventory of inflectional paradigms.

3.4 Paradigm Induction

Paradigm induction is a critical step in the construction of a rule-based morphological analyzer and generator. Rather than storing each inflected form independently, the goal is to group lemmas that follow identical morphological behavior into clusters called **paradigms**. Each paradigm represents a reusable set of transformation rules that describe how to derive surface forms from lemmas using suffixation and morphological feature combinations. This abstraction allows for efficient dictionary construction, generalization to unseen words, and compact representation.

To induce transformation rules, the system compares each lemma with its surface forms and identifies the longest common prefix. The suffix of the surface form that differs from the lemma is marked as the portion to **remove**, while the suffix present in the lemma but absent from the form is marked as the portion to **add**. For example, for the English verb `walked` and its lemma `walk`, the transformation is Remove: `ed`, Add: .

Each transformation is stored as a tuple: `(POS, remove, add, feats)`, where `feats` is a sorted tuple of morphological feature key-value pairs. These transformation tuples are then grouped by lemma. After collecting all such tuples, the system inverts the mapping: instead of mapping each lemma to its transformations, it maps each **set of transformations** to the group of lemmas that share it. This process yields the final list of paradigms.

Each paradigm is associated with multiple lemmas that share a common set of transformation rules. These are saved in a dictionary format where each entry is of the form `"lemma1 lemma2 ...": [(POS, remove, add, feats), ...]`. This file, stored as `lemma_paradigms.json`, serves as the input for dictionary generation in Apertium's XML format.

The system also prints statistics such as the total number of unique paradigms and total number of lemmas. These paradigms are central to the design of the analyzer: rather than writing separate rules for each word, a paradigm can be encoded once and reused for all associated lemmas using the `<par n="..."/>` tag in the Apertium `.dix` file. This results in both compactness and consistency.

By automating the induction of paradigms from annotated corpora, the system reduces the burden of manual rule writing and improves scalability. It also allows the analyzer to generalize better to new lemmas and to support morphological generation, where the same paradigms can be used in reverse to generate surface forms from grammatical features.

3.5 Apertium Dictionary Generation

After extracting morphological paradigms, the final step in building a rule-based morphological analyzer is to compile these paradigms into an XML dictionary format compatible with the Apertium platform. This format, known as the `.dix` file, defines the analyzer's finite-state transducer (FST) and encodes lemmas, transformation rules, and grammatical features in a structured and reusable format.

The Python script responsible for this transformation begins by loading the paradigm mappings from `lemma_paradigms.json`. Each entry in this file maps a group of lemmas to a shared set of transformation rules. These transformation rules are tuples of the form `(POS, remove, add, feats)`, where `feats` is a list of morphological feature key-value pairs (e.g., `Number=Sing`).

First, a comprehensive **alphabet** section is initialized. It defines the complete set of Unicode characters used in the respective language, including all vowels, consonants, modifiers, and special signs. This ensures that Apertium's compiler correctly interprets all graphemes used in the dictionary.

Next, the script iterates over each lemma group and its associated transformation rules:

- Each rule is converted into an `<e>` entry under a new `<pardef>` block, which describes how the suffixes are replaced during analysis and generation.
- Each rule also includes one or more `<s n="key=value"/>` tags that denote the morphological features associated with that transformation (e.g., `<s n="Number=Sing"/>`).
- The POS tag is also encoded as a feature (e.g., `<s n="pos=NOUN"/>`).

To avoid redundancy, the script maintains a growing list of all unique features declared in the `<sdefs>` section. This section acts as a schema for all morphological features used in the dictionary and ensures consistency across entries.

Each group of lemmas is associated with a uniquely named paradigm (e.g., `0_paradigm`, `1_paradigm`, etc.). Then, for each lemma, a corresponding `<e>` entry is added under the `<section id="main">` block, which links the lemma to its paradigm using the `<par n="..."/>` tag. This structure allows a single paradigm definition to be reused by many lemmas.

Finally, the XML content is assembled into a complete `.dix` document, containing:

- A DOCTYPE declaration for Apertium's dictionary DTD.
- The `<alphabet>` section listing all Tamil characters.
- The `<sdefs>` section defining all POS and morphological features.
- The `<pardefs>` section listing all paradigms and their rules.
- The `<section id="main">` block containing lemma entries.

The dictionary is written to a file named `morph_analyser.dix`, which can then be compiled using Apertium's `lt-comp` tool to generate a binary analyzer or generator. This file provides the complete mapping between surface forms and their underlying morphological structure, and can be reversed using `lt-proc -g` for generation tasks.

