UNIT I: Introduction to NLP

Introduction to NLP: Origins and Challenges, Language and Grammar in NLP, Regular Expressions and Finite-State Automata, Tokenization: Text Segmentation and Sentence Splitting, Morphological Parsing: Stemming and Lemmatization, Spelling Error Detection and Correction, Minimum Edit Distance and Applications, Statistical Language Models: Unigram, Bigram, and Trigram Models, Processing Indian Languages in NLP.

INTRODUCTION TO NLP

Natural Language Processing (NLP) is a subfield of **Artificial Intelligence (AI)** and **Linguistics** that enables machines to understand, interpret, and generate human languages. It bridges the gap between **human communication** and **computer understanding**.

Example:

- Voice assistants like Alexa, Siri, and Google Assistant.
- Machine translation systems (Google Translate).
- Chatbots and question-answering systems.

2. Importance of NLP

- Enables **human-computer interaction** in natural languages.
- Automates text processing tasks (translation, summarization, sentiment analysis).
- Useful in healthcare, finance, education, law, and customer service.
- Drives **data-driven decision-making** by extracting knowledge from large text datasets.

3. Core Components of NLP

1. **Syntax (Structure of Language):**
    - Parsing, POS tagging, grammar analysis.
    - Example: "The cat sat on the mat" → identifies subject, verb, object.
2. **Semantics (Meaning of Language):**
    - Word sense disambiguation, semantic roles, logical forms.
    - Example: “Bank” (riverbank vs. financial bank).
3. **Pragmatics (Context of Language):**

- Considers context, intent, and world knowledge.
- Example: "Can you open the window?" is a request, not a question about ability.

4. Common NLP Tasks

- **Tokenization:** Splitting text into words/sentences.
- **Stopword Removal:** Filtering common words like "is, the, a."
- **Stemming & Lemmatization:** Reducing words to base form (running → run).
- **Named Entity Recognition (NER):** Identifying entities (e.g., names, places).
- **Sentiment Analysis:** Detecting opinions and emotions.
- **Machine Translation:** Translating text from one language to another.
- **Speech Processing:** Converting speech to text and vice versa.

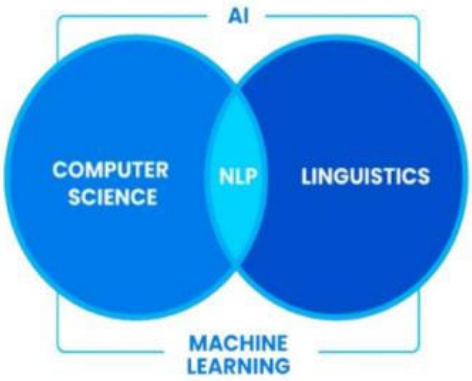
5. Challenges in NLP

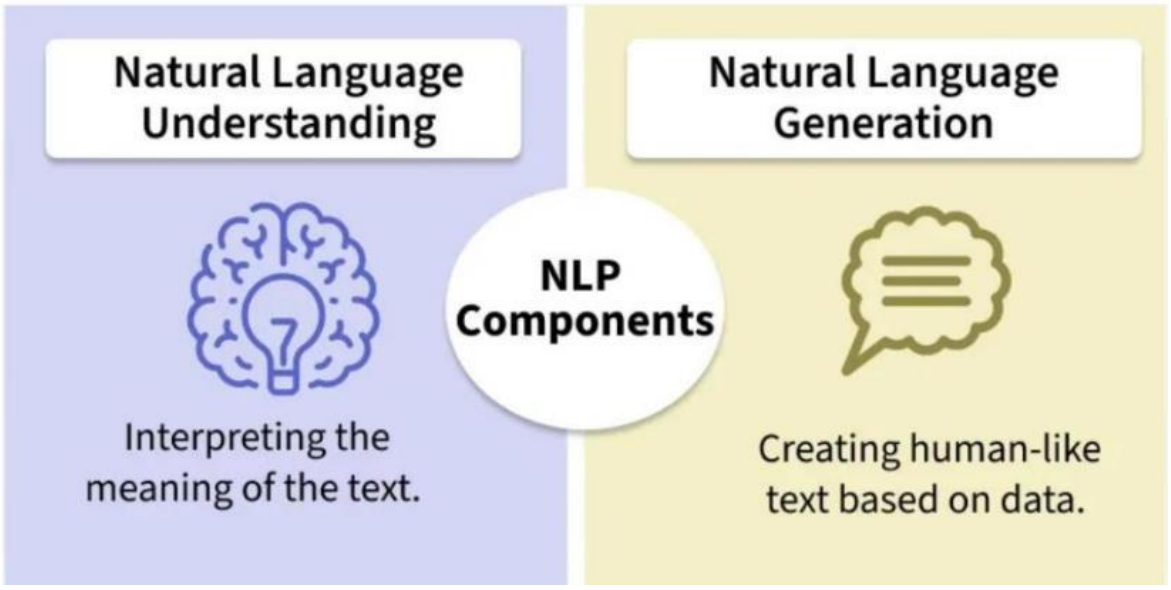
- **Ambiguity:** Words and sentences may have multiple meanings.
- **Resource Scarcity:** Low-resource languages lack sufficient corpora.
- **Context Understanding:** Computers struggle with sarcasm, irony, and metaphors.
- **Multilinguality:** Handling multiple languages and dialects.

6. Applications of NLP

- **Search Engines** (Google, Bing).
- **Virtual Assistants** (Siri, Alexa).
- **Healthcare** (clinical text analysis, medical chatbots).
- **Social Media Analytics** (sentiment detection, trend analysis).
- **Education** (automated essay grading, language learning apps).

UNIT I





ORIGINS AND CHALLENGES

NLP evolved from **rule-based symbolic systems** to **statistical models**, and now to **transformer-based architectures**. However, it continues to face challenges in **ambiguity, context, multilinguality, bias, and ethical concerns**.

1. Origins of NLP

- **1950s: Birth of NLP**
    - Alan Turing proposed the **Turing Test (1950)** to measure machine intelligence through language.
    - Early attempts: **Rule-based systems** and **Symbolic AI** for translation and text processing.

- **1960s–70s: Early Experiments**
    - **Machine Translation** projects (like the Georgetown–IBM experiment).
    - Development of **SHRDLU** (1968), which could follow natural language commands in a blocks world.
- **1980s: Statistical NLP**
    - Shift from symbolic approaches to **statistical methods** using corpora.
    - Introduction of **Hidden Markov Models (HMMs)** in speech recognition.
- **1990s–2000s: Data-Driven NLP**
    - Rise of **Machine Learning models** (Naive Bayes, SVMs, Decision Trees).
    - Availability of large annotated corpora (Penn Treebank, Brown Corpus).
- **2010s: Deep Learning Era**
    - **Word Embeddings** (Word2Vec, GloVe).
    - **Recurrent Neural Networks (RNNs), LSTMs, and GRUs** for sequence modeling.
- **2020s: Transformer-based NLP**
    - Models like **BERT, GPT, T5, and LLaMA** revolutionized NLP.
    - Enabled **state-of-the-art performance** in translation, question answering, and summarization.

2. Challenges in NLP

- **Ambiguity**
    - Words/sentences can have multiple meanings (e.g., *"bank" → river bank vs. financial bank*).
- **Context Understanding**
    - Handling sarcasm, irony, pragmatics, and discourse-level understanding.
- **Multilinguality**
    - Different grammar, word order, morphology across languages.
- **Low-Resource Languages**

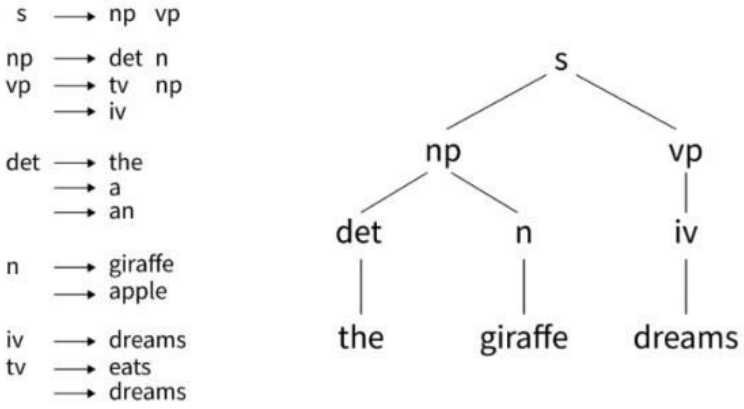
- Many Indian and African languages lack annotated corpora.

- **World Knowledge Integration**
  - Machines need background knowledge to interpret meaning correctly.
- **Bias and Fairness**
  - Models can amplify social, gender, or cultural biases from training data.
- **Scalability**
  - Large transformer models require **huge computational resources**.
- **Ethical Issues**
  - Misinformation, deepfakes, surveillance, and misuse of NLP technology.

LANGUAGE AND GRAMMAR IN NLP

Language and grammar are the **foundation of NLP**. Grammar formalisms like **CFG, PSG, and Dependency Grammar** help computationally model language, while challenges like **ambiguity and idioms** make the task complex.

UNIT I



- Defines a set of production rules that describe all possible sentences in a language.
- Useful in parsing sentences and building syntax trees.
- **Example rule:**

$$S \rightarrow NP\ VP$$

$$NP \rightarrow Det\ Noun$$

$$VP \rightarrow Verb\ NP$$

1. Role of Language in NLP

- Language is the **medium of communication** between humans and machines.
- NLP aims to model both the **structure (syntax)** and **meaning (semantics)** of language for computational use.
- A system must handle **words, phrases, clauses, and sentences** while preserving meaning.

2. Grammar in NLP

Grammar defines the **rules and structure** of a language. In NLP, grammar is essential for:

- **Parsing**: Analyzing sentence structure.
- **Machine Translation**: Ensuring syntactic correctness.
- **Question Answering**: Understanding sentence intent.

3. Types of Grammar in NLP

(a) Morphology

- Study of **word formation** and structure.
- Example: *play → plays, playing, played*.

(b) Syntax

- Study of **sentence structure** (rules for arranging words).
- Example:
  - Correct: *The cat sat on the mat.*
  - Incorrect: *Cat mat sat the on.*

(c) Semantics

- Deals with **meaning of words and sentences**.
- Example: *John kicked the ball* vs. *The ball kicked John* (different meanings).

(d) Pragmatics

- Study of **contextual meaning** (depends on situation).
- Example: *"Can you pass the salt?"* → a request, not a question about ability.

(e) Discourse

- Concerned with **multiple sentences together**.
- Example: *She went to the market. She bought apples.* → pronoun *she* refers to same person.

(f) Phonology (Speech-based NLP)

- Study of **sound systems** in language.

4. Grammar Formalisms in NLP

1. **Phrase Structure Grammar (PSG)**
   - Represents sentences as hierarchical structures.
   - Example: Sentence (S) → Noun Phrase (NP) + Verb Phrase (VP).
2. **Dependency Grammar**
   - Represents grammatical relationships as **dependencies between words**.

- Example: In *"She eats apples"*, *eats* → head verb, *she* → subject, *apples* → object.

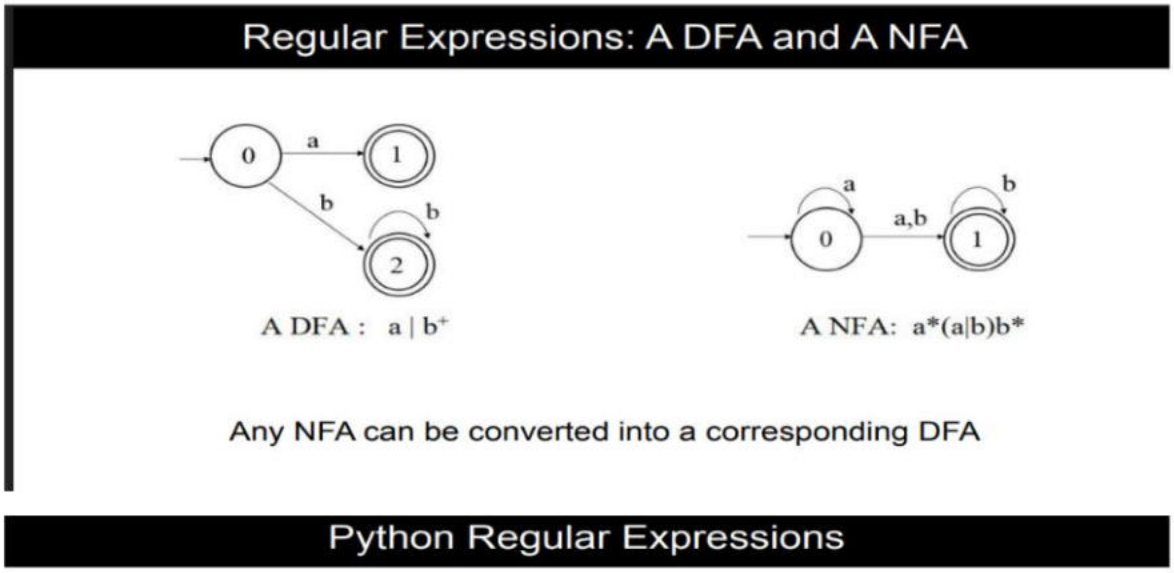
3. **Context-Free Grammar (CFG)**
   - Uses rules like **S → NP VP** to generate valid sentences.
4. **Transformational Grammar** (Chomsky)
   - Describes how deep structures can be transformed into surface forms.

5. Challenges in Using Grammar for NLP

- **Ambiguity:**
  - *"I saw the man with a telescope"* → who has the telescope?
- **Ellipsis:** Missing words in sentences.
- **Idioms:** Non-literal expressions (*"kick the bucket"* ≠ kick + bucket).
- **Cross-linguistic variation:** Different languages follow different grammar rules.

REGULAR EXPRESSIONS AND FINITE-STATE AUTOMATA

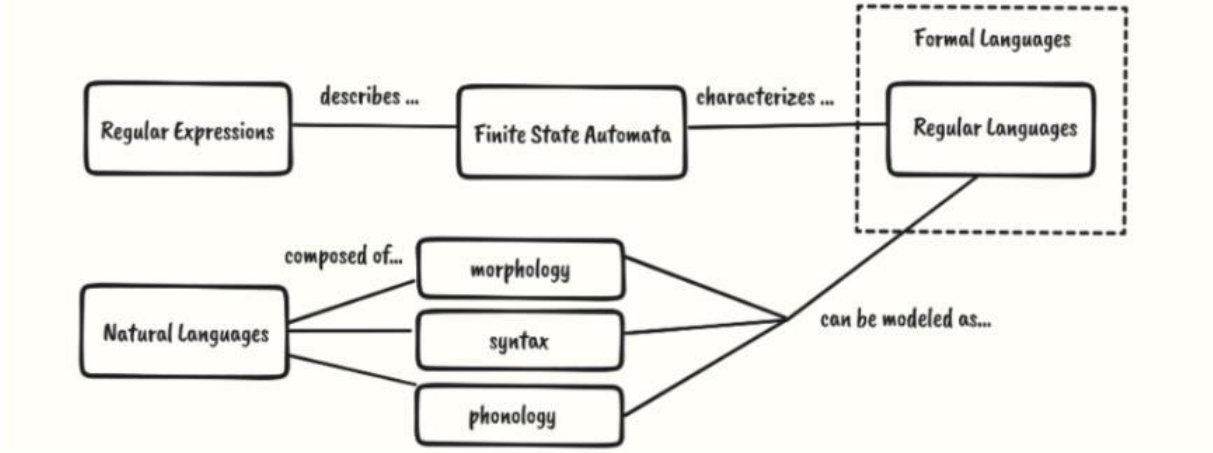
Regular Expressions provide a compact way of specifying patterns in text, while Finite-State Automata provide the computational framework for recognizing those patterns. Together, they form the foundation of many **NLP preprocessing tasks** such as tokenization, morphological analysis, and text search.