The use of automatically induced paradigms, combined with programmatically generated Apertium XML, enables rapid bootstrapping of morphological analyzers with minimal

3.6 Evaluation Framework for Morphological Analyzers

To systematically assess the quality and effectiveness of the morphological analyzers built using Apertium, a custom Python-based evaluation pipeline was developed. This script provides comprehensive functionality for reading annotated test datasets, running analyses using the compiled finite-state transducer (FST), and computing evaluation metrics such as accuracy, precision, recall, and F1-score.

1. Test Data Loading:
The script accepts evaluation data in multiple formats:

- `.conllu`: The Universal Dependencies format is parsed to extract word forms, lemmas, and morphological feature annotations. These are converted into Apertium-style analysis strings (e.g., `lemma<POS><Feature1><Feature2>`).
- `.txt`: For English datasets formatted as tab-separated values, the script extracts tokens, lemmas, POS tags, and feature values while ignoring unspecified fields (marked with '*').
- Generic TSV: Tab-separated files with at least two columns are also supported, interpreting the first column as the input word and the second as the expected analysis.

2. Morphological Analysis using lt-proc:
Each word is passed to the Apertium analyzer using `lt-proc`, which returns

the surface analysis. The script extracts the part after the '/' symbol from the analyzer's output (e.g., `ŵord/lemma<tag1><tag2>$`) and normalizes it to match the expected format. The normalization step also transforms any POS tags represented as `<pos=TAG>` into `<TAG>` to ensure consistent evaluation.

3. Tag-Based Comparison:
The script evaluates each prediction by decomposing both the predicted and expected strings into a set of tags (e.g., lemma, POS, features). It then calculates:

- **True Positives (TP):** Tags present in both prediction and target.
- **False Positives (FP):** Tags predicted but not present in the target.
- **False Negatives (FN):** Tags in the target but missing from the prediction.
- **True Negatives (TN):** Implied by tags not appearing in either, based on the tag universe.

This enables a fine-grained evaluation of morphological tag prediction quality.

4. Metric Calculation:
Using standard formulas, the script computes global precision, recall, and F1-score from TP, FP, and FN counts:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad \text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Additionally, overall **accuracy** is computed as the proportion of correct predictions (TP + TN) to total tag slots.

5. Per-Tag Metrics:
Beyond aggregate metrics, the script also maintains separate TP, FP, and FN counts for each individual morphological tag. This enables identifying the most problematic features (e.g., tense, gender, case) and guides further refinement of the analyzer.

6. Output Generation:
All results are saved to a specified directory in the following formats:

- `metrics.json` — Overall accuracy, precision, recall, F1 score.
- `tag_metrics.json` — Per-tag precision, recall, and F1.
- `predictions.txt` — Side-by-side prediction vs. target for each word.
- `error_analysis.txt` — A filtered version highlighting incorrect predictions.

7. Optional FST Compilation:
If the user passes the `--compile` flag or if no compiled `.bin` file exists, the script automatically compiles the `.dix` dictionary into an FST using:

```
lt-comp lr morph_analyser.dix morph.bin
```

8. Command Line Interface:
The script supports the following arguments:

- `--language`: Language name (used to locate resources).
- `--test_file`: Path to evaluation dataset.
- `--compile`: Whether to compile the .dix file.
- `--dix_path` and `--bin_path`: Optional custom paths to FST resources.
- `--output_dir`: Where to store evaluation outputs.

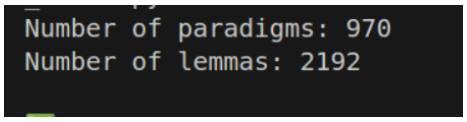
4 Challenges

During the course of building and evaluating the morphological analyzers and generators for Tamil, Hindi, and English, we noted several significant observations that influenced both our understanding of the languages and the computational challenges involved in modeling their morphology. These observations fall into three broad categories: linguistic, technical, and evaluative.

4.1 Linguistic Challenges

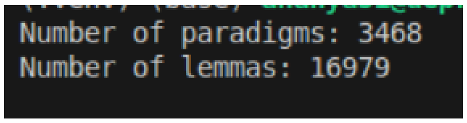
- **Morphological richness varies greatly across languages:**

  1. Tamil exhibited a high level of agglutination with consistent morpheme concatenation. This caused the script to generate a larger number of paradigms for Tamil as it is impossible for the dataset to cover all possible word forms of a word containing different features. As the model grouped words based on similarity in features, Tamil ended up with the highest ratio of number of paradigms to lemmas

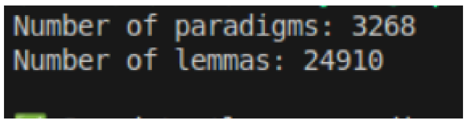
     ```
     Number of paradigms: 970
     Number of lemmas: 2192
     ```

  2. Hindi's morphology was more fusional with a lesser number of features expressing morphology as compared to Tamil, but still in good number. Thus, the ratio of paradigms to lemmas was a little lower as it should be.

     ```
     Number of paradigms: 3468
     Number of lemmas: 16979
     ```

  3. English, being morphologically poor, had the lowest paradigms to lemmas ratio.

     ```
     Number of paradigms: 3268
     Number of lemmas: 24910
     ```

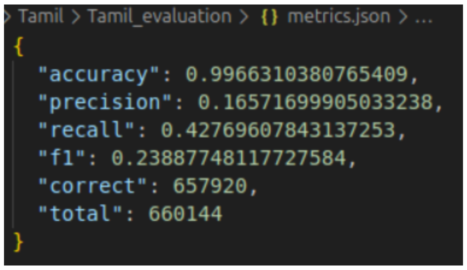
- **Tamil and Hindi require careful handling of phonological rules:** Morphophonemic alternations, especially in Tamil (e.g., vowel shortening, consonant doubling), introduced discrepancies between surface forms and lemmas. These could not be handled purely through suffix stripping. Hindi also showed issues with sandhi and stem alternations.

- **Shared lemmas with multiple POS tags:** Certain forms, especially in English (e.g., *run*), were ambiguous across parts of speech. Without disambiguation, these led to redundant or incorrect paradigm groupings.

4.2 Technical Challenges

- **Zero-width characters caused invisible errors:** Unicode characters such as Zero Width Joiner (U+200D) and Zero Width Non-Joiner (U+200C) caused parsing mismatches and erroneous token splits, particularly in Tamil and Hindi.
- **Coverage vs. correctness trade-off in FSTs:** The finite-state transducers generated many surface forms, but not all were correct. This led to high recall but low precision, especially in Tamil, where overgeneration occurred due to broad suffix rules.
- **CoNLL-U annotations were not always consistent:** Morphological feature tagging was sometimes inconsistent within or across UD datasets (e.g., `Gender=Masc` vs. `Gender=Masculine`), affecting paradigm extraction and rule matching.
- **Paradigm induction benefited from longest common prefix:** A simple longest-prefix strategy for identifying lemma-to-form differences worked surprisingly well for regular inflectional patterns, though it failed for irregular forms or stem-internal changes.

4.3 Evaluation and Performance Challenges



```
> Tamil > Tamil_evaluation > {} metrics.json > ...
{
  "accuracy": 0.9966310380765409,
  "precision": 0.16571699905033238,
  "recall": 0.42769607843137253,
  "f1": 0.23887748117727584,
  "correct": 657920,
  "total": 660144
}
```

Figure 1: Tamil Scores

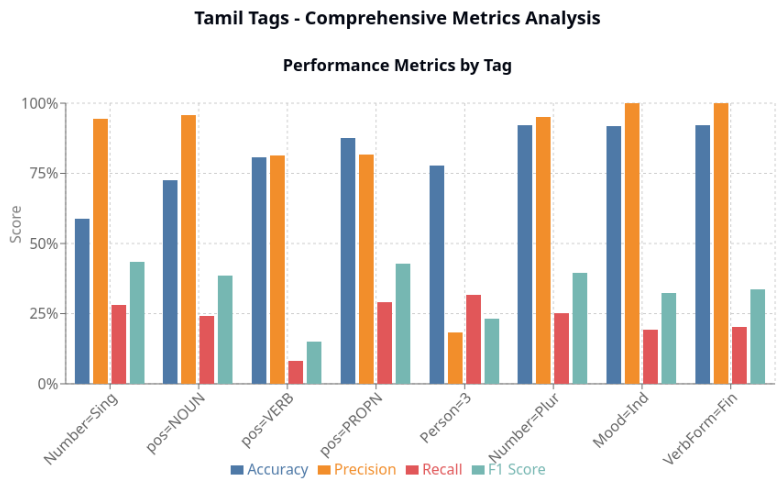
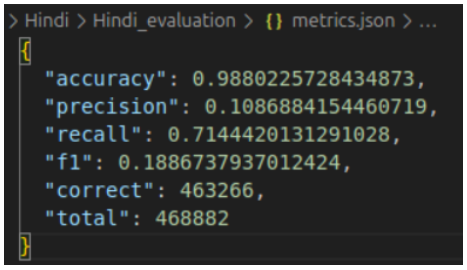