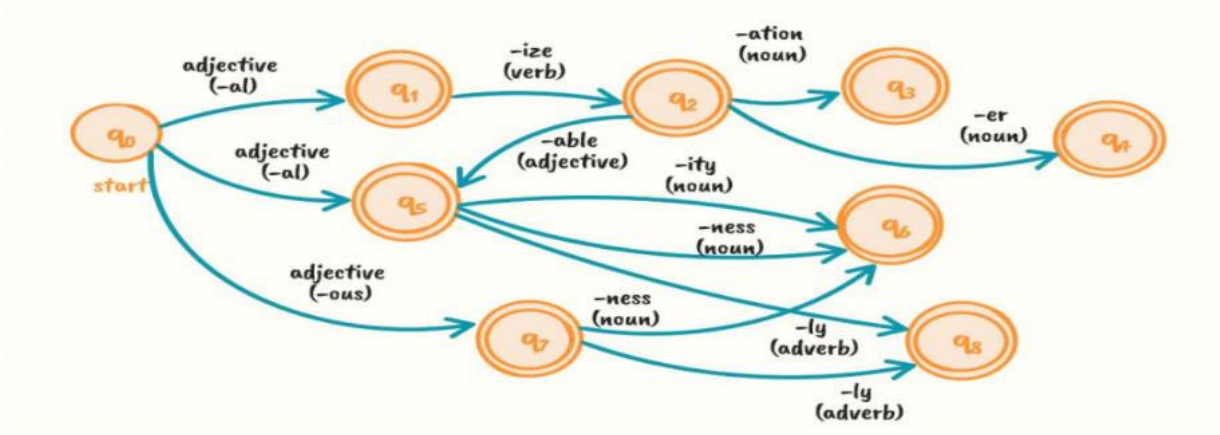
Finite State Automata (FSAs)

A finite-state automaton $M = (Q, \Sigma, q_0, F, \delta)$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- A finite alphabet Σ of input symbols (e.g. $\Sigma = \{a, b, c, \ldots\}$)
- A designated start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ:
  - The transition function for a deterministic (D)FSA: $Q \times \Sigma \rightarrow Q$
    $\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$
    If the current state is q and the current input is w, go to q'
  - The transition function for a nondeterministic (N)FSA: $Q \times \Sigma \rightarrow 2^Q$
    $\delta(q,w) = Q'$ for $q \in Q, Q' \subseteq Q, w \in \Sigma$
    If the current state is q and the current input is w, go to any $q' \in Q'$



UNIT I



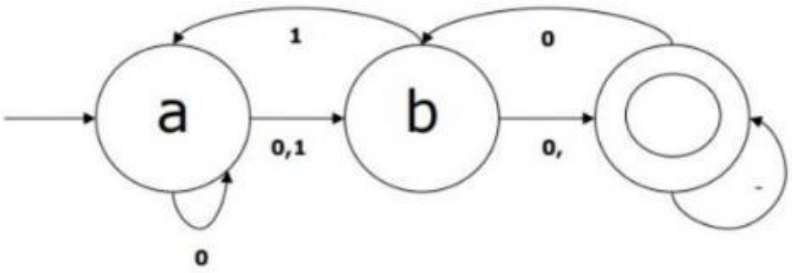
Example of NDFA

Suppose a NDFA be

- Q = {a, b, c},
- Σ = {0, 1},
- q0 = {a},
- F = {c},
- Transition function δ is shown in the table as follows –

Current State	Next State for Input 0	Next State for Input 1
A	a, b	B
B	C	a, c
C	b, c	C

The graphical representation of this NDFA would be as follows –



1. Regular Expressions (Regex)

- **Definition**: A regular expression is a sequence of symbols and characters that defines a search pattern, mainly for string matching and text processing.
- **Role in NLP**:
  - Tokenization (splitting sentences/words).
  - Pattern matching (finding dates, phone numbers, emails).
  - Text normalization (removing punctuation, replacing characters).
  - Morphological analysis (finding word stems, suffixes, prefixes).
- **Examples**:
  - \d+ → matches numbers (e.g., 123).
  - [A-Z][a-z]+ → matches proper nouns (e.g., India, John).

2. Finite-State Automata (FSA)

- **Definition**: A **Finite-State Automaton** is a mathematical model of computation consisting of **states** and **transitions**, used to recognize patterns described by **regular expressions**.
- **Components**:
  - **States**: Different conditions (start, accept, reject).
  - **Alphabet**: Set of input symbols (e.g., {a, b}).
  - **Transitions**: Rules that define how to move between states.
  - **Accepting States**: Indicate successful recognition of a string.
- **Types of FSA**:
  - **Deterministic Finite Automata (DFA)** – Only one possible transition per input.
  - **Non-deterministic Finite Automata (NFA)** – Multiple possible transitions.

3. Relationship between Regex and FSA

- Every **regular expression** can be represented as a **finite-state automaton**.
- Regex is a **declarative way** to describe a pattern, while FSA is a **procedural way** to recognize it.

- Example: Regex (ab)* can be represented by an FSA that loops between states when it sees a followed by b.

4. Applications in NLP

- **Tokenization**: Splitting words and sentences.
- **Morphological Parsing**: Handling word stems and inflections.
- **Lexical Analysis**: Identifying keywords in text.
- **Spell Checking**: Recognizing valid word patterns.
- **Named Entity Recognition (NER)**: Detecting names, places, dates.

5. Example

Regex: cat|dog

- Matches either "cat" or "dog".
- FSA equivalent:
  - Start state → "c-a-t" → Accept state
  - OR → "d-o-g" → Accept state

TOKENIZATION: TEXT SEGMENTATION AND SENTENCE SPLITTING

Tokenization is a **fundamental preprocessing step** in NLP that involves text segmentation, sentence splitting, and word tokenization. Advanced methods like **BPE and WordPiece** are crucial in modern deep learning–based NLP models.

1. Introduction

- **Tokenization** is the process of breaking raw text into **smaller units** called *tokens*.
- Tokens can be **words, subwords, characters, or sentences**, depending on the application.
- It is the **first step** in most NLP pipelines.

2. Levels of Tokenization

1. **Text Segmentation**
   - Dividing text into meaningful units.
   - Needed especially in languages **without spaces** (e.g., Chinese, Japanese).

- Example:
  - Input: "我喜欢学习NLP"
  - Segmented: ["我", "喜欢", "学习", "NLP"]

2. **Sentence Splitting (Sentence Boundary Detection)**
   - Splitting paragraphs into sentences.
   - Challenges: Abbreviations, punctuation ambiguities.
   - Example:
     - Input: "Dr. Smith went to Washington. He arrived at 5 p.m."
     - Sentences:
       - "Dr. Smith went to Washington."
       - "He arrived at 5 p.m."
3. **Word Tokenization**
   - Splitting sentences into words.
   - Example:
     - Input: "NLP is fun!"
     - Tokens: ["NLP", "is", "fun", "!"]

3. Techniques for Tokenization

- **Rule-based Approaches**
  - Use regular expressions and punctuation rules.
  - Works well for English and space-delimited languages.
- **Dictionary-based Approaches**
  - Match words from a lexicon (common for Chinese, Japanese).
- **Statistical Models**
  - Use probabilities of word boundaries (e.g., Hidden Markov Models).
- **Neural Network Approaches**

- Byte-Pair Encoding (BPE), WordPiece, SentencePiece.
- Common in **Transformer models** (BERT, GPT).

4. Challenges in Tokenization

- **Ambiguity**: "I saw the man with the telescope."
- **Multi-word expressions**: "New York", "kick the bucket".
- **Contractions**: "don't" → ["do", "n't"].
- **Hyphenation**: "state-of-the-art".
- **Code-mixed text**: "मैं school जा रहा हूँ" (Hindi-English mix).

5. Applications in NLP

- Text preprocessing for **search engines, translation, speech recognition**.
- Basis for **morphological analysis**.
- Feeding input to **machine learning models**.
- Improves **semantic understanding** by creating meaningful units.

MORPHOLOGICAL PARSING: STEMMING AND LEMMATIZATION

- **Stemming** is fast and heuristic-based but may produce non-words.
- **Lemmatization** is slower but linguistically accurate, mapping words to their dictionary forms.
- Both are essential **morphological parsing techniques** used in text preprocessing and NLP tasks.

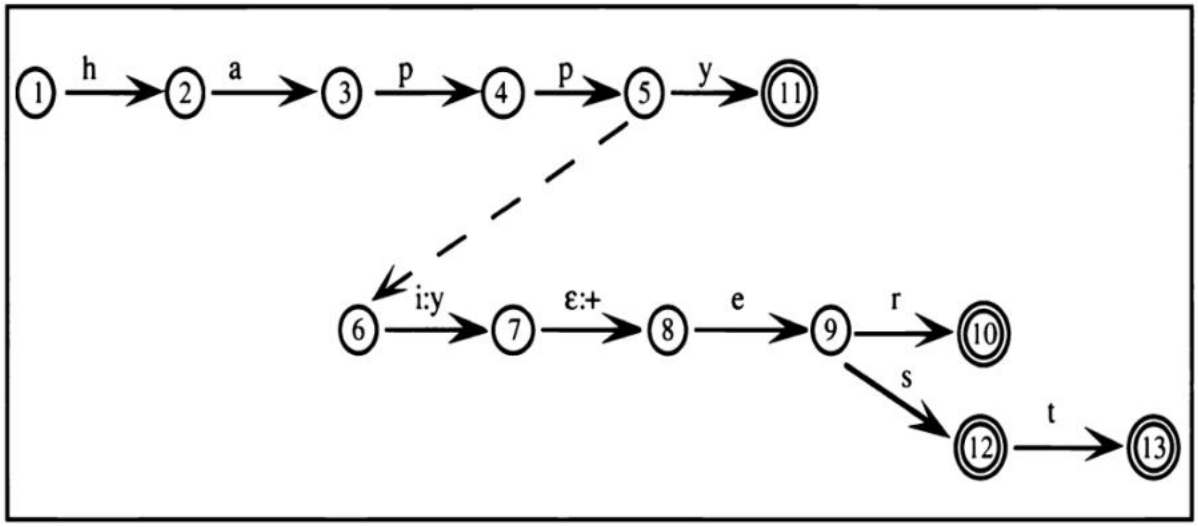


Figure 3.27 A simple FST for the forms of *happy*

network. The FST in Figure 3.28 accepts the following words, which all start with *t*: *tie* (state 4), *ties* (10), *trap* (7), *traps* (10), *try* (11), *tries* (15), *to* (16), *torch* (19), *torches* (15), *toss* (21), and *tosses* (15). In addition, it outputs the appropriate sequence of morphemes.

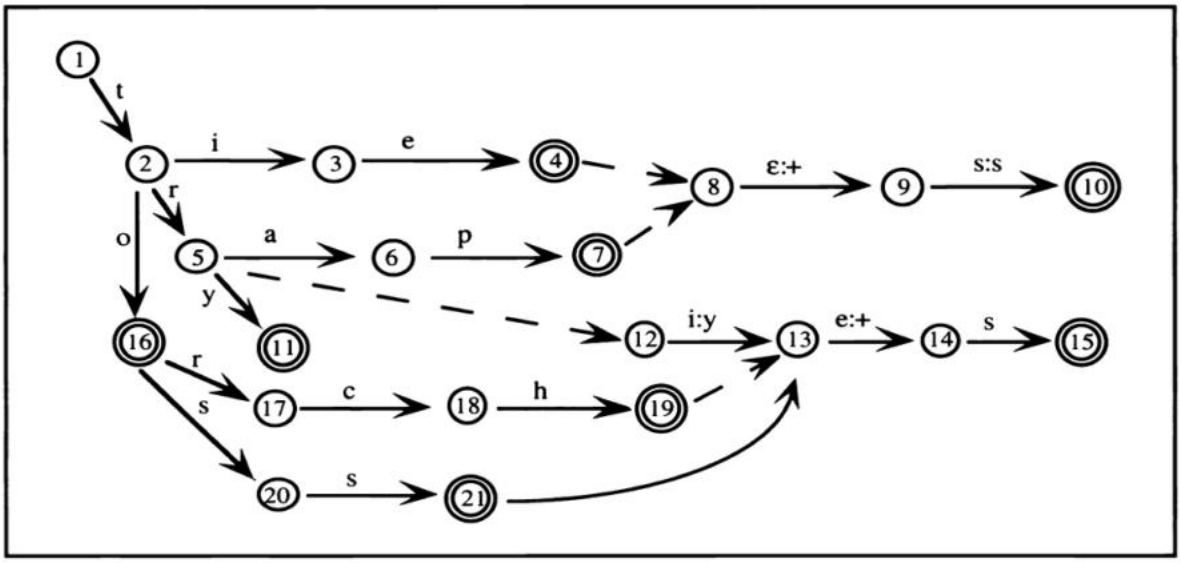


Figure 3.28 A fragment of an FST defining some nouns (singular and plural)

1. Introduction

- **Morphology** is the study of the internal structure of words.

- In NLP, **morphological parsing** analyzes words into their **root (stem/lemma)** and **affixes** (prefix, suffix, inflection).
- It is essential for **normalizing words** to their base form so that different variations of a word can be treated as equivalent.

2. Stemming

- **Definition**: Stemming reduces a word to its **root form** by chopping off prefixes or suffixes, often using **rule-based heuristics**.
- **Characteristics**:
  - Fast, simple, and language-specific.
  - Output may not be a valid dictionary word.
- **Examples**:
  - playing → play
  - studies → studi
  - happiness → happi
- **Algorithms**:
  - **Porter Stemmer** (most widely used).
  - Snowball Stemmer.
  - Lancaster Stemmer.

3. Lemmatization

- **Definition**: Lemmatization reduces a word to its **lemma (dictionary form)** using vocabulary and **morphological analysis**.
- **Characteristics**:
  - Produces valid words.
  - Requires knowledge of **Part-of-Speech (POS)**.
- **Examples**:
  - playing → play
  - studies → study

- better → good (uses lexical knowledge).

- **Techniques**:
  - Rule-based morphological analysis.
  - Dictionary lookups (WordNet Lemmatizer).

4. Stemming vs. Lemmatization

Feature	Stemming	Lemmatization
Output	May not be valid word	Always valid word
Method	Rule-based chopping	Dictionary + POS analysis
Speed	Faster	Slower (needs resources)
Accuracy	Lower (over-stemming, under-stemming)	Higher (linguistically informed)

---

5. Applications in NLP

- **Information Retrieval**: Improves search accuracy (e.g., "studying" → "study").
- **Text Mining**: Normalizes words for frequency analysis.
- **Machine Translation**: Handles inflected word forms.
- **Sentiment Analysis**: Groups word variants for better sentiment detection.
- **Question Answering**: Matches user queries to documents despite word variation.

SPELLING ERROR DETECTION AND CORRECTION

- Spelling error detection identifies **non-word and real-word errors**.
- Correction uses methods like **edit distance, phonetic matching, language models, and neural networks**.
- It improves the **accuracy and usability** of NLP systems across multiple domains.

1. Introduction

- **Spelling error detection** is the process of identifying misspelled words in text.
- **Correction** involves suggesting the most likely intended word.

- It is crucial in NLP for improving text quality in **search engines, chatbots, machine translation, speech recognition, and document processing**.

2. Types of Spelling Errors

1. **Non-word Errors**
    - Occur when a word is not in the dictionary.
    - Example: "recieve" → "receive".
2. **Real-word Errors**
    - Grammatically valid but contextually incorrect.
    - Example: "He went too the park." → "He went to the park.".
3. **Typographical Errors**
    - Caused by mistyping (extra, missing, or wrong characters).
    - Example: "splling" → "spelling".
4. **Phonetic Errors**
    - Words spelled as they sound.
    - Example: "nite" → "night".

3. Techniques for Error Detection

- **Dictionary Lookup**
    - Compare each token with a vocabulary list.
    - Works well for non-word errors.
- **Statistical Models**
    - N-grams or probabilistic models to detect unusual word sequences.
- **Context-based Models**
    - Use surrounding words to detect real-word errors.
    - Example: "There book is on the table." → "Their book...".

4. Error Correction Methods

1. **Edit Distance (Levenshtein Distance)**
    - Measures similarity between words by counting insertions, deletions, substitutions.
    - "recieve" → "receive" (1 substitution + 1 transposition).
2. **Phonetic Algorithms**
    - Soundex, Metaphone: Map words to phonetic codes.
    - Example: "fone" → "phone".
3. **Statistical Language Models**
    - Use N-grams to select correction based on context.
    - Example: "I red a book." → "I read a book.".
4. **Neural Network Models**
    - Sequence-to-sequence models correct spelling errors in context.
    - Example: "She luvs NLP" → "She loves NLP".

5. Applications in NLP

- **Search Engines**: "Did you mean...?" suggestions.
- **Word Processors & Editors**: Auto-correct and spell-check.
- **OCR and Speech-to-Text**: Corrects recognition errors.
- **Chatbots & Virtual Assistants**: Improves natural interaction.

MINIMUM EDIT DISTANCE AND APPLICATIONS

- **Minimum Edit Distance** quantifies similarity between two strings using **insertions, deletions, substitutions (and sometimes transpositions)**.
- Widely applied in **spell-checking, search engines, speech recognition, OCR, and translation evaluation**.

1. Introduction

- **Minimum Edit Distance (MED)** is the **minimum number of operations** required to transform one string into another.
- Operations usually include:

1. **Insertion** (add a character)
2. **Deletion** (remove a character)
3. **Substitution** (replace one character with another)

Example:

- "kitten" → "sitting"
  - k → s (substitution)
  - e → i (substitution)
  - add g (insertion)
  - **Total MED = 3**

2. Types of Edit Distance

1. **Levenshtein Distance**
   - Allows insertion, deletion, and substitution (all cost = 1).
   - Most widely used in NLP.
2. **Damerau–Levenshtein Distance**
   - Includes **transposition of adjacent characters**.
   - Example: "caht" → "chat".
3. **Weighted Edit Distance**
   - Different operations may have different costs.
   - Example: substituting "m" with "n" might be cheaper than substituting "m" with "z".

Algorithm for Minimum Edit Distance

Problem

Given two strings:

- `A[1…m]` of length **m**
- `B[1…n]` of length **n**

Find the **minimum number of operations (insert, delete, substitute)** to convert `A` into `B`.