Figure 2: Top Tamil tags (ordered by frequency) with Accuracy, Precision, Recall, F1 and Frequency

Key Observations:

- Singular, 3rd person words are very frequent in text, followed by plural. POS tags, being more generalized than specific word markers like case, gender, etc., also show up very often in the dataset.
- Tamil tags generally show very high precision but lower recall. This hints at different tagging practices (e.g., NOUN vs pos=NOUN) found across datasets and the rich morphology of Tamil, which leads to undertagging.
- Accuracy generally increases as frequency decreases. The F1 scores are generally low due to poor recall, despite high precision.



```
> Hindi > Hindi_evaluation > {} metrics.json > ...
{
    "accuracy": 0.9880225728434873,
    "precision": 0.1086884154460719,
    "recall": 0.7144420131291028,
    "f1": 0.1886737937012424,
    "correct": 463266,
    "total": 468882
}
```

Figure 3: Hindi Scores

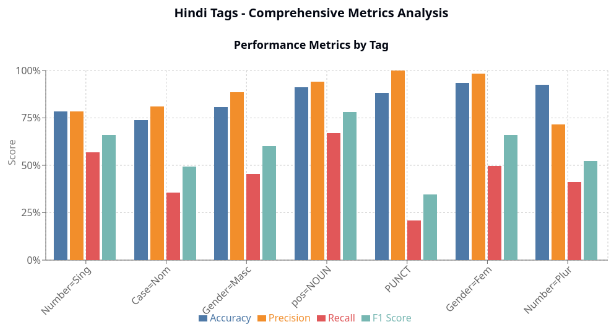
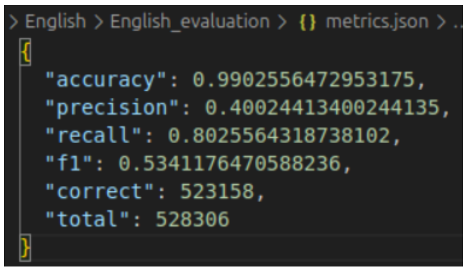


Figure 4: Top Hindi tags (ordered by frequency) with Accuracy, Precision, Recall, F1 and Frequency

Key Observations:

- Number=Sing (similar to the Tamil and English data), Case=Nom, and Gender=Masc are the most frequent tags in the Hindi test set (333, 326, and 290 occurrences respectively).
- PUNCT has 100 precision but relatively low recall (20.90), indicating it never gives false positives but misses many true positives.
- Accuracy generally increases as frequency decreases. Gender=Fem has the highest accuracy despite being on the lower side of frequency distribution.
- Data is relatively more balanced, showing proportionate precision, recall and F1 scores. Tags have moderate recall scores.
- pos=NOUN has the highest F1 score with strong balance between precision and recall.



```
> English > English_evaluation > {} metrics.json > ...
{
  "accuracy": 0.9902556472953175,
  "precision": 0.40024413400244135,
  "recall": 0.8025564318738102,
  "f1": 0.5341176470588236,
  "correct": 523158,
  "total": 528306
}
```

Figure 5: English Scores

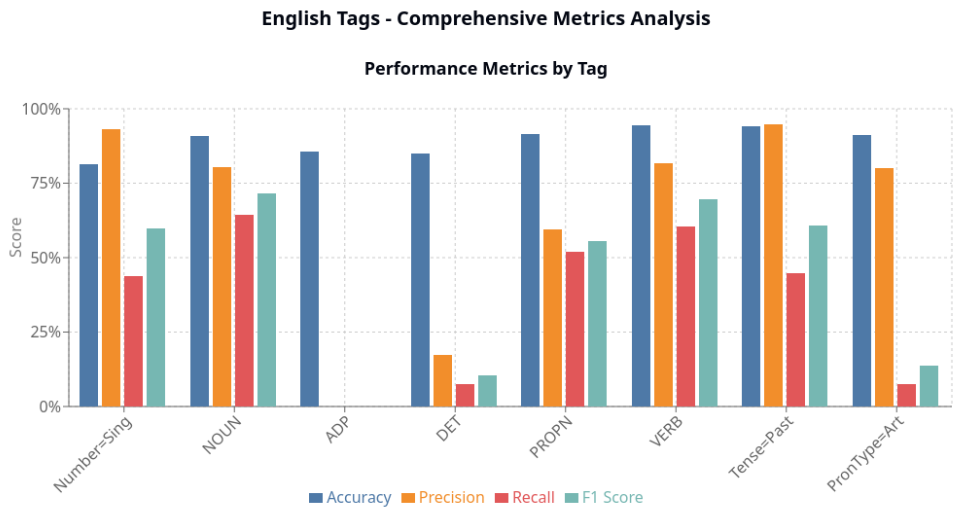


Figure 6: Top English tags (ordered by frequency) with Accuracy, Precision, Recall, F1 and Frequency

Key Observations:

- There's significant variation in precision scores, with ADP having close to 0 precision while Tense=Past has the highest at 94.64. ADP is often used inconsistently across corpora What's ADP in one dataset may be SCONJ, PART, or even ADV in another, leading to many mismatches.
- All tags have good and almost equal accuracies, showing a positive of English having more resources.
- Several tags have very low recall scores, indicating many false negatives.

4.3.1 General Performance Challenges

Accuracy was misleading: High accuracy in our evaluation (e.g., ¿99%) was largely due to true negatives—tokens like punctuation, numerals, or non-lexical items. Precision and F1 score were more realistic indicators of system performance on morphologically rich words.

Tamil had the most coverage challenges: Due to its richer morphotactics and the relatively smaller size of its annotated corpus, the Tamil analyzer showed more undergeneration, especially for honorific or compound forms.

Generation failures were often due to missing lemmas or incorrect paradigms: When testing generation via `lt-proc -g`, many inputs failed to yield output unless the lemma and its exact grammatical tags were already present and correctly linked in the `.dix` file.

Cross-lingual design generalization worked: Despite typological differences, our shared pipeline (from lexicon extraction to `.dix` generation) was adaptable to all three languages, demonstrating the power of a language-agnostic design informed by Universal Dependencies annotations.

5 Observations and Analysis

5.1 Introduction

This section provides a detailed analysis of the morphological analysers generated for three typologically diverse languages: Tamil, Hindi, and English. The analysers were constructed using data extracted from annotated Universal Dependencies corpora in CoNLL-U format, processed into lexicons and paradigms, and ultimately compiled into finite-state transducers (FSTs) using Apertium's Lttoolbox.

5.2 Evaluation Methodology

The evaluation of the morphological analysers was carried out by examining the number of lemmas and paradigms extracted from the training corpus and the coverage and accuracy of morphological generation and analysis. The main components evaluated include:

- The number of lemmas and paradigms extracted.
- The quality of the generated `.dix` dictionary.
- The successful compilation into binary FSTs using `lt-comp`.
- The coverage and correctness of outputs from `lt-proc`.

  All three FST-based analyzers perform reasonably well in identifying the correct lemma and morphological tags for frequently occurring and morphologically simple words. However, the performance deteriorates when handling morphologically complex, ambiguous, or unseen words. Notably:

  - **Tamil**: Performs well on agglutinative suffix chains but fails to capture all morphophonemic alternations and noun-adjective derivations.
  - **Hindi**: Handles basic inflectional morphology but overgenerates due to lack of disambiguation between homographs (e.g., *ΓΓ*).
  - **English**: Performs better due to the relatively simpler inflectional system but suffers from ambiguity in irregular verb forms.

  It is also important to note that these analyzers operate on a word-by-word basis and do not incorporate contextual information. As a result, they cannot disambiguate between homographs or infer syntactic roles based on surrounding words.

  Despite a very high accuracy for all the 3 morph analyzer-generators, the analyzer reports a low precision and F1 score. This discrepancy is due to the inherent differences in how each metric is computed:

  - **Accuracy** measures all correct decisions, including trivially correct rejections of punctuation or foreign words.

  - **Precision** penalizes overgeneration—if a word has multiple analyses and only one is correct, the rest count as false positives.
  - **Recall** reflects how many correct analyses were recovered from the gold set.
  - **F1 Score** balances both precision and recall.

In large corpora, many tokens do not require morphological analysis (punctuation, numerals, foreign words), contributing to high true negatives and thereby inflating the accuracy.

5.3 General Observations

Tamil Tamil, being an agglutinative Dravidian language, exhibits extensive inflectional morphology, especially for verbs. Despite having fewer irregular forms compared to Indo-Aryan languages, Tamil presents challenges due to:

- **Multiple Spellings:**Especially in Tamil and Hindi, words often appear in alternate orthographies (e.g., vowel sandhi variations or optional diacritics). These variants are not always captured in the dictionary, leading to missed analyses.
- **Sandhi and phonological alternations:** Tamil, in particular, exhibits complex morphophonemic changes that are not explicitly represented in the lemma, such as vowel shortening, consonant doubling,vowel harmony and insertion of glides at morpheme boundaries. These are not fully accounted for in the current FST rules.
- **4. Encoding Issues:**
- **Encoding and script variations:** Unicode normalization (e.g., NFC vs. NFD forms), inconsistent use of ZWJ/ZWNJ characters,combining characters,visually identical graphemes and typos in the corpus can result in mismatches between dictionary entries and tokenized input.
- **Multiple orthographic conventions:** Variation in how compound verbs or participles are written caused redundancy and coverage issues.
- **Data sparsity:** The available annotated dataset did not fully represent the breadth of Tamil's inflectional paradigms, leading to undergeneration.