Algorithm (Dynamic Programming Approach)

Algorithm MinimumEditDistance(A, B):

Input: Two strings A of length m, and B of length n

Output: Minimum edit distance between A and B

1. Create a matrix D of size (m+1) × (n+1)

2. Initialize first row and column:

For i = 0 to m:

D[i][0] = i // cost of deleting all characters

For j = 0 to n:

D[0][j] = j // cost of inserting all characters

3. Fill the matrix row by row:

For i = 1 to m:

For j = 1 to n:

If A[i] == B[j]:

cost = 0

Else:

cost = 1

D[i][j] = minimum of:

(a) D[i-1][j] + 1 // deletion

(b) D[i][j-1] + 1 // insertion

(c) D[i-1][j-1] + cost // substitution

4. Return D[m][n] as the minimum edit distance.

Time and Space Complexity

- Time: `O(m × n)`
- Space: `O(m × n)` (can be reduced to `O(min(m, n))` with optimization).

4. Applications of Minimum Edit Distance in NLP

1. **Spell Checking & Correction**
   - Suggest the word closest to the misspelled word.
   - "recieve" → "receive".
2. **Machine Translation Evaluation**
   - Metrics like **WER (Word Error Rate)** use MED to measure translation or transcription accuracy.
3. **Information Retrieval & Search Engines**
   - Fuzzy matching for queries with typos.
   - "tehcnology" → "technology".
4. **Speech Recognition & OCR**
   - Match recognized text with reference text to compute accuracy.
5. **DNA and Protein Sequence Analysis (Bioinformatics)**

- MED measures similarity between genetic sequences.

6. **Plagiarism Detection**
   - Finds similarity between documents using edit operations.

Here's a **worked example** of Minimum Edit Distance using the classic `"kitten" → "sitting"` pair.

1) Dynamic-Programming Table (Levenshtein distance)

Rows = source `"kitten"` (including empty prefix `ε`)

Columns = target `"sitting"` (including `ε`)

Copy code

```
      ε  s  i  t  t  i  n  g
   ┌──────────────────────────
ε  │ 0  1  2  3  4  5  6  7
k  │ 1  1  2  3  4  5  6  7
i  │ 2  2  1  2  3  4  5  6
t  │ 3  3  2  1  2  3  4  5
t  │ 4  4  3  2  1  2  3  4
e  │ 5  5  4  3  2  2  3  4
n  │ 6  6  5  4  3  3  2  3  ← result = 3
```

- Cell `(i, j)` = minimum edits to turn the first `i` chars of `"kitten"` into the first `j` chars of `"sitting"`.
- Final answer is **bottom-right cell = 3**.

2) One optimal sequence of edits (cost = 3)

Start: `kitten`

1. **Substitute** `k → s`: `sitten` (cost 1)
2. **Substitute** `e → i`: `sittin` (cost 1)
3. **Insert** `g` at end: `sitting` (cost 1)

Total cost = **1 + 1 + 1 = 3** ✅

STATISTICAL LANGUAGE MODELS: UNIGRAM, BIGRAM, AND TRIGRAM MODELS

- **Unigram**: Ignores context, simplest.
- **Bigram**: Considers previous word, captures local context.
- **Trigram**: Considers previous two words, better context modeling.
- Statistical LMs are **foundational in NLP** for probability-based text processing.

1. Introduction

- **Language Models (LMs)** assign probabilities to sequences of words.
- They are essential in NLP for:
  - **Speech recognition**
  - **Machine translation**
  - **Text prediction**
  - **Spell correction**

Mathematically, for a word sequence $W = w_1, w_2, ..., w_n$, the probability is:

$$P(W) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, ..., w_{n-1})$$

---

2. Unigram Model

- Considers **each word independently**.
- Probability of a sequence:

$$P(w_1, w_2, ..., w_n) \approx P(w_1) \cdot P(w_2) \cdots P(w_n)$$

- **Advantages**: Simple, easy to compute.
- **Disadvantages**: Ignores word context, low accuracy.

- **Example**:

Word	Frequency	Probability
I	50	0.05
love	10	0.01
NLP	5	0.005

Sequence probability: `P("I love NLP") = 0.05 × 0.01 × 0.005 = 0.0000025`

3. Bigram Model

- Considers **current word conditioned on previous word**.
- Probability:

$$P(w_1, w_2, ..., w_n) \approx P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdots P(w_n|w_{n-1})$$

- Captures some **context**.
- **Example**:

Bigram	Frequency	Probability
I love	8	0.16
love NLP	4	0.5

Sequence probability: `P("I love NLP") = P("I") × P("love"|I) × P("NLP"|love) = 0.05 × 0.16 × 0.5 = 0.004`

4. Trigram Model

- Considers **current word conditioned on previous two words.**
- Probability:

$$P(w_n|w_{n-2}, w_{n-1})$$

- Captures **more context**, better predictions.
- **Example**:

Trigram	Probability
I love NLP	0.25

Sequence probability: `P("I love NLP") = P("I") × P("love"|I) × P("NLP"|I love) = 0.05 × 0.16 × 0.25 = 0.002`

5. Applications

- **Speech Recognition**: Predict next word to correct errors.
- **Machine Translation**: Evaluate fluency of candidate translations.
- **Text Generation / Autocomplete**: Suggest words based on context.
- **Spell Checking**: Choose most probable word in a sequence.

---

6. Limitations

- **Data Sparsity**: Many sequences may not appear in corpus.
- **Higher-order n-grams**: Trigrams, 4-grams → better context but more data needed.
- **Doesn't capture long-range dependencies**: Neural LMs are better for this.

PROCESSING INDIAN LANGUAGES IN NLP

Processing Indian languages in NLP requires **handling rich morphology, script diversity, free word order, and resource scarcity**. Modern approaches combine **rule-based, statistical, and deep learning models** to build NLP applications like MT, sentiment analysis, and speech recognition.

1. Introduction

- **Indian languages** pose unique challenges for NLP due to:
    - **Rich morphology** (inflection, derivation, compounding)
    - **Free word order** (SOV/SVO variations)
    - **Multiple scripts** (Devanagari, Tamil, Telugu, Bengali, etc.)
    - **Code-mixing** (mix of English with Indian languages)
- NLP applications: **machine translation, sentiment analysis, speech recognition, OCR, and text-to-speech systems**.

2. Challenges in Processing Indian Languages

1. **Morphological Complexity**
    - High number of word forms due to inflection.
    - Example (Hindi verbs for root "खेल" /khel/ "play"):
        - खेलता (plays), खेलती (plays – feminine), खेलेगा (will play), खेलेगी (will play – feminine).
2. **Word Segmentation**
    - Some languages (e.g., Hindi, Bengali) have **compound words** without spaces.
    - Example: रेल्वेस्टेशन → रेल्वे + स्टेशन
3. **Ambiguity**
    - Words may have **multiple meanings**.
    - Example: "कला" → art OR skill
4. **Script Diversity**
    - NLP systems must handle **Unicode normalization**.
    - Transliteration between scripts is often required.
5. **Resource Scarcity**
    - Lack of large **annotated corpora, lexicons, and language tools** compared to English.

3. Approaches for Indian Languages

1. **Rule-based Methods**
    - Morphological analyzers, POS taggers based on **grammatical rules**.
2. **Statistical Models**
    - Use **n-grams, HMMs, CRFs** for tagging and translation.
3. **Neural and Deep Learning Models**
    - RNNs, LSTMs, and Transformer-based models (mBERT, IndicBERT, MuRIL).
4. **Transliteration and Transliteration Engines**
    - Convert words between scripts for cross-lingual processing.
5. **Corpus Creation**
    - Indian Language Corpora (ILCI, FIRE, EMILLE, CFILT tools) support NLP research.

4. Applications

- **Machine Translation (MT)**
    - Hindi ↔ English, Tamil ↔ English, Marathi ↔ Hindi.
- **Sentiment Analysis**
    - Analyzing social media content in Indian languages.
- **Spell Checking & Correction**
    - Detecting errors in Indian language texts.
- **Speech Recognition**
    - Voice assistants supporting Hindi, Tamil, Telugu, etc.
- **Optical Character Recognition (OCR)**
    - Converting scanned Indian language documents into editable text.

UNIT II: Word-Level and Syntactic Analysis

Introduction, Part-of-Speech (POS) Tagging: Rule-Based, Stochastic and Transformation-Based Approaches, Hidden Markov Models (HMM) and Maximum Entropy Models for POS Tagging, Context-Free Grammar (CFG) and Constituency Parsing, Treebanks and Normal Forms for Grammar, Top-Down and Bottom-Up Parsing Strategies, CYK Parsing Algorithm, Probabilistic Context-Free Grammars (PCFGs), Feature Structures and Unification.

WORD-LEVEL AND SYNTACTIC ANALYSIS

- **Word-level analysis**: Tokens, POS tags, morphological structure, and named entities.
- **Syntactic analysis**: Parses sentence structure using **constituency or dependency grammars**.
- Together, they enable machines to **understand language structure** for various NLP applications.

1. Introduction

- **Word-level analysis** deals with understanding individual words, their forms, and meanings.
- **Syntactic analysis** (or parsing) deals with **sentence structure** and grammatical relationships between words.
- Both are essential for **semantic understanding, machine translation, question answering, and text generation**.

2. Word-Level Analysis

1. **Tokenization**
    - Breaking text into **words or subwords**.
2. **Morphological Analysis**
    - Examining **word structure** (prefix, root, suffix).
    - Includes **stemming** and **lemmatization**.
3. **Part-of-Speech (POS) Tagging**
    - Assign grammatical categories to words: noun, verb, adjective, etc.
    - Example: "The cat sleeps." → The/DT cat/NN sleeps/VB

4. **Named Entity Recognition (NER)**
    - Identify **proper nouns** like person names, locations, organizations.
    - Example: "Barack Obama was born in Hawaii." → Barack Obama/PER, Hawaii/LOC

3. Syntactic Analysis

- **Goal:** Understand the **structure of sentences** and relationships between words.

1. **Constituency Parsing (Phrase Structure)**
    - Represents sentences as **nested phrases** (NP, VP, etc.)
    - Example: "The cat sleeps" → S → NP (The cat) + VP (sleeps)
2. **Dependency Parsing**
    - Represents **grammatical relationships** as directed links between words.
    - Example: "The cat sleeps" → sleeps is head, cat is subject.
3. **Grammar Formalisms**
    - **Context-Free Grammar (CFG)**: Rules like S → NP VP
    - **Dependency Grammar**: Focus on head-dependent relations

4. Applications of Word-Level and Syntactic Analysis

- **Machine Translation**: Understand structure to preserve meaning.
- **Question Answering**: Identify subjects, objects, and relations.
- **Information Extraction**: Extract structured data from unstructured text.
- **Text Summarization**: Analyze sentence structure for key points.
- **Spell Checking & Grammar Correction**: Detect structural errors.

5. Challenges

- **Ambiguity**: Words can have multiple POS tags.
    - Example: "Book a flight" vs "Read a book"
- **Complex Sentences**: Handling nested or long sentences.
- **Free Word Order Languages**: Harder parsing for languages like Hindi or Japanese.

- **Resource Scarcity**: Limited annotated corpora for many languages.

PART-OF-SPEECH (POS) TAGGING

POS tagging is a **key step in NLP pipelines** that labels each word with its grammatical category. Techniques range from **rule-based to deep learning**, and it is crucial for **parsing, translation, and semantic understanding**.

1. Introduction

- **POS Tagging** is the process of assigning **grammatical categories** (noun, verb, adjective, etc.) to each word in a sentence.
- It is a **fundamental step** in NLP for:
    - **Syntactic parsing**
    - **Information extraction**
    - **Machine translation**
    - **Question answering**

2. Common POS Tags

- **Noun (NN)**: cat, book, city
- **Proper Noun (NNP)**: John, India
- **Verb (VB)**: run, eat
- **Adjective (JJ)**: big, red
- **Adverb (RB)**: quickly, very
- **Determiner (DT)**: the, a, an
- **Pronoun (PRP)**: he, she, it
- **Preposition (IN)**: in, on, at
- **Conjunction (CC)**: and, or, but

Example:
Sentence: "The cat sleeps on the mat."
POS Tagged: The/DT cat/NN sleeps/VB on/IN the/DT mat/NN ./.

3. Techniques for POS Tagging

1. **Rule-Based Tagging**
    - Uses **hand-crafted linguistic rules** and dictionaries.
    - Example Rule: If a word ends with *-ing* → likely a verb.
2. **Stochastic / Probabilistic Tagging**
    - Uses **statistical models** like **Hidden Markov Models (HMMs)**.
    - Assigns tags based on probability of a sequence: P(tag|previous tag).
3. **Transformation-Based Tagging**
    - Uses **Brill's tagger**: learns rules from annotated corpora.
4. **Neural Network / Deep Learning Models**
    - LSTM, Bi-LSTM, Transformers (BERT-based taggers)
    - Capture long-range dependencies and context.

4. Applications of POS Tagging

- **Parsing & Grammar Checking:** Identifies structure and grammatical errors.
- **Information Extraction:** Extracts entities, relations, and facts.
- **Machine Translation:** Helps preserve syntactic structure across languages.
- **Text-to-Speech Systems:** Determines correct pronunciation (e.g., "lead" as noun vs verb).
- **Word Sense Disambiguation:** Helps infer meaning using syntactic context.

5. Challenges

- **Ambiguity:** Words with multiple possible tags.
    - Example: "Can" → verb or modal auxiliary?
- **Unknown Words:** Words not present in the training corpus.
- **Domain Variation:** POS patterns differ across text types (news, social media, medical text).
- **Free Word Order Languages:** Complexity increases for languages like Hindi or Japanese.

RULE-BASED POS TAGGING

Rule-Based POS tagging relies on a **dictionary + handcrafted linguistic rules** to assign grammatical categories. While **interpretable and useful for small-scale systems**, it struggles with ambiguity and scalability.

1. Introduction

- **Rule-Based POS Tagging** assigns a part-of-speech (POS) to each word in a sentence using:
    1. **Lexical knowledge** (dictionary of words and possible tags)
    2. **Hand-crafted linguistic rules**
- One of the **earliest POS tagging methods**, especially before statistical approaches became common.

2. Components of a Rule-Based Tagger

1. **Lexicon / Dictionary**
    - A list of words and their possible POS tags.
    - Example:
        - "book" → noun, verb
        - "run" → noun, verb
2. **Rules**
    - Linguistic rules applied to resolve ambiguity.
    - Types of rules:
        - **Contextual Rules**: Use surrounding words to decide the tag.
            - Example: If a word follows a determiner (DT), tag it as a noun (NN).
        - **Morphological Rules**: Use word suffix/prefix patterns.
            - Example: Words ending in -ing → verb (VBG).
        - **Fallback Rules**: Default to the most common tag in lexicon if no other rules apply.

3. Working of a Rule-Based Tagger

1. **Look up each word** in the dictionary for possible POS tags.
2. **Apply disambiguation rules** based on context or morphology.
3. **Assign the most appropriate tag** to each word.

Example:
Sentence: "The cat sleeps on the mat."

- Lexicon lookup:
  - "The" → DT
  - "cat" → NN
  - "sleeps" → VB, NNS
- Rule application:
  - If previous word = DT, current word = NN → "cat" tagged as NN
  - "sleeps" follows NN → likely VB → tagged as VB

Output: "The/DT cat/NN sleeps/VB on/IN the/DT mat/NN ./"

4. Advantages

- **No training data required**
- Can be **very accurate** for well-defined domains
- Easy to **interpret and debug**

5. Disadvantages

- **Labor-intensive**: Rules must be manually crafted for each language.
- **Limited coverage**: Cannot handle all lexical ambiguities or unknown words.
- **Not scalable** for large corpora or multiple languages.
- **Context limitation**: Cannot capture long-range dependencies like neural models.

6. Applications

- Early **POS tagging systems** in English and other languages.
- Useful in **domain-specific NLP systems** where training data is scarce.

- Basis for **hybrid systems** combining rule-based + statistical models.

STOCHASTIC (PROBABILISTIC) POS TAGGING

1. Introduction

- **Stochastic POS Tagging** assigns part-of-speech tags based on **probabilities derived from annotated corpora**.
- It is also called **statistical POS tagging**.
- Uses **contextual information** to resolve ambiguities that rule-based methods may fail to handle.

2. Core Idea

- POS tagging is modeled as a **sequence labeling problem**:

$$\hat{T} = \arg\max_{T} P(T|W)$$

Where:

- $W = w_1, w_2, ..., w_n$ → sequence of words
- $T = t_1, t_2, ..., t_n$ → sequence of POS tags

- Using **Bayes' theorem**:

$$P(T|W) \propto P(W|T) \cdot P(T)$$

3. Techniques

1. **Hidden Markov Models (HMMs)**
   - Words = observed states
   - POS tags = hidden states
   - Compute:
     - **Transition probabilities**: $P(t_i|t_{i-1})$ → probability of tag given previous tag
     - **Emission probabilities**: $P(w_i|t_i)$ → probability of word given tag
   - Use **Viterbi algorithm** to find the most likely tag sequence.
2. **N-gram Taggers**
   - Use **bigram or trigram probabilities** of tags to capture context.
   - Example:
     - P(VB | DT NN) = probability that a word is verb given previous tags Determiner + Noun
3. **Maximum Entropy Models**
   - Use features like suffixes, capitalization, previous/following words to estimate probabilities.
4. **Neural/Deep Learning Models**
   - LSTM, Bi-LSTM, and Transformer-based models learn **contextual embeddings**.
   - Capture long-range dependencies better than simple HMMs.

4. Example (HMM Tagging)

Sentence: `"The cat sleeps"`

- Tags: DT, NN, VB
- **Step 1:** Compute **transition probabilities**:
  - P(NN|DT) = 0.5
  - P(VB|NN) = 0.6
- **Step 2:** Compute **emission probabilities**:
  - P("cat"|NN) = 0.8
  - P("sleeps"|VB) = 0.7
- **Step 3:** Use **Viterbi** to select tag sequence with maximum probability → `The/DT cat/NN sleeps/VB`

---

5. Advantages

- Can handle **ambiguous words** using context.
- Learns from **data**, adaptable to new domains.
- Scalable for large corpora and multiple languages.

---

6. Disadvantages

- Requires **annotated corpora** for training.
- Probabilities may be **sparse** for rare words or sequences.
- Cannot capture **long-range dependencies** without advanced models.

---

7. Applications

- **POS tagging** in large-scale corpora
- **Syntactic parsing**
- **Machine translation**
- **Speech recognition**

Example: HMM POS Tagging – Step by Step

Sentence: `"The cat sleeps"`

Possible POS tags:

- `The` → DT
- `cat` → NN
- `sleeps` → VB

We are given:

Transition probabilities (tag → tag):

- P(NN | DT) = 0.5
- P(VB | NN) = 0.6

Emission probabilities (word | tag):

- P("cat" | NN) = 0.8
- P("sleeps" | VB) = 0.7

Step 1: Initialization

- Start with **first word** `"The"`.
- `"The"` is usually tagged as **DT** with probability 1 (or P(DT) from corpus).

So the **initial Viterbi probability for DT** = 1.

Step 2: Recursion (Second word `"cat"`)

We calculate probability for each possible tag of `"cat"` using **Viterbi formula**:

$$V_t(s) = \max_{s'}[V_{t-1}(s') \cdot P(s|s') \cdot P(w_t|s)]$$

Where:

- $V_t(s)$ = probability of best path ending in state/tag `s` at time `t`
- s' = previous tag
- w_t = current word

For `"cat"` with tag **NN**:

$$V(”cat”, NN) = V(”The”, DT) \cdot P(NN|DT) \cdot P(”cat”|NN)$$

Plug in values:

$$V(”cat”, NN) = 1 \cdot 0.5 \cdot 0.8 = 0.4$$

Best previous tag = DT (only one option).

Step 3: Recursion (Third word `"sleeps"`)

For `"sleeps"` with tag **VB**:

$$V(\text{"}sleeps\text{"}, VB) = V(\text{"}cat\text{"}, NN) \cdot P(VB|NN) \cdot P(\text{"}sleeps\text{"}|VB)$$

Plug in values:

$$V(\text{"}sleeps\text{"}, VB) = 0.4 \cdot 0.6 \cdot 0.7 = 0.168$$

Best previous tag = NN

Step 4: Termination

- The **Viterbi probability for the full sequence** = 0.168
- **Best path** (sequence of tags with maximum probability):

$$\text{The/DT} \rightarrow \text{cat/NN} \rightarrow \text{sleeps/VB}$$

Step 5: Interpretation

- The Viterbi algorithm **chooses the most probable tag sequence** based on:
    1. **Transition probabilities** (grammar context)
    2. **Emission probabilities** (likelihood of word given tag)
- Final tagging respects both **word meaning** and **syntactic context**.

Word	Tag	Viterbi Probability	Previous Tag
The	DT	1	–
cat	NN	0.4	DT
sleeps	VB	0.168	NN

TRANSFORMATION-BASED POS TAGGING (BRILL TAGGER)

- **Transformation-Based POS Tagging** starts with an **initial tagging**, then **iteratively corrects errors** using learned rules from training data.

- It is **interpretable, accurate**, and widely used in hybrid NLP systems.

1. Introduction

- **Transformation-Based Learning (TBL)** is a **hybrid approach** to POS tagging.
- Introduced by **Eric Brill (1992)**.
- Combines the **accuracy of rule-based tagging** with **learning from annotated corpora**.
- Tags are initially assigned using a **simple method**, then **iteratively improved** by applying learned transformations.

2. Core Idea

1. Start with **initial tags** (e.g., from a lexicon or default most frequent tag).
2. **Learn rules** that correct errors in the tagged text using a **training corpus**.
3. Apply the **learned transformation rules** to new text.

- Each **transformation rule** has the form:

"Change tag **X** to **Y** when the word's context satisfies condition **C**."

3. Example of Transformation Rules

- **Initial tagging:** Most frequent tag per word
- **Transformation rules learned from data:**
    1. Change NN → VB if the previous word is "to"
        - Example: "to run" → run/VB
    2. Change NN → JJ if the next word is "car"
        - Example: "fast car" → fast/JJ car/NN
    3. Change VB → NN if the word ends with -ing and previous word is the

4. Advantages

- **High accuracy**: Can approach state-of-the-art POS tagging performance.
- **Interpretable rules**: Each rule is readable and understandable.
- **Requires less data** than fully statistical methods.
- **Combines advantages** of rule-based and statistical approaches.

5. Disadvantages

- **Training can be time-consuming** for large corpora.
- May **not generalize well** to highly different domains.
- Complexity increases if too many rules are learned.

6. Applications

- **POS Tagging** in English and other languages.
- Basis for **hybrid NLP systems** that combine rules and statistical models.
- Useful for **error correction in tagging** ambiguous or rare words.

HIDDEN MARKOV MODELS (HMM)

- **HMMs model sequential data in NLP where the observations (words) are visible but states (tags) are hidden.**
- **Core components: transition, emission, and initial probabilities.**
- **The Viterbi algorithm decodes the best sequence of hidden states.**

1. Introduction

- **HMMs** are **statistical models** for sequences where the system is assumed to be a **Markov process with hidden states**.
- Widely used in NLP tasks such as:
  - **POS tagging**
  - **Speech recognition**
  - **Named Entity Recognition (NER)**
- Key idea: **We observe outputs (words), but the underlying states (tags) are hidden.**

2. Components of an HMM

1. **States (Hidden)**
   - Example: POS tags like `NN, VB, JJ`
   - Denoted as $S = \{s_1, s_2, ..., s_N\}$
2. **Observations**
   - Words in a sentence.
   - Denoted as $O = \{o_1, o_2, ..., o_T\}$
3. **Transition Probabilities**
   - Probability of moving from state s_i to s_j:

$$a_{ij} = P(s_j|s_i)$$

4. **Emission Probabilities**
   - Probability of observing word o_t in state s_j:

$$b_j(o_t) = P(o_t|s_j)$$

5. **Initial Probabilities**
   - Probability of starting in state s_i:

$$\pi_i = P(s_i \text{ at t=1})$$

3. HMM Problems

1. **Evaluation**
   - Compute the probability of a sequence of observations:

$$P(O|\lambda)$$

2. **Decoding**
   - Find the **most likely sequence of hidden states** given the observations (Viterbi algorithm).
3. **Learning**
   - Estimate HMM parameters (transition and emission probabilities) from training data (Baum-Welch algorithm).

4. HMM in POS Tagging

- Words = observations
- POS tags = hidden states

Example: Sentence: `"The cat sleeps"`

- **Hidden states:** DT, NN, VB
- **Transition probabilities:** P(NN|DT) = 0.5, P(VB|NN) = 0.6
- **Emission probabilities:** P("cat"|NN) = 0.8, P("sleeps"|VB) = 0.7
- Use **Viterbi algorithm** to find best tag sequence → `"The/DT cat/NN sleeps/VB"`

5. Advantages

- Captures sequential dependencies between tags.
- Probabilistic approach handles ambiguity naturally.
- Can be trained from annotated corpora.

6. Disadvantages

- Assumes Markov property (current state depends only on previous state), ignoring long-range dependencies.
- Emission probabilities may be sparse for rare words.
- Requires large annotated corpora for accurate parameter estimation.

7. Applications

- POS Tagging
- Speech Recognition
- Named Entity Recognition (NER)
- Machine Translation (alignment modeling)
- Bioinformatics (gene sequence modeling)

- **Each cell in the Viterbi lattice = probability of best path ending in that tag.**
- **Transition probabilities = likelihood of one tag following another.**
- **Emission probabilities = likelihood of a word being generated by a tag.**
- **The Viterbi algorithm combines these to find the most probable sequence.**

HMM POS Tagging Example – Step by Step

Sentence: `"The cat sleeps"`

Hidden states (POS tags): `DT, NN, VB`

Observations (words): `"The", "cat", "sleeps"`

Given Probabilities:

- **Transition probabilities (tag → tag):**
  - P(NN|DT) = 0.5
  - P(VB|NN) = 0.6
- **Emission probabilities (word|tag):**
  - P("cat"|NN) = 0.8
  - P("sleeps"|VB) = 0.7
- **Initial tag probability:** Assume P(DT at start) = 1

Step 1: Initialization

- First word = `"The"`
- Only possible tag = **DT** (given probability 1)
- **Viterbi probability for DT at t=1:**

$$V_1(DT) = P(DT) = 1$$

Step 2: Recursion (Second word `"cat"`)

We calculate the Viterbi probability for each possible tag of `"cat"` using:

$$V_t(s) = \max_{s'}[V_{t-1}(s') \cdot P(s|s') \cdot P(w_t|s)]$$

- For `"cat"` with tag **NN**:

$$V_2(NN) = V_1(DT) \cdot P(NN|DT) \cdot P(\text{"}cat\text{"}|NN)$$

$$V_2(NN) = 1 \cdot 0.5 \cdot 0.8 = 0.4$$

- Best previous tag = **DT**

Step 3: Recursion (Third word `"sleeps"`)

- For `"sleeps"` with tag **VB**:

$$V_3(VB) = V_2(NN) \cdot P(VB|NN) \cdot P(\text{"}sleeps\text{"}|VB)$$

$$V_3(VB) = 0.4 \cdot 0.6 \cdot 0.7 = 0.168$$

- Best previous tag = **NN**

Step 4: Termination

- Maximum Viterbi probability at last word = 0.168
- Traceback the **best previous tags**:

Word	Tag	Previous Tag	Viterbi Probability
The	DT	–	1
cat	NN	DT	0.4
sleeps	VB	NN	0.168

- **Final best sequence:**

$$\text{The/DT} \rightarrow \text{cat/NN} \rightarrow \text{sleeps/VB}$$

Step 5: Interpretation

1. Start with initial probability of first tag.
2. Multiply previous Viterbi probability × transition probability × emission probability at each step.
3. Choose the maximum probability path at each step.
4. Trace back to get the most likely sequence of POS tags.

MAXIMUM ENTROPY (MAXENT) MODELS FOR POS TAGGING

- MaxEnt models compute **probabilities of tags using rich features** from words and context.
- Tag with **maximum probability** is assigned.
- Advantages over HMM: **flexible, feature-rich, no strong independence assumptions**.

1. Introduction

- Maximum Entropy Models are probabilistic models used in NLP for sequence labeling tasks like POS tagging.
- Based on the principle of maximum entropy: among all probability distributions satisfying given constraints, choose the one with highest entropy (most uniform / least biased).

- Advantages: Can incorporate diverse features, not limited to sequential dependencies like HMMs.

2. Core Idea

- POS tagging: Assign a **tag** t to a word w given its **context** C.

$$P(t|w,C) = \frac{1}{Z(w,C)} \exp\left(\sum_i \lambda_i f_i(t,w,C)\right)$$

Where:

- $f_i(t,w,C)$ = feature function (indicator functions)
- λ_i = weight of the feature learned from data
- $Z(w,C)$ = normalization factor ensuring probabilities sum to 1

3. Features Used in POS Tagging

MaxEnt models allow **rich features**, such as:

1. **Lexical Features**
   - Current word, suffixes, prefixes, capitalization
   - Example: If word ends in -ing, likely VB
2. **Contextual Features**
   - Previous and next words or tags
   - Example: If previous word = to, current word → VB
3. **Orthographic Features**
   - Numbers, hyphens, punctuation
   - Example: If word contains digits → NN (numeric)
4. **Combined Features**
   - Previous tag + current word, word shape, etc.

4. How MaxEnt POS Tagging Works

1. **Training Phase**

- Input: Annotated corpus (words + correct tags)
- Learn weights (λi\lambda_iλi) for each feature to maximize likelihood

2. **Tagging Phase**
   - For each word:
     - Extract features from word and context
     - Compute probabilities of all possible tags
     - Assign tag with **highest probability**

5. Example

Sentence: "The cat sleeps"

Features for "sleeps":

- Current word = "sleeps"
- Previous word = "cat"
- Previous tag = "NN"
- Word suffix = "ps"

Compute probability for each candidate tag:

- P(VB | features) = 0.75
- P(NN | features) = 0.10
- P(JJ | features) = 0.05

→ Assign **VB** as tag for "sleeps" because it has **highest probability**.

6. Advantages

- Can incorporate **arbitrary, overlapping features**.
- Does **not require independence assumptions** like HMMs.
- Often achieves **higher accuracy** in POS tagging than HMMs.

7. Disadvantages

- Computationally **more expensive** than HMMs for large feature sets.

- Requires **good feature engineering** (though deep learning reduces this need).
- Needs a **large annotated corpus** for robust performance.

8. Applications

- **POS Tagging** (main application)
- **Named Entity Recognition (NER)**
- **Chunking / Shallow Parsing**
- **Information Extraction**

CONTEXT-FREE GRAMMAR (CFG)

- CFG is a **formal grammar** with rules defining how sentences are structured.
- Consists of **non-terminals, terminals, production rules, and a start symbol**.
- Widely used in **parsing and syntactic analysis** in NLP.

1.	S → NP VP	5.	NAME → John
2.	VP → V NP	6.	V → ate
3.	NP → NAME	7.	ART → the
4.	NP → ART N	8.	N → cat

Grammar 3.2 A simple grammar

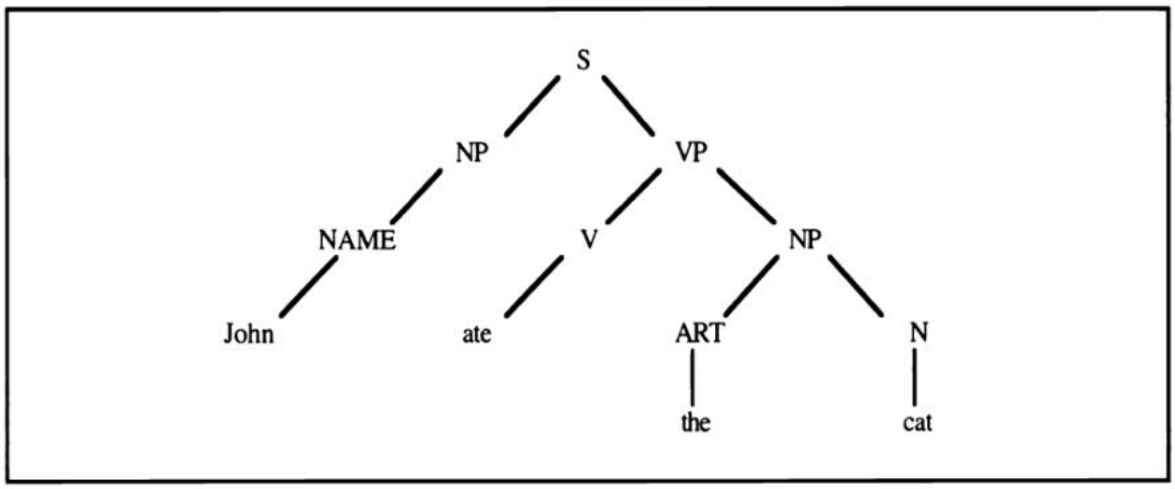


Figure 3.1 A tree representation of *John ate the cat*

1. Introduction

- **CFG** is a formal grammar used to describe **syntactic structures of languages**.
- Widely used in **parsing sentences**, **syntactic analysis**, and **compiler design**.
- A CFG consists of **rules that describe how sentences can be generated from a set of symbols**.

2. Components of a CFG

A CFG is defined as a 4-tuple $G = (N, \Sigma, P, S)$:

1. **N (Non-terminal symbols)**
   - Abstract symbols representing syntactic categories (e.g., S, NP, VP, Det, Noun, Verb).
2. **Σ (Terminal symbols)**
   - Actual words in the language (e.g., `"cat"`, `"sleeps"`, `"the"`).
3. **P (Production rules)**
   - Rules describing how non-terminals can be expanded into terminals or other non-terminals.
   - Example: `S → NP VP`
4. **S (Start symbol)**
   - Typically `S` representing a complete sentence.