Hindi Hindi, as an Indo-Aryan language with a relatively rich system of inflection, especially in verbs and nouns, exhibited the following patterns:

- **High morphological variability:** Multiple spellings, sandhi effects, and postpositions complicate the extraction of canonical paradigms.
- **Non-standardized annotation:** The CoNLL-U annotations sometimes inconsistently encoded features like gender and number, leading to duplicated or contradicting paradigm entries.

- **Orthographic normalization:** ZWJ characters and script ambiguities had to be removed explicitly to maintain consistency.
- **Incomplete lexical coverage:** While the analyser could capture several regular inflection patterns, forms from derivational or compound constructions were frequently missed.

English English has a relatively simpler morphology compared to Tamil and Hindi, with a limited set of inflectional suffixes.

- **Regular vs. irregular forms:** While regular verb conjugations were well-captured, irregular verbs (e.g., "go" $\rightarrow$ "went") presented a challenge due to lack of rules in the dataset.
- **Part-of-speech ambiguity:** Homographs (e.g., "book" as noun vs. verb) were occasionally misclassified due to the reliance on POS tags without disambiguation.
- **Sparse morphological data:** The annotated datasets often excluded derivational morphology, thus limiting the analyser's scope.

5.4 Quantitative Summary

Language	**Lemmas**	**Paradigms**	**Ratio**
Tamil	2192	970	2.25
Hindi	16979	3468	4.8
English	24910	3268	7.6

Overview of extracted morphological data per language

5.5 Detailed Explanation of Metric Discrepancy

Finite State Transducer (FST) based morphological analyzers, such as those built using Apertium's `lttoolbox`, are deterministic and dictionary-driven: given a word, they either provide an analysis (e.g., lemma and morphological tags) or fail to do so. In evaluating such systems, especially against a gold-standard dataset like a tagged corpus (e.g., in CoNLL-U format), we rely on metrics like accuracy, precision, recall, and F1 score to determine how well the analyzer performs.

High Accuracy Explained: In large corpora such as UD (Universal Dependencies) datasets, a vast number of tokens are not morphologically rich words. These include:

- Punctuation (e.g., .", ,"),
- Numbers (e.g., 2024"),
- Symbols (e.g.,

- Foreign or unrecognized words,
- Function words (e.g., and", "or").

Such tokens are often correctly ignored or trivially marked as unknown, which contributes a large number of true negatives. This significantly boosts the accuracy score.

Low Precision: Precision suffers when the analyzer returns multiple (often incorrect or irrelevant) analyses for a given word. This overgeneration is typical in rule-based FST systems that do not include a disambiguation step. For example, a Hindi word like *ΓΓ* might be analyzed both as a noun (tax") and a verb (do"), leading to false positives. Since the system operates word-by-word and lacks contextual awareness, it cannot select the contextually appropriate analysis.

Moderate Recall: Despite overgeneration, the analyzer still captures a substantial portion of the correct analyses. This is due to good coverage of high-frequency lemmas and affixes.

Low F1 Score: The F1 score reflects the trade-off between precision and recall. Since precision is particularly low, F1 remains low as well.

5.6 Recommendations

- Incorporate disambiguation (e.g., using Constraint Grammar or statistical models) to reduce overgeneration.
- Improve dictionary coverage by adding more lemmas and inflectional paradigms.
- Implement sandhi rules and morphophonemic alternation handling, especially for Tamil.
- Normalize input strings to a consistent Unicode form and clean the corpus for typos and non-standard spellings.

5.7 Challenges and Limitations

The morphological analyzers faced several common limitations across languages:

- **Multiple Spellings:** Especially prevalent in user-contributed datasets or noisy annotations, this caused a splitting of forms across paradigms. Tamil showed particularly high variance due to regional orthographic norms.
- **Encoding Issues:** ZWJ and ZWNJ characters, as well as invisible diacritics, caused mismatches during matching and sorting. Manual sanitization was required to ensure uniformity.