3. Example CFG

Sentence: `"The cat sleeps"`

Non-terminals (N): `S, NP, VP, Det, N, V`

Terminals (Σ): `"the", "cat", "sleeps"`

Production Rules (P):

1. S → NP VP
2. NP → Det N
3. VP → V
4. Det → "the"
5. N → "cat"
6. V → "sleeps"

Start Symbol: S

4. Derivation Example

Step-by-step derivation of sentence "The cat sleeps":

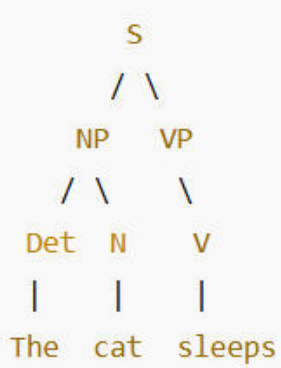
1. Start with `S`
2. Apply rule `S → NP VP` → `NP VP`
3. Apply rule `NP → Det N` → `Det N VP`
4. Apply rules for terminals:
   - `Det → "The"`
   - `N → "cat"`
   - `VP → V → "sleeps"`

Final derivation: `"The cat sleeps"` ✅

5. Parse Tree

A **parse tree** represents the structure of a sentence according to CFG:

mathematica



6. Advantages of CFG

- Can model **hierarchical structure** of sentences.
- Supports **automated parsing**.
- Useful in **NLP tasks** like POS tagging, syntax checking, and machine translation.

7. Limitations of CFG

- Cannot easily handle **long-distance dependencies** (e.g., subject-verb agreement across clauses).
- May not capture **all linguistic nuances**, especially in free-word-order languages.
- Ambiguity can lead to **multiple parse trees** for the same sentence.

8. Applications in NLP

- **Syntactic parsing**
- **Grammar checking**
- **Machine Translation**
- **Question Answering**
- **Information Extraction**

CONSTITUENCY PARSING

- Constituency Parsing identifies **phrases and their hierarchical relationships** in a sentence.
- Uses **CFG or probabilistic methods** to generate a **parse tree**.
- Provides **structured syntactic information** for various NLP tasks.

```
a:      (CAT ART
         ROOT A1
         AGR 3s)
be:     (CAT V
         ROOT BE1
         VFORM base
         IRREG-PRES +
         IRREG-PAST +
         SUBCAT (_adjp _np))
cry:    (CAT V
         ROOT CRY1
         VFORM base
         SUBCAT _none)
dog:    (CAT N
         ROOT DOG1
         AGR 3s)
fish:   (CAT N
         ROOT FISH1
         AGR (3s 3p)
         IRREG-PL +)
happy:  (CAT ADJ
         SUBCAT _vp:inf)
he:     (CAT PRO
         ROOT HE1
         AGR 3s)
is:     (CAT V
         ROOT BE1
         VFORM pres
         SUBCAT (_adjp _np)
         AGR 3s)
Jack:   (CAT NAME
         AGR 3s)
man:    (CAT N1
         ROOT MAN1
         AGR 3s)
men:    (CAT N
         ROOT MAN1
         AGR 3p)
saw:    (CAT N
         ROOT SAW1
         AGR 3s)
saw:    (CAT V
         ROOT SAW2
         VFORM base
         SUBCAT _np)
saw:    (CAT V
         ROOT SEE1
         VFORM past
         SUBCAT _np)
see:    (CAT V
         ROOT SEE1
         VFORM base
         SUBCAT _np
         IRREG-PAST +
         EN-PASTPRT +)
seed:   (CAT N
         ROOT SEED1
         AGR 3s)
the:    (CAT ART
         ROOT THE1
         AGR (3s 3p))
to:     (CAT TO)
want:   (CAT V
         ROOT WANT1
         VFORM base
         SUBCAT (_np _vp:inf _np_vp:inf))
was:    (CAT V
         ROOT BE1
         VFORM past
         AGR (1s 3s)
         SUBCAT (_adjp _np))
were:   (CAT V
         ROOT BE
         VFORM past
         AGR (2s 1p 2p 3p)
         SUBCAT (_adjp _np))
```

Figure 4.6 A lexicon

1. Introduction

- **Constituency Parsing** (or **Phrase Structure Parsing**) analyzes a sentence to identify its **constituent parts**, such as **noun phrases (NP), verb phrases (VP), and prepositional phrases (PP)**.
- Based on **Context-Free Grammar (CFG)**.
- Helps machines understand **hierarchical structure** of sentences.

2. Core Concepts

1. **Constituents**
   - A group of words that function as a **single unit** in a sentence.
   - Example: `"The cat"` → Noun Phrase (NP), `"sleeps on the mat"` → Verb Phrase (VP)
2. **Parse Tree**
   - Represents hierarchical structure of sentence using **nodes for constituents** and **leaves for words**.

3. Example

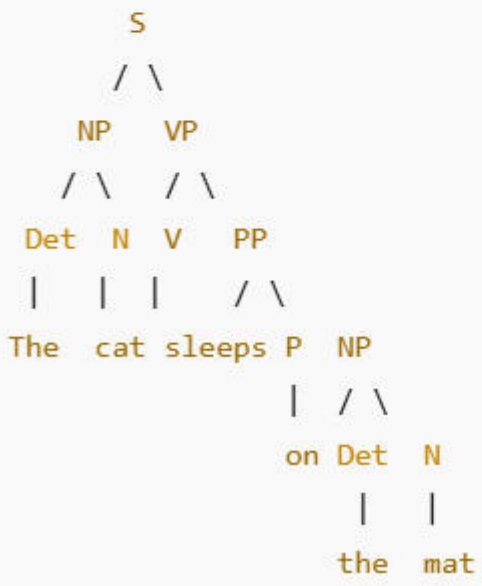
Sentence: `"The cat sleeps on the mat"`

Constituents:

- NP → `"The cat"`
- VP → `"sleeps on the mat"`
- PP → `"on the mat"`

Parse Tree:

mathematica



4. Steps in Constituency Parsing

1. **Tokenization** – Break sentence into words.
2. **POS Tagging** – Assign POS tags to words.
3. **Grammar Application** – Use CFG rules to combine words into constituents.
4. **Tree Construction** – Build a hierarchical parse tree.

---

5. Methods

1. **Rule-Based / Grammar-Based Parsing**
   - Uses manually created CFG rules.
   - Example: Earley Parser, CKY Parser.
2. **Statistical / Probabilistic Parsing**
   - Uses Probabilistic CFG (PCFG) to handle ambiguity.
   - Example: P("NP → Det N") = 0.8
3. **Neural Network-Based Parsing**
   - Uses embeddings and deep learning to predict constituency structure.
   - Example: Neural CRF, Transformer-based parsers.

6. Advantages

- Captures **hierarchical and phrasal structure**.
- Useful for **machine translation, summarization, and question answering**.
- Provides **interpretable syntactic analysis**.

7. Limitations

- **Ambiguity:** Sentences may have multiple valid parse trees.
- **Complexity:** Parsing long sentences can be computationally expensive.
- Less effective for **free-word-order languages** unless probabilistic or neural methods are used.

8. Applications

- **Syntactic analysis** for NLP pipelines.

- **Machine Translation:** Helps maintain sentence structure.
- **Question Answering & Information Extraction:** Identify subject, object, and phrases.
- **Grammar Checking & Correction**

TREEBANKS AND NORMAL FORMS FOR GRAMMAR

- **Treebanks:** Annotated corpora with POS tags and parse trees for training and evaluation.
- **Normal forms:** Standardized representations of grammar rules (CNF, GNF) to simplify parsing.
- Both are **essential for building and evaluating NLP parsing systems**.

1. Treebanks

Definition:

- A **treebank** is a **corpus of sentences annotated with syntactic or semantic parse trees**.
- Provides **gold-standard examples** for training and evaluating parsers.

Purpose in NLP:

- Helps **train statistical parsers**.
- Used in **POS tagging, syntactic parsing, and grammar evaluation**.
- Enables **comparative evaluation** of parsing algorithms.

Examples of Treebanks:

1. **Penn Treebank (PTB)** – English, widely used in NLP research.
2. **Universal Dependencies (UD) Treebanks** – multilingual, dependency-based annotation.
3. **NEGRA Treebank** – German, constituency-based.

Structure:

article (ART) *the* and a common noun (N) *cat*. In list notation this same structure could be represented as

```
(S   (NP   (NAME John))
     (VP   (V ate)
           (NP   (ART the)
                 (N cat))))
```

- Each sentence is annotated with:
    - POS tags for each word
    - Phrase structure or dependency structure
- Stored as **bracketed trees** or **dependency graphs**

Example (Bracketed):
Sentence: "The cat sleeps"

(S

(NP (DT The) (NN cat))

(VP (VB sleeps))

)

2. Normal Forms for Grammar

Definition:

- **Normal forms** are standardized ways to represent grammar rules.
- Simplify parsing and grammar analysis.

Common Normal Forms:

1. **Chomsky Normal Form (CNF)**
    - Each production rule is either:
        1. A → BC (two non-terminals)
        2. A → a (a single terminal)
    - Example:

- Original: S → NP VP
- CNF: S → X VP, X → NP

2. **Greibach Normal Form (GNF)**
    - Each production starts with a **terminal** followed by **zero or more non-terminals**.
    - Example: S → aA B

Why Normal Forms Are Useful:

- **Simplify parsing algorithms** (like CYK parser uses CNF).
- Reduce **ambiguity and complexity** in automated parsing.
- Facilitate **formal proofs and computational efficiency**.

3. Applications in NLP

- **Treebanks**
    - Train and evaluate **statistical and neural parsers**.
    - Serve as **gold-standard datasets**.
- **Normal Forms**
    - Simplify **syntactic parsing algorithms**.
    - Used in **compiler design, CFG parsing, and NLP education**.

TOP-DOWN AND BOTTOM-UP PARSING STRATEGIES

Parsing strategies determine **how a parser constructs a parse tree** for a sentence based on a grammar. The two main strategies are **Top-Down** and **Bottom-Up Parsing**.

- **Top-Down Parsing:** Start from root, expand non-terminals, match input.
- **Bottom-Up Parsing:** Start from input tokens, combine into constituents, reach root.
- Both produce the **same parse tree** but follow opposite directions.

1. Top-Down Parsing

S
⇒ NP VP (rewriting S)
⇒ NAME VP (rewriting NP)
⇒ John VP (rewriting NAME)
⇒ John V NP (rewriting VP)
⇒ John ate NP (rewriting V)
⇒ John ate ART N (rewriting NP)
⇒ John ate the N (rewriting ART)
⇒ John ate the cat (rewriting N)

Definition:

- **Top-Down Parsing** starts from the **start symbol (S)** and tries to derive the **input sentence** by recursively expanding non-terminals using grammar rules.
- Works **from root to leaves** of the parse tree.

Characteristics:

- Predictive approach: tries to match input tokens by applying rules.
- Can use **lookahead** to reduce backtracking (e.g., in **LL parsers**).
- Can suffer from **left recursion** and may require grammar modification.

Example:

Grammar:

```mathematica
S → NP VP
NP → Det N
VP → V
Det → the
N → cat
V → sleeps
```

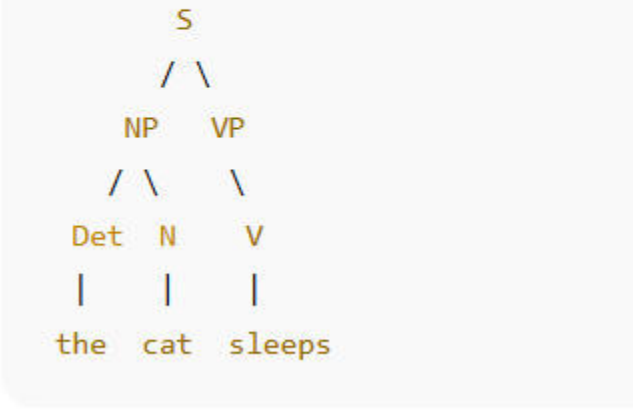
Sentence: `"The cat sleeps"`

Top-Down Parse Sequence:

1. Start with `S`
2. Apply `S → NP VP` → `NP VP`
3. Expand `NP → Det N` → `Det N VP`
4. Match terminals: `Det → the`, `N → cat`, `VP → V → sleeps`

Parse Tree:

mathematica



3. Bottom-Up Parsing

In a **bottom-up strategy**, you start with the words in the sentence and use the rewrite rules backward to reduce the sequence of symbols until it consists solely of S. The left-hand side of each rule is used to rewrite the symbol on the right-hand side. A possible bottom-up parse of the sentence *John ate the cat* is

⇒ NAME ate the cat	(rewriting John)
⇒ NAME V the cat	(rewriting ate)
⇒ NAME V ART cat	(rewriting the)
⇒ NAME V ART N	(rewriting cat)
⇒ NP V ART N	(rewriting NAME)
⇒ NP V NP	(rewriting ART N)
⇒ NP VP	(rewriting V NP)
⇒ S	(rewriting NP VP)

Definition:

- **Bottom-Up Parsing** starts from the **input sentence (leaves)** and tries to construct the **parse tree up to the start symbol**.
- Works **from leaves to root**.

Characteristics:

- Data-driven approach: combines words into phrases, then phrases into sentences.

- Avoids left-recursion problems.
- Can be implemented using **shift-reduce parsers**.

Example:

Sentence: `"The cat sleeps"`

Bottom-Up Parse Sequence:

1. Start with words: `the cat sleeps`
2. Reduce `the` → `Det`
3. Reduce `cat` → `N`
4. Combine `Det N` → `NP`
5. Reduce `sleeps` → `V`
6. Combine `V` → `VP`
7. Combine `NP VP` → `S`

Parse Tree: Same as top-down.

Feature	**Top-Down Parsing**	**Bottom-Up Parsing**
Approach	Root → Leaves (Start → Input)	Leaves → Root (Input → Start)
Predictive	Yes	No
Handles Left Recursion	Poor	Good
Backtracking Needed	Often	Sometimes
Example Implementation	Recursive Descent, LL Parser	Shift-Reduce, LR Parser

4. Applications

- **Top-Down Parsing:** Suitable for **predictive parsers** and simple CFGs.

- **Bottom-Up Parsing:** Used in **LR parsers**, **shift-reduce parsers**, and large-scale NLP parsing systems.
- Both strategies are **foundational for syntactic analysis, machine translation, and grammar checking**.

CYK PARSING ALGORITHM

CYK (Cocke–Younger–Kasami) Parsing Algorithm

- **CYK Algorithm** is a **bottom-up parser** using **CNF CFGs**.
- Fills a **triangular table** with non-terminals that generate substrings.
- Efficiently checks **sentence validity** and can construct parse trees.

1. Introduction

- **CYK Algorithm** is a **bottom-up parsing algorithm** for **Context-Free Grammars (CFG)**.
- Works only with grammars in **Chomsky Normal Form (CNF)**.
- Efficiently determines **if a sentence can be generated by a CFG** and produces a **parse tree**.
- Widely used in **NLP for syntactic parsing**.

2. Requirements

- Input:
  1. Sentence: a sequence of words $w_1, w_2, \ldots, w_n$
  2. CFG in **Chomsky Normal Form (CNF)**
     - Rules of the form:
       - A → BC (two non-terminals)
       - A → a (single terminal)
- Output:
  - A parse table showing which non-terminals can generate which substrings.
  - Optional parse tree(s) for the sentence.

3. Algorithm Steps

Step 1: Initialize Table

- Create a triangular table $T[i, j]$ for all substrings $w_i ... w_j$.
- Fill **diagonal cells** with non-terminals that generate each word:

$$T[i, i] = \{A | A \to w_i \text{ in grammar}\}$$

Step 2: Fill Table (Bottom-Up)

- For each substring length $l = 2$ to n:
  - For each start index $i = 1$ to $n - l + 1$:
    - End index $j = i + l - 1$
    - For each split point $k = i$ to $j - 1$:
      - If rule $A \to BC$ exists and $B \in T[i, k], C \in T[k + 1, j]$, then add A to $T[i, j]$

Step 3: Check Start Symbol

- If start symbol $S \in T[1, n]$, sentence is **grammatically correct** according to CFG.

4. Example

Grammar in CNF:

```
mathematica

S → NP VP
NP → Det N
VP → V
Det → the
N → cat
V → sleeps
```

Sentence: `"the cat sleeps"`

Step 1: Fill diagonals (length 1):

Word	Non-Terminals
the	Det
cat	N
sleeps	V

Step 2: Fill length 2 substrings:

- `"the cat"` : Det + N → NP → add NP to T[1,2]
- `"cat sleeps"` : N + V → no rule → T[2,3] empty

Step 3: Fill length 3 substring (full sentence):

- `"the cat sleeps"` : NP (T[1,2]) + V (T[3,3]) → S → add S to T[1,3]

✅ S ∈ T[1,3] → sentence is valid

5. Advantages

- Efficient $O(n^3 \times |G|)$ algorithm for CFG parsing.
- Systematic **bottom-up parsing** method.
- Can produce **all possible parse trees**.

---

6. Disadvantages

- Requires **grammar in CNF** → conversion needed.
- Computationally **expensive for very long sentences**.
- Less intuitive than top-down parsers for small grammars.

---

7. Applications

- **Syntactic parsing** in NLP
- **Grammar checking**
- **Machine translation**
- **Bioinformatics (sequence parsing)**

PROBABILISTIC CONTEXT-FREE GRAMMARS (PCFGS)

- **PCFGs** = CFG + probabilities.
- **Probabilities** allow choosing the **most likely parse** among multiple possibilities.
- Essential in **statistical NLP** for disambiguation and robust parsing.

1. Introduction

- **PCFGs** are an extension of **Context-Free Grammars (CFGs)** that assign **probabilities to production rules**.
- Used to handle **ambiguity in natural language**, e.g., multiple parse trees for a sentence.
- Widely used in **statistical parsing, machine translation, and speech recognition**.

2. Components of a PCFG

A PCFG is a 5-tuple $G = (N, \Sigma, P, S, Prob)$:

1. **N (Non-terminals)** – Syntactic categories like `S, NP, VP, Det, N, V`.
2. **Σ (Terminals)** – Words in the language.
3. **P (Production Rules)** – CFG rules.
4. **S (Start Symbol)** – Typically `S` for sentence.
5. **Prob (Rule Probabilities)** – Each rule A → β has probability:

$$P(A \to \beta) = \text{probability of choosing this expansion of A}$$

- **Constraint**: Sum of probabilities for all rules with same left-hand side = 1.

$$\sum_{\forall \beta} P(A \to \beta) = 1$$

3. Example PCFG

Grammar for `"the cat sleeps"`:

Rule	Probability
S → NP VP	1.0
NP → Det N	0.9
NP → N	0.1
VP → V	1.0
Det → the	1.0
N → cat	0.5
N → dog	0.5
V → sleeps	1.0

4. Parse Tree Probability

- Probability of a parse tree = **product of probabilities of all applied rules.**

Example: Parse tree for `"the cat sleeps"`:

mathematica Copy code

```
        S
       / \
     NP   VP
    / \    \
  Det  N    V
   |   |    |
 the  cat  sleeps
```

- Probability = P(S→NP VP) × P(NP→Det N) × P(Det→the) × P(N→cat) × P(VP→V) × P(V→sleeps)

$$P = 1.0 \times 0.9 \times 1.0 \times 0.5 \times 1.0 \times 1.0 = 0.45$$

---

5. Advantages

- **Handles ambiguity** by selecting the most probable parse.
- Can be trained from **treebanks** to capture real-language usage.
- Provides a **probabilistic ranking of parse trees**.

6. Disadvantages

- Accuracy depends on **quality and size of annotated corpus**.
- Assumes **independence of rules**, which may not capture long-range dependencies.
- Computationally more expensive than CFG parsing.

7. Applications

- **Statistical Syntactic Parsing** (disambiguation of multiple parse trees)
- **Machine Translation** (syntax-based translation)
- **Speech Recognition** (probable syntactic structure for word sequences)
- **Information Extraction** (finding structured information from text)

How PCFG Probabilities Are Computed

1. PCFG Basics

- A **PCFG (Probabilistic Context-Free Grammar)** assigns a **probability to each production rule**.
- The probability reflects **how likely a particular expansion of a non-terminal occurs in actual language usage**.
- Constraint: For any non-terminal A, the sum of probabilities of all its expansions = 1.

$$\sum_{\forall \beta} P(A \to \beta) = 1$$

2. Example Grammar

Given rules for sentence `"the cat sleeps"`:

Rule	Probability
S → NP VP	1.0
NP → Det N	0.9
NP → N	0.1
VP → V	1.0
Det → the	1.0
N → cat	0.5
N → dog	0.5
V → sleeps	1.0

3. How These Probabilities Are Derived

Step 1: Collect Data

- Probabilities are usually computed from a **treebank** (annotated corpus of sentences with parse trees).
- Count how often each production rule is used.

Example: Suppose we have the following corpus for NP:

NP Rule	Count in corpus
NP → Det N	90
NP → N	10

- Total NP expansions = 90 + 10 = 100

Step 2: Compute Probability

$$P(\text{NP} \rightarrow \text{Det N}) = \frac{\text{Count}(\text{NP} \rightarrow \text{Det N})}{\text{Total NP expansions}} = \frac{90}{100} = 0.9$$

$$P(\text{NP} \rightarrow \text{N}) = \frac{10}{100} = 0.1$$

Step 3: Repeat for Other Non-terminals

- For N → cat / N → dog:

Suppose in corpus:

- N → cat occurs 50 times
- N → dog occurs 50 times
- Total N expansions = 100

$$P(N \rightarrow cat) = \frac{50}{100} = 0.5, \quad P(N \rightarrow dog) = 0.5$$

- Terminal rules often have probability 1 if only one terminal occurs for a non-terminal.
  - Example: Det → the → 1.0
- Start symbol S usually has only **one production** in simple grammar → probability = 1.0

4. Summary of Steps

1. **Collect counts** of each rule in a treebank or corpus.
2. **Compute probability** = (count of rule) / (total counts of all expansions for that non-terminal).
3. Ensure probabilities for **all expansions of a non-terminal sum to 1**.

✅ Key Points

- Probabilities **reflect real usage frequency** in a corpus.
- PCFG helps **disambiguate parses** by favoring more frequent constructions.
- Without a corpus, probabilities can be **manually estimated** for small examples (as in textbooks).

FEATURE STRUCTURES AND UNIFICATION

- **Feature structures:** Attribute-value pairs representing linguistic info.
- **Unification:** Merging FS consistently; fails if conflicts exist.
- Essential for **agreement checking, parsing, and constraint-based grammar systems**.

1. Introduction

- **Feature Structures (FS)** are a way to represent **rich linguistic information** about words, phrases, or syntactic categories.
- Common in **unification-based grammars** like **Head-Driven Phrase Structure Grammar (HPSG)**.
- They help encode **syntactic, semantic, morphological, and agreement information** compactly.

2. What is a Feature Structure?

- A **feature structure** is essentially a **set of attribute-value pairs**.
- Example: For the word "dogs":

[

CATEGORY: Noun

NUMBER: Plural

PERSON: 3rd

GENDER: Neutral

]

- Attributes = CATEGORY, NUMBER, PERSON, GENDER
- Values = Noun, Plural, 3rd, Neutral
- Can also include **nested feature structures**, e.g., for a verb phrase:

[

CATEGORY: VP

HEAD: [CATEGORY: V, TENSE: Present, NUMBER: Singular]

SUBJ: [CATEGORY: NP, NUMBER: Singular]

]

3. Unification

- **Unification** is the process of **merging two feature structures** consistently.
- If two structures **agree on shared features**, they are **unified**.
- If they **conflict**, unification **fails**.

4. Example of Unification

FS1 (Subject NP):

[CATEGORY: NP, NUMBER: Singular]

FS2 (Verb VP Head):

[CATEGORY: V, NUMBER: Singular]

Unifying NP and VP features for agreement:

- Both have NUMBER = Singular → compatible → unification succeeds.
- If VP had NUMBER = Plural → conflict → unification fails.

5. Advantages of Feature Structures

1. **Expressive** – Can capture multiple linguistic properties in one structure.
2. **Handles agreement constraints** – E.g., subject-verb agreement.

3. **Supports modular grammars** – Features can be added or reused.
4. **Basis for unification-based parsers** – Widely used in HPSG, LFG, and TAG.

6. Applications in NLP

- **Syntactic parsing** – unification ensures features match across constituents.
- **Morphological analysis** – number, gender, tense, person.
- **Semantic interpretation** – features can include semantic roles.
- **Constraint-based grammars** – HPSG, Lexical Functional Grammar (LFG).

UNIT III - TEXT CLASSIFICATION AND INFORMATION RETRIEVAL

Naïve Bayes Classifier for Text Classification, Training and Optimization for Sentiment Analysis, Information Retrieval: Basic Concepts and Design Features, Information Retrieval Models: Classical, Non-Classical, and Alternative Models, Cluster Model, Fuzzy Model, and LSTM-Based Information, Retrieval, Word Sense Disambiguation (WSD) Methods: Supervised and Dictionary-Based Approaches.

TEXT CLASSIFICATION AND INFORMATION RETRIEVAL

1. Text Classification

📖 Definition:

Text Classification is the task of **assigning predefined categories/labels** to a given piece of text (document, sentence, or word).

- T**ext Classification** → "What type of text is this?"
- **Information Retrieval** → "Which documents are relevant to my query?"
- Example:
    - Email → Spam / Not Spam
    - News Articles → Politics / Sports / Technology

Approaches:

1. **Rule-based**
    - Uses manually crafted rules, keywords, and regular expressions.
2. **Machine Learning-based**
    - Represent text as features (Bag-of-Words, TF-IDF, Word Embeddings).
    - Train classifiers like Naïve Bayes, SVM, Decision Trees, Neural Networks.
3. **Deep Learning-based**
    - CNN, RNN, LSTM, Transformers (BERT, GPT).
    - Automatically learn hierarchical features from raw text.

Applications:

- Spam filtering (Gmail)
- Sentiment analysis (positive/negative reviews)
- Document categorization (legal, medical, academic)
- Chatbots & Customer Support

2. Information Retrieval (IR)

📖 Definition:

Information Retrieval is the process of **finding relevant documents** from a large collection in response to a user query.

- Example: Google search engine retrieving documents/webpages.

Components of IR System:

1. **Document Collection** (corpus)
2. **Indexing** (inverted index for fast retrieval)
3. **Query Processing** (user input text is tokenized, parsed, expanded)
4. **Retrieval Models** (ranking algorithms)
5. **Evaluation** (precision, recall, F1-score, MAP).

Popular IR Models:

- **Boolean Model** → exact matching using AND, OR, NOT.
- **Vector Space Model (VSM)** → uses cosine similarity with TF-IDF.
- **Probabilistic Models** → BM25, Language Models.
- **Neural IR** → BERT-based semantic search.

Applications:

- Web Search (Google, Bing)
- Recommender Systems (Netflix, Amazon)
- Question Answering (Quora, StackOverflow)
- Legal & Healthcare document retrieval

3. Relationship Between Text Classification & IR

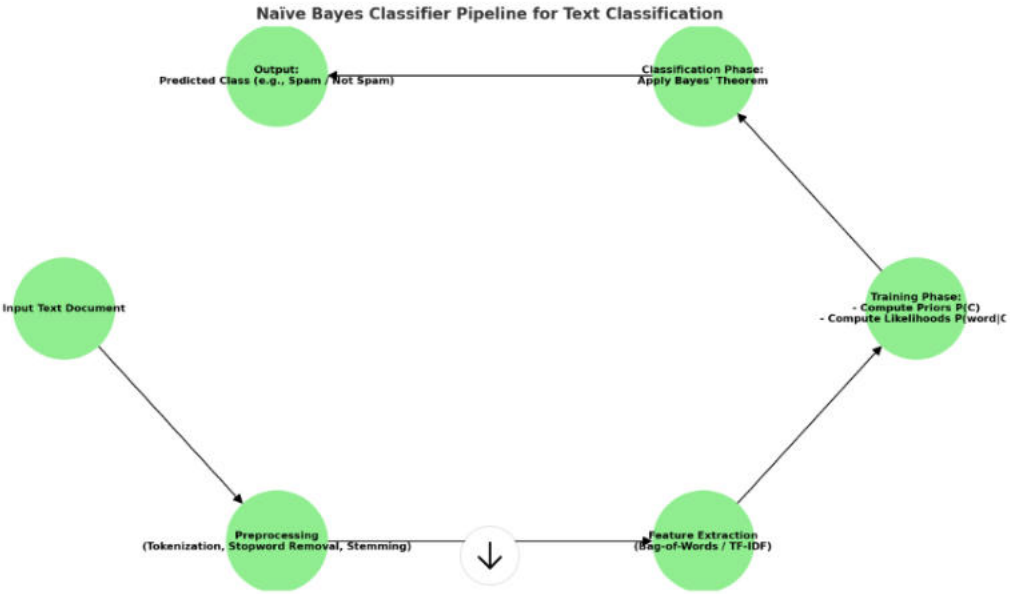
Text Classification	Information Retrieval
Assigns a label/category to text	Finds & ranks documents relevant to a query
Supervised learning task	Search & ranking problem
Example: Classify an email as spam	Example: Retrieve top 10 webpages about spam detection

NAÏVE BAYES CLASSIFIER FOR TEXT CLASSIFICATION

1. What is Naïve Bayes?

Naïve Bayes is a baseline classifier for text (spam filtering, sentiment analysis, topic categorization) — simple yet surprisingly effective in practice.

- A **probabilistic classifier** based on **Bayes' Theorem**.
- Assumes **conditional independence** between features (hence "naïve").
- Widely used for **text classification** because it works well with **high-dimensional data** (e.g., thousands of words).



2. Bayes' Theorem

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Where:

- $P(C|X)$ → Probability of class C given text X (posterior).
- $P(X|C)$ → Likelihood of seeing text X given class C.
- $P(C)$ → Prior probability of class C.
- $P(X)$ → Evidence (constant for all classes).

Classifier chooses the class with the **highest posterior probability**.

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{n} P(x_i|C)$$

3. Types of Naïve Bayes

1. **Multinomial Naïve Bayes** → best for word counts, Bag-of-Words, TF-IDF.
2. **Bernoulli Naïve Bayes** → binary features (word present/absent).
3. **Gaussian Naïve Bayes** → continuous features (rare in text).

4. Example: Spam Detection

Suppose we classify emails into **Spam (S)** or **Not Spam (N)**.

- Training data:
  - Spam emails often contain words like "free", "offer", "win".
  - Not spam emails often contain words like "meeting", "project", "report".

New email: `"free offer today"`

Steps:

1. Compute **priors**:

$$P(Spam),\ P(NotSpam)$$

2. Compute **likelihoods** for each word:
   - $P(free|Spam), P(offer|Spam), P(today|Spam)$
   - $P(free|NotSpam), P(offer|NotSpam), P(today|NotSpam)$
3. Apply Bayes rule:

$$P(Spam|email) \propto P(Spam) \times P(free|Spam) \times P(offer|Spam) \times P(today|Spam)$$

4. Compare with Not Spam probability → Choose higher one.

5. Advantages

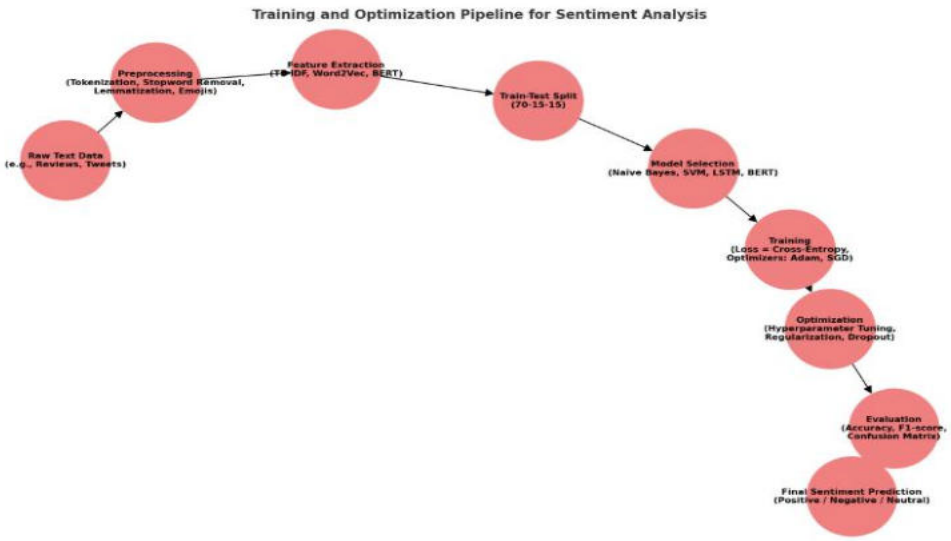
- Simple and fast.
- Works well for **large vocabulary, sparse data**.
- Requires less training data.

6. Limitations

- Assumes independence of words (not always true).
- Can perform poorly if features are highly correlated.
- Doesn't capture semantic meaning of words.

TRAINING AND OPTIMIZATION FOR SENTIMENT ANALYSIS

Training sentiment analysis involves preparing clean data, selecting an appropriate model, and optimizing it with proper hyperparameter tuning, feature engineering, and regularization to achieve strong generalization.



1. Training Pipeline

Step 1: Data Preparation

- Collect labeled dataset → e.g., tweets, reviews, or news headlines with *Positive, Negative, Neutral* tags.
- Preprocess:
  - Tokenization
  - Stopword removal
  - Lemmatization/Stemming
  - Handling emojis/emoticons (e.g., 😊 = positive)
  - Converting text → numerical features (Bag-of-Words, TF-IDF, Word2Vec, BERT embeddings).

Step 2: Model Selection

- **Classical ML Models:** Naïve Bayes, Logistic Regression, SVM.
- **Deep Learning Models:** CNNs, RNNs (LSTM, GRU), Transformers (BERT, RoBERTa).

Step 3: Training

- Split dataset → Train (70%), Validation (15%), Test (15%).
- Feed text features into the model.
- Model learns to map inputs → sentiment labels.
- Loss functions:

- **Cross-Entropy Loss** (for classification).
- Optimized using algorithms like **SGD, Adam, RMSprop**.

2. Optimization Techniques

A. Feature Engineering

- Use **TF-IDF** instead of raw word counts.
- Use **n-grams** (bigrams, trigrams) to capture context (e.g., "not good" vs "good").
- Pretrained embeddings (Word2Vec, GloVe, BERT) improve semantic understanding.