- **Data Sparsity:** Many inflectional forms for verbs and nouns were not represented in the CoNLL-U datasets, resulting in analysers that lacked comprehensive coverage. This led to undergeneration during `lt-proc` testing.
- **Sandhi and Morphophonology:** Particularly in Tamil and Hindi, phonological alternations were not addressed by the rule-based system, as it lacks phonological conditioning mechanisms unless manually encoded.
- **Annotation Inconsistency:** Variability in tagging conventions for features (e.g., "Gender=Masc" vs. Gender=Masculine") led to duplicate or inconsistent entries in the lexicon.

5.8 Conclusions

The FST-based morphological analyzer offered fast and rule-governed analysis but require careful tuning of rules and lexicon to handle real-world variability. Evaluating such systems solely by accuracy is misleading—precision, recall, and F1 provide a more realistic measure of performance. The Tamil, Hindi, and English systems demonstrate promising initial results but highlight the need for expanded lexicons, morphological coverage, and disambiguation techniques to improve practical usability. Going forward, expanding the dictionary to include derivational morphology, better handling of named entities and foreign words, and building statistical disambiguators can improve utility in NLP pipelines. Furthermore, adapting analyzers to be robust against noisy or informal text (e.g., social media) and enabling language-specific post-processing (e.g., compounding in Germanic or Dravidian languages) are promising directions for future work.

Overall, the morphological analyser generator proved effective for automatically building draft finite-state dictionaries using available annotated corpora. While the approach worked reasonably well for English and moderately well for Hindi, Tamil presented the greatest challenges due to script complexity, agglutination, and morphophonological variation. Key takeaways include:

- The modular design enables adaptation to different languages with relatively minimal changes.
- The reliance on CoNLL-U formatted data ensures wide applicability but also introduces risks due to annotation inconsistency.
- Enhancements such as rule-based morphophonemic modules, expanded training data, and integration with language-specific normalizers would improve both coverage and accuracy.

5.9 Future Work

Moving forward, improvements to the analyser could include:

- Developing custom normalization and tokenization pipelines per language.
- Incorporating external dictionaries or manually curated paradigms to enhance coverage.
- Applying semi-supervised or neural methods to learn sandhi and allomorphic variations.
- Extending the system to handle derivational morphology and compounding.

These refinements would help transition the analysers from basic prototype tools into robust components suitable for real-world NLP applications in resource-scarce languages.

Files Generated:

- `lexicon.csv` — The extracted lexicon with POS, lemmas, and inflected forms.
- `lemma_paradigms.json` — Mapping of lemma clusters to morphological transformation rules.
- `morph_analyser.dix` — Apertium-format dictionary used to compile the analyser.

Compiled with: `lt-comp`, `lt-proc` from Apertium.
Supported languages: Hindi, Tamil, English.

6 Conclusion

The Project started with the objective of creating a Morphological Analyzer for Indian languages, namely Hindi, English and Telugu (but is easily adabptable to other languages). Multiple challenges were encountered in the beginning, but through a series of systematic steps, improvements, analysis and reading, our objectives have been achieved and we have arrived at multiple conclusions.

This project has been implemented using the lttoolbox module from the Apertium platform. Morphological analysis forms the backbone of numerous Natural Language Processing tasks, and creating a finite-state transducer-based solution serves as a foundational step towards building more complex language tools, especially for low-resource languages. Over the duration of this work, a functional morphological analyzer was successfully developed, lexicons and paradigms were created, and the system was tested in each of the three languages against over 1000+ manually annotated words and tokens.

6.1 Performance of Finite-State Transducers in Morphological Analysis

While statistical and neural methods dominate recent NLP trends, rule-based systems like Apertium retain critical value. These systems allow for precise control, transparency, and adaptability that are often difficult to achieve with black-box models.

One of the most significant advantages of using FSTs is their efficiency, both in terms of memory usage and runtime performance. Once compiled into a binary format using tools like lt-comp, an FST can process thousands of word forms per second with deterministic accuracy. Unlike statistical models, which rely on probabilities and training data, FSTs perform lossless, rule-governed transformations, making them particularly well-suited for low-resource languages where annotated corpora may be sparse or nonexistent. Furthermore, the bidirectionality of FSTs allows the same system to be used for both analysis and generation, enhancing reusability and modularity. Despite their strengths, FSTs require meticulous design, including the careful crafting of paradigms and lexical entries, and may struggle with irregularities or exceptions not explicitly accounted for.

6.2 Language-based Observations

The analyzer demonstrated varying levels of performance across the three languages: English, Hindi and Tamil as seen above in the observations.

6.2.1 English

In the case of English, the analyzer performed exceptionally well. This is largely due to English's relatively simple morphological structure, limited inflectional variations, and the abundance of existing lexical resources and paradigms that could be directly adapted or referred to during rule design. The majority of English words follow predictable inflectional patterns (e.g., pluralization with -s, past tense with -ed), making finite-state rule encoding straightforward and highly effective.