B. Hyperparameter Tuning

- Adjust **learning rate, batch size, regularization, dropout rate**.
- Use **Grid Search, Random Search, or Bayesian Optimization**.
- Early stopping to prevent overfitting.

C. Regularization

- **Dropout** in neural networks.
- **L2 regularization** (weight decay).
- Ensures model generalizes well on unseen data.

D. Data Augmentation

- Synonym replacement, back translation, paraphrasing.
- Helps when labeled data is small.

E. Class Imbalance Handling

- If dataset has more positives than negatives:
  - Oversample minority class (SMOTE).
  - Undersample majority class.
  - Class-weight adjustment in loss function.

3. Evaluation Metrics

- **Accuracy** (not enough if imbalanced).
- **Precision, Recall, F1-score** → better indicators.
- **Confusion Matrix** to analyze errors.

4. Example Workflow

1. Input: "This movie was amazing, I loved it!"
2. Preprocessing: tokens = [“movie”, “amazing”, “loved”]
3. Features: TF-IDF vector or BERT embedding.
4. Training: Model learns that “amazing”, “loved” → Positive class.
5. Optimization: Adjust learning rate + dropout until best validation F1-score.
6. Output: **Positive Sentiment** ✅

INFORMATION RETRIEVAL: BASIC CONCEPTS AND DESIGN FEATURES

1. Basic Concepts of Information Retrieval

Information Retrieval is the process of **storing, searching, and retrieving information** from large collections of unstructured or semi-structured data, such as documents, text, or multimedia. Unlike databases (which work with structured data), IR deals with **natural language text**.

🔹 **Key Concepts:**

- **Document** – A unit of information (article, webpage, email, report, etc.).
- **Collection/Corpus** – The set of documents to be searched.
- **Query** – The information need expressed by the user (often in keywords or natural language).
- **Indexing** – Creating efficient data structures (e.g., inverted index) to enable fast retrieval.
- **Matching/Retrieval** – Comparing queries with documents using a ranking model.
- **Relevance** – The degree to which a retrieved document satisfies the user’s information need.

👉 Example: Searching *"best machine learning books"* in Google.

- The query = your keywords
- Corpus = billions of webpages
- Retrieval = Google’s ranking algorithms returning relevant links

2. IR vs. Database Retrieval

- **Database Retrieval** → Structured data, exact matching (SQL queries).
- **Information Retrieval** → Unstructured/semi-structured data, approximate matching, ranking-based results.

3. Design Features of an IR System

When building an IR system (like Google Search, PubMed, or a digital library), key design features include:

(a) Document Representation

- Documents must be represented in a way that computers can process.
- Common model → **Vector Space Model** (each document is a vector of terms/weights, often using **TF-IDF**).

(b) Indexing

- **Inverted Index**: Maps each term → list of documents containing it.
- Optimized for **fast keyword-based search**.

(c) Query Processing

- Handling synonyms, stemming, lemmatization, stop-word removal.
- Support for **Boolean queries (AND, OR, NOT)** or **natural language queries**.

(d) Retrieval Models

- **Boolean Model** → Exact matches (AND/OR logic).
- **Vector Space Model** → Measures similarity (e.g., cosine similarity).
- **Probabilistic Models** → Estimate probability of relevance (e.g., BM25).
- **Neural IR Models** → Use embeddings and deep learning (e.g., BERT).

(e) Ranking & Relevance Feedback

- Results ranked by a **scoring function**.
- **Relevance feedback**: Users mark documents as relevant/non-relevant → system refines ranking.

(f) User Interface & Interaction

- Query box, autocomplete, suggestions, snippets, highlighting of keywords, etc.

(g) Performance Metrics

- **Precision** = fraction of retrieved docs that are relevant.
- **Recall** = fraction of relevant docs that were retrieved.
- **F1-score, MAP, nDCG** used in advanced systems.

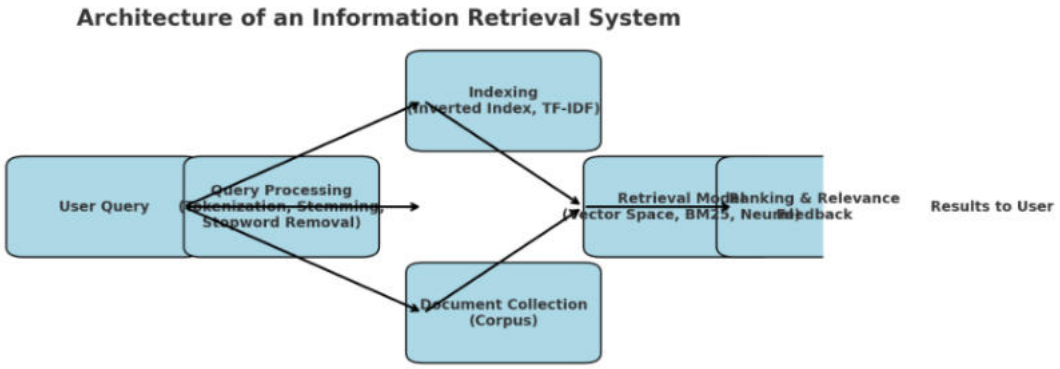


diagram of an Information Retrieval (IR) System Architecture showing the key design features:

- **User Query → Query Processing** (tokenization, stemming, stopword removal)
- **Indexing** (inverted index, TF-IDF, etc.)
- **Document Collection (Corpus)**
- **Retrieval Models** (vector space, BM25, neural models)
- **Ranking & Relevance Feedback**
- **Results back to User**

INFORMATION RETRIEVAL MODELS: CLASSICAL

Classical IR models are the earliest approaches to retrieving relevant documents from a collection based on a user's query. They rely on mathematical and statistical principles to model documents, queries, and their relationships.

1. Boolean Model

- **Concept**:
  Uses **set theory** and **Boolean logic** (AND, OR, NOT) to match queries with documents.
- **Representation**:
  - Documents and queries are represented as sets of terms.
  - Example: Query = *AI AND Healthcare* → retrieves only documents containing both terms.
- **Advantages**:
  - Simple and easy to implement.
  - Provides precise matching (all or nothing).
- **Disadvantages**:

- No ranking of results (all matching documents are equally relevant).
- Too rigid (slight mismatch in keywords leads to no results).

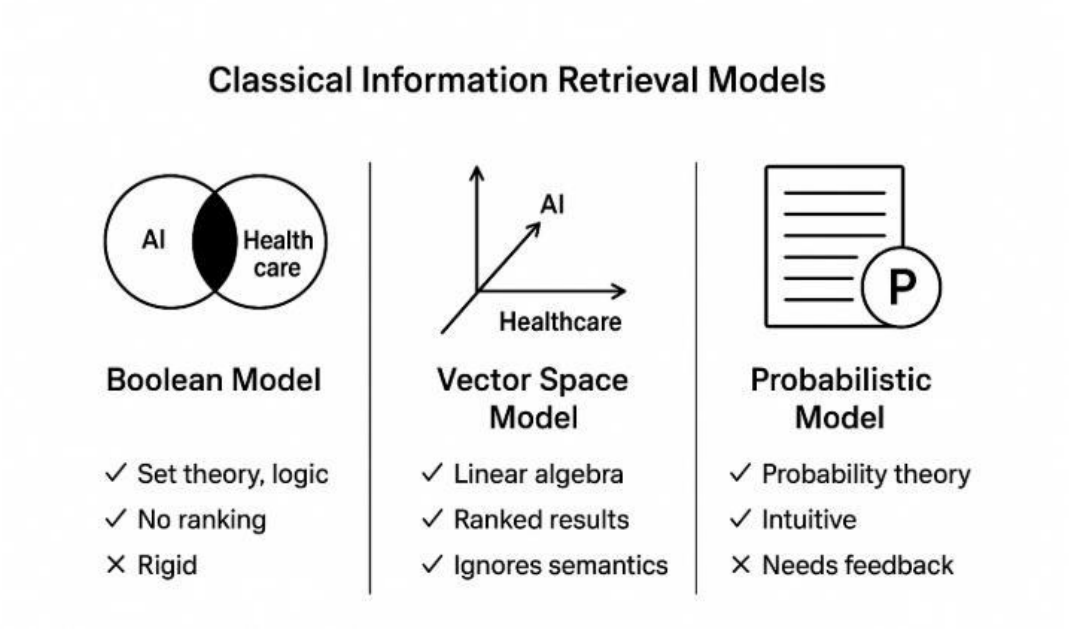
2. Vector Space Model (VSM)

- **Concept**:
  Represents documents and queries as vectors in a **multi-dimensional term space**.
- **Similarity**:
  - Uses **cosine similarity** between query and document vectors.
  - Weights terms using **TF-IDF** (Term Frequency–Inverse Document Frequency).
- **Advantages**:
  - Provides ranking of documents by relevance.
  - Can handle partial matches.
- **Disadvantages**:
  - Assumes term independence (ignores semantic relationships).
  - High-dimensional space is computationally expensive.

3. Probabilistic Model

- **Concept**:
  Estimates the probability that a document is relevant to a given query.
- **Principle**:
  Documents are ranked by their probability of relevance.
- **Example**:
  Binary Independence Model (BIM) assumes each term is either present or absent and independent.
- **Advantages**:
  - Intuitive probabilistic interpretation.
  - Forms the basis for modern ranking models (e.g., BM25).
- **Disadvantages**:
  - Assumes independence between terms.
  - Requires initial relevance feedback to refine probabilities.

Model	Basis	Ranking	Pros	Cons
Boolean	Set theory, logic	No	Simple, precise	No ranking, rigid
Vector Space Model	Linear algebra	Yes	Ranked results, partial match	Ignores semantics, high-dimensional
Probabilistic Model	Probability theory	Yes	Intuitive, modern basis	Needs assumptions, feedback



INFORMATION RETRIEVAL MODELS NON-CLASSICAL

Non-Classical Information Retrieval (IR) models were developed to overcome the limitations of classical models (Boolean, Vector Space, Probabilistic). They aim to handle semantics, uncertainty, context, and linguistic structure more effectively.

- **Fuzzy & Extended Boolean:** Handle vagueness and flexibility.
- **LSI & PLSA:** Capture latent semantics and topics.
- **Neural & Language Models:** Power modern search engines with context, probability, and deep learning.

Here are the main **Non-Classical IR Models**:

1. Fuzzy Set Model

- **Basis:** Fuzzy logic and set theory.

- **Key Idea:** Documents and queries are not treated as exact matches but as having degrees of relevance.
- **Features:**
    - Handles vagueness in user queries.
    - Provides graded (soft) matching instead of rigid matching.
    - Useful when query terms are imprecise.

2. Extended Boolean Model

- **Basis:** Combines Boolean logic with Vector Space concepts.
- **Key Idea:** Replaces strict AND/OR logic with *soft decision functions*.
- **Features:**
    - Allows partial matching between documents and queries.
    - Improves ranking over classical Boolean model.
    - More flexible in handling user queries.

3. Latent Semantic Indexing (LSI) Model

- **Basis:** Linear algebra (Singular Value Decomposition, SVD).
- **Key Idea:** Captures hidden semantic relationships between words and documents.
- **Features:**
    - Handles synonymy (different words with similar meaning).
    - Reduces dimensionality of term-document matrix.
    - Improves retrieval accuracy for concept-based queries.

4. Neural Network Model

- **Basis:** Machine learning using neural networks.
- **Key Idea:** Learns patterns of relevance between queries and documents.
- **Features:**
    - Models complex, non-linear relationships.
    - Can incorporate semantics and context.
    - Used in modern search engines with deep learning.

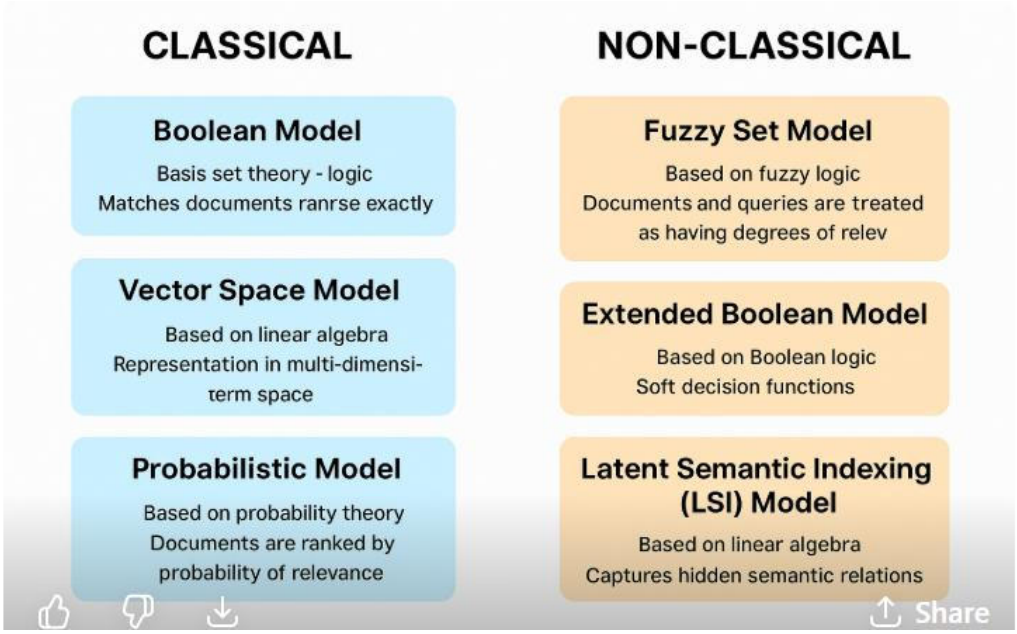
5. Probabilistic Latent Semantic Analysis (PLSA)

- **Basis:** Statistical modeling, probability distributions.

- **Key Idea:** Each document is represented as a mixture of latent topics.
- **Features:**
  - Extracts topics automatically.
  - Useful in text mining and clustering.
  - Predecessor of LDA (Latent Dirichlet Allocation).

6. Language Models for IR

- **Basis:** Statistical language modeling.
- **Key Idea:** Each document is treated as a probabilistic language model. The query likelihood is estimated for ranking.
- **Features:**
  - Very effective in modern IR.
  - Handles term dependencies better.
  - Forms the basis of many current search engines.



INFORMATION RETRIEVAL MODELS ALTERNATIVE MODELS

Here are some **Alternative Information Retrieval (IR) Models** beyond classical and non-classical approaches:

1. Inference Network Model

- Uses a probabilistic graphical model.

- Represents dependencies between queries, documents, and terms.
- Computes the probability of a document being relevant using Bayesian inference.

2. Neural IR Models

- Based on deep learning techniques.
- Uses embeddings (e.g., Word2Vec, BERT) to represent queries and documents.
- Can capture semantic meaning beyond keyword matching.
- Includes models like DSSM (Deep Structured Semantic Model), ColBERT, and Transformer-based retrieval.

3. Language Models for IR

- Treats each document as a probabilistic language model.
- Query likelihood model: ranks documents by the probability of generating the query from the document model.
- Popular example: LMIR with smoothing methods (Jelinek-Mercer, Dirichlet).

4. Cluster-Based Retrieval Models

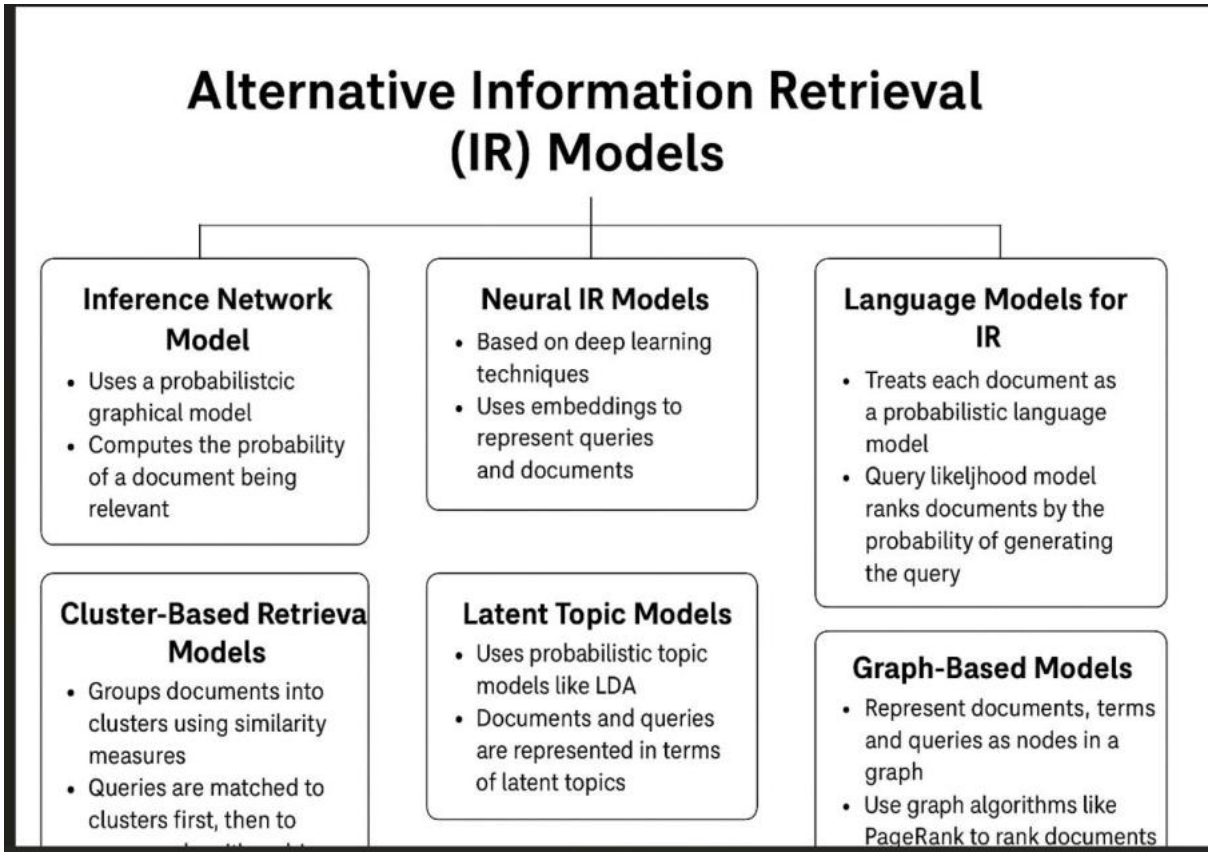
- Groups documents into clusters using similarity measures.
- Queries are matched to clusters first, then to documents within clusters.
- Improves efficiency and sometimes effectiveness.