6.2.2 Hindi and Tamil

For Hindi and Tamil, the performance was quite accurate, but demonstrated lower precision, recall and F1 score as compared to English. Both languages, especially Tamil, are morphologically rich, involving complex systems of inflection, derivation, and compounding. In Hindi, challenges included handling sandhi rules, gender agreement, and case-marking in nouns and verbs. Tamil, being an agglutinative language, posed difficulties in suffix stacking, morphophonemic changes, and verb conjugation across multiple tenses and moods. Additionally, while English benefited from wide documentation and multiple digital lexical resources, building accurate paradigms and dictionaries for Hindi and Tamil required more effort. To summarize, here is a table depicting performance scores across the three languages. More detailed per-tag metrics and full results are available in the Results directory of the project.

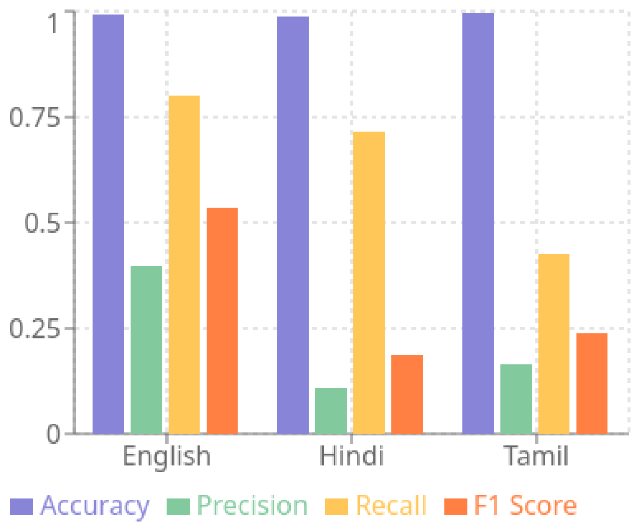


Figure 7: Accuracy, Precision, F1 and Recall comparison across languages

6.3 Potential Improvements

Error Type	Description
Overgeneration	Generates invalid or unlikely forms
Undergeneration	Fails to produce valid forms
Incorrect Tag	Wrong part-of-speech or feature tag
Ambiguity	Multiple outputs
Orthographic Error	Errors due to spelling variations
Segmentation Error	Incorrect splitting or joining of words
Data Sparsity	Training data lacks coverage
Affix Handling Error	Suffix or prefix improperly analyzed
Compound Error	Errors in compound word handling
Inflection Error	Wrong tense/gender/number output
Derivational Error	Incorrect derivation or root generation
Encoding Error	Script/font issues causing misreading
Unknown Word Error	Unseen word not handled correctly
Contextual Mismatch	Correct word, wrong in sentence context

Table 1: Summary of error types encountered in the project

Several aspects could be improved for greater accuracy, precision and robustness. Firstly, expanding the dictionary with more irregular forms and domain-specific vocabulary would enhance precision, particularly in morphologically rich languages like Hindi and Tamil. Refining existing paradigms to better capture edge cases (such as compound constructions, borrowed terms, and regional variants) could reduce both undergeneration and overgeneration. Furthermore, explicitly incorporating morphophonemic rules would help handle Tamil's complex suffix stacking and phonological transformations more accurately.

Another key improvement would be introducing sentence level context to disambiguate analyses, especially in languages with high homography or free word order. Currently, the system operates purely at the word level. Integrating a context-aware layer could help resolve ambiguities that static FSTs alone cannot handle. This could be achieved through the integration of rule-based FSTs with statistical or neural models, such as Hidden Markov Models (HMMs), Conditional Random Fields (CRFs), or transformer based taggers, which could provide probabilistic ranking or disambiguation of multiple morphological outputs.

6.4 Looking Forward

This project lays the groundwork for more accessible and scalable linguistic tools, especially for under resourced Indian languages.

Looking ahead, this analyzer can serve as a foundational layer in larger language technology pipelines. With continued expansion, both in terms of linguistic coverage and technical sophistication, it can be adapted into cross-lingual platforms that promote language preservation and accessibility. The potential integration of this system with spell-checkers, grammar correction tools, and educational tools opens the door to impactful real-world applications, especially in regions where digital linguistic support remains scarce.

6.5 Conclusion

In conclusion, the accuracy of the Morphological Analyzer varies based on the language, the model used, and the amount and type of data used for training. The project was a great learning experience and a significant step for us toward building and understanding interpretable and efficient morphological analysis tools.