5. Latent Topic Models

- Uses probabilistic topic models like Latent Dirichlet Allocation (LDA).
- Documents and queries are represented in terms of latent topics.
- Helps in semantic matching by bridging vocabulary gaps.

6. Graph-Based Models

- Represent documents, terms, and queries as nodes in a graph.
- Use graph algorithms like PageRank to rank documents based on structural relationships.
- Example: Hyperlink-Induced Topic Search (HITS), PageRank-based IR.



CLUSTER MODEL

The **Cluster Model** in Information Retrieval (IR) is a **non-classical retrieval model** that improves search efficiency and effectiveness by grouping similar documents together. Instead of matching a query against the entire collection, it matches the query against clusters of documents first.

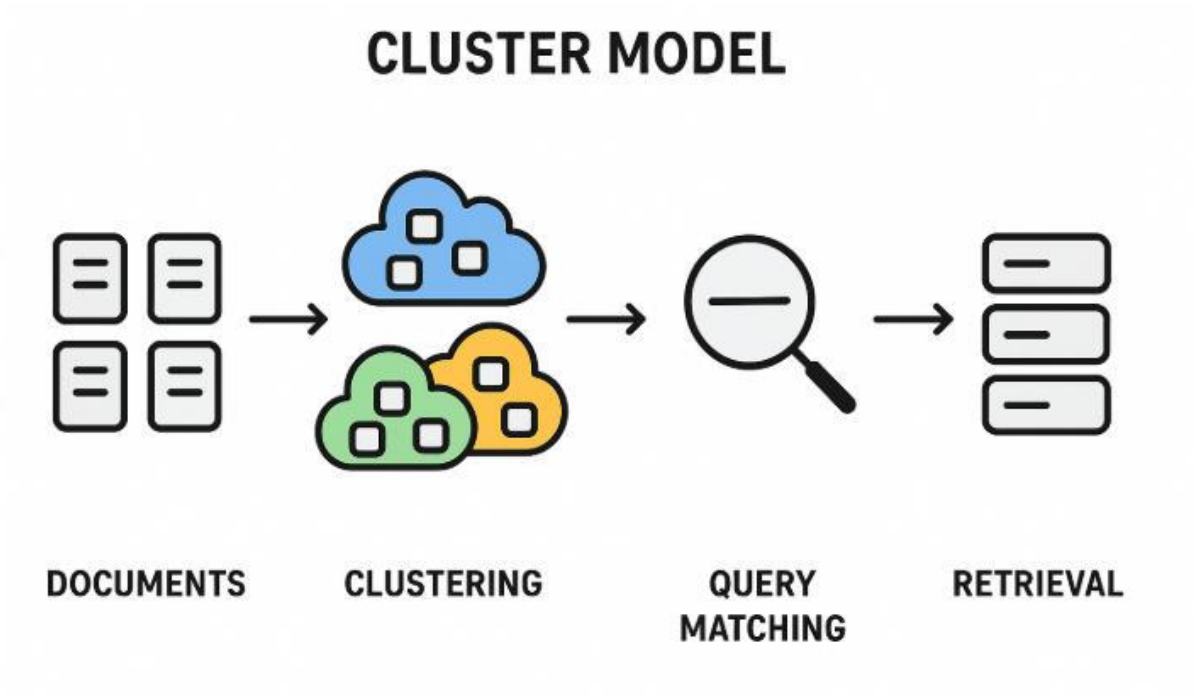
Key Concepts of Cluster Model:

1. **Clustering Documents**
   - Documents are grouped into clusters based on similarity (e.g., cosine similarity, Jaccard similarity).
   - Clustering methods include **k-means, hierarchical clustering, or density-based clustering**.
2. **Query Matching**
   - When a query is submitted, it is first compared to **cluster centroids** (representative summaries of clusters).
   - The most relevant cluster(s) are selected.
   - Finally, documents within those clusters are ranked and retrieved.

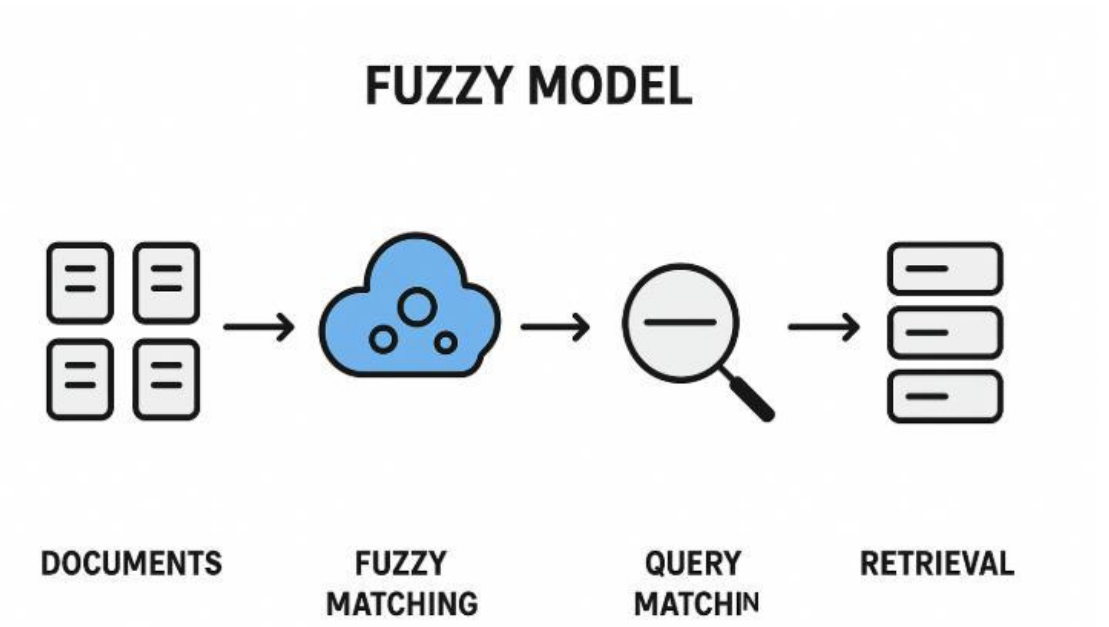
3. **Advantages**
    - Reduces search space → faster retrieval.
    - Groups similar documents → improves relevance.
    - Useful for exploratory search (e.g., browsing topics).
4. **Limitations**
    - Clustering is computationally expensive for large datasets.
    - Cluster quality strongly affects retrieval performance.
    - Not all queries benefit equally from clustering.

Example Use Case:

- A digital library groups research papers into clusters (e.g., AI, ML, NLP).
- A user searching "neural networks" will first be directed to the ML/NLP clusters, avoiding irrelevant clusters (like databases).



FUZZY MODEL



The **Fuzzy Model** in Information Retrieval (IR) is a **non-classical model** that uses **fuzzy set theory** to represent documents and queries. Unlike Boolean models, which use rigid "yes/no" matching, the Fuzzy Model allows **partial matching** with degrees of relevance.

Key Features of the Fuzzy Model:

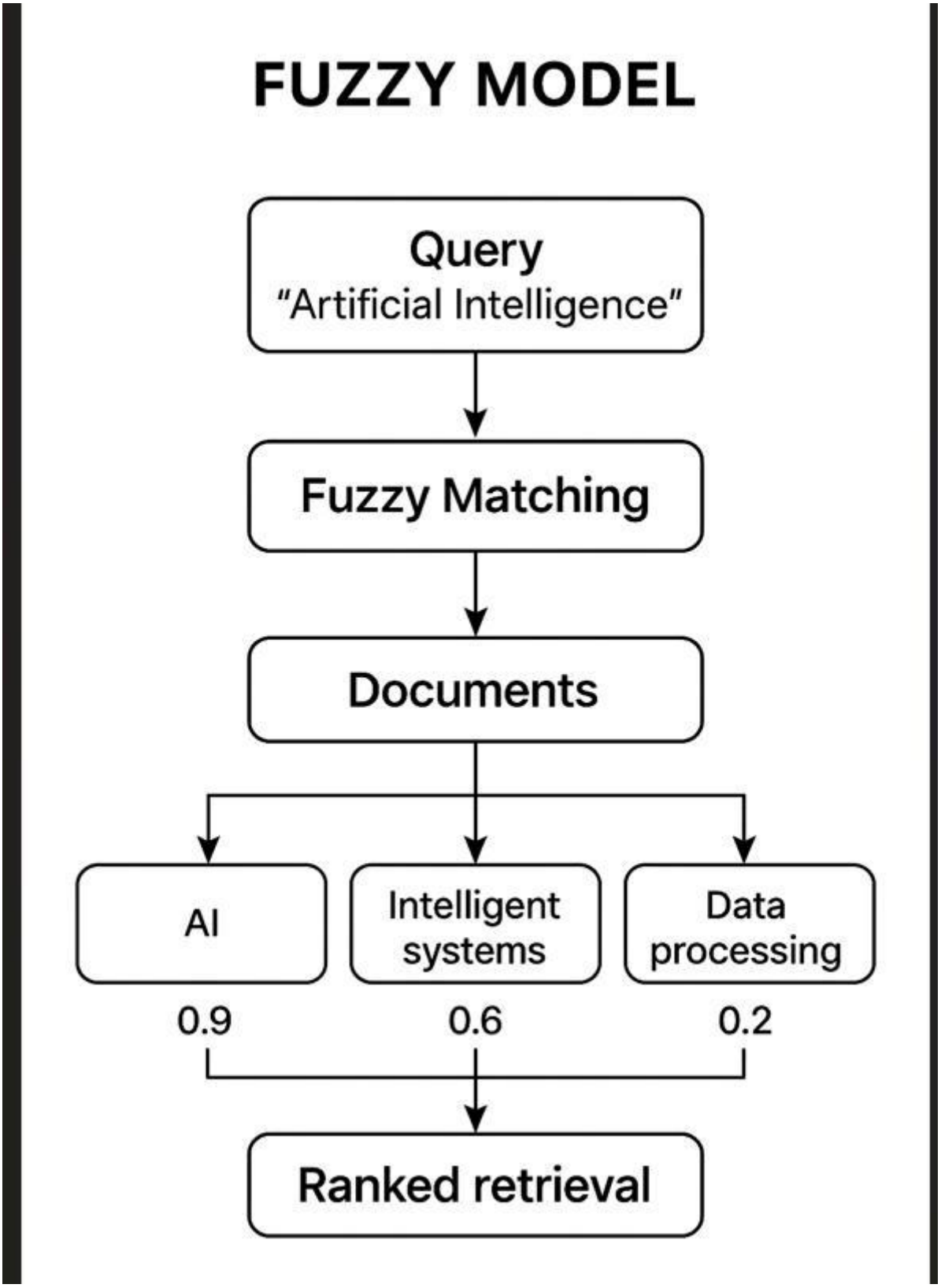
1. **Fuzzy Sets**
   - Documents and queries are represented as fuzzy sets of terms.
   - Each term has a **membership value** (between 0 and 1), indicating its importance in the document or query.
2. **Similarity Measurement**
   - Relevance between a document and a query is measured using **fuzzy similarity functions**.
   - Example:
     - In Boolean → "machine learning" must appear exactly.
     - In Fuzzy → If "learning algorithms" appears, it is still considered **partially relevant**.
3. **Advantages**
   - Captures uncertainty and vagueness in human language.
   - Provides ranking of results rather than strict true/false output.
   - Handles synonymy, polysemy, and partial matches better.

4. **Limitations**
    - More computationally complex than Boolean models.
    - Membership values and similarity functions must be well-defined, which can be challenging.

Example

- Query: *"Artificial Intelligence"*
- Document A: contains "AI" → high fuzzy match (0.9).
- Document B: contains "intelligent systems" → partial fuzzy match (0.6).
- Document C: contains "data processing" → low fuzzy match (0.2).

Thus, the fuzzy model retrieves A > B > C.



LSTM-BASED INFORMATION RETRIEVAL

The **LSTM-Based Information Retrieval model** applies **deep learning**—specifically **Long Short-Term Memory (LSTM) networks**—to improve how queries and documents are matched. Unlike classical IR models, LSTM-based models capture **sequential and contextual dependencies in text**, making them powerful for natural language understanding.

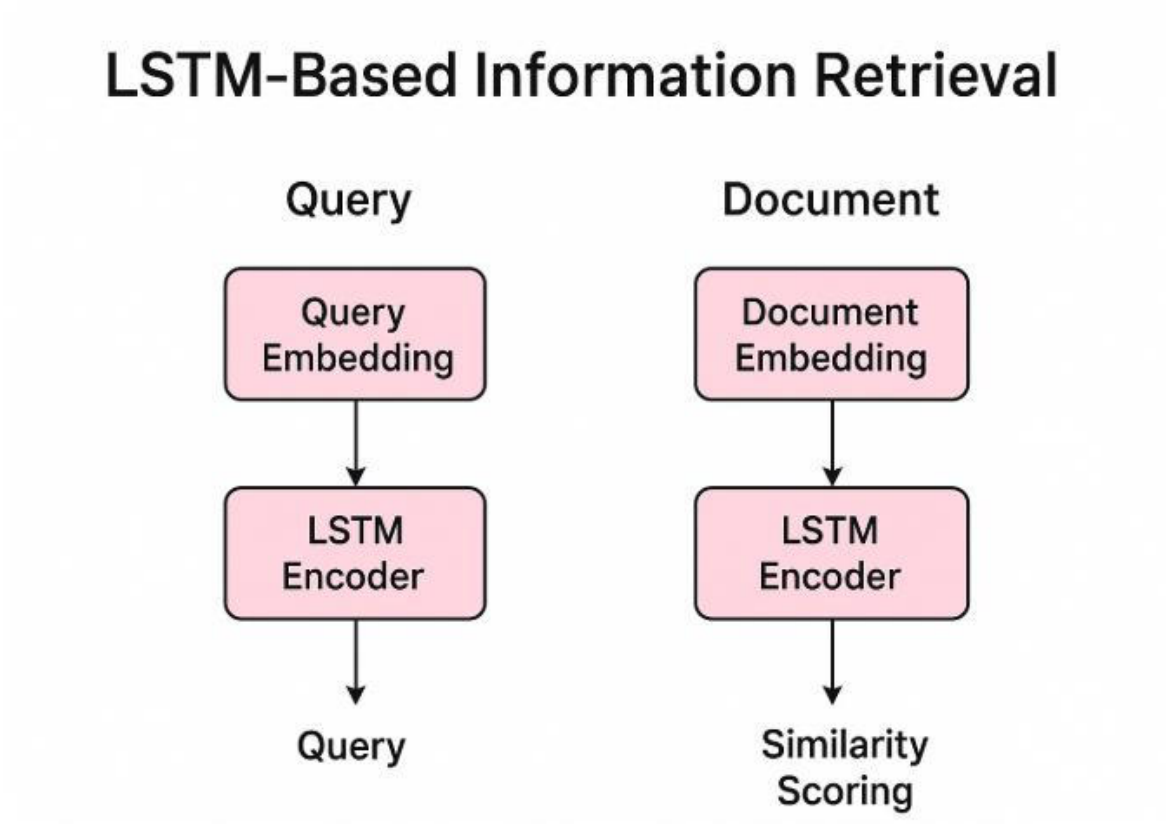
ey Concepts

1. **Why LSTM for IR?**
    - Traditional models (Boolean, Vector Space, BM25) treat queries and documents as bags of words, ignoring word order.
    - LSTMs handle **sequential data** and can capture **long-term dependencies**, which is critical in understanding natural language queries.
2. **Architecture**
    - **Input Layer**: Queries and documents are represented as word embeddings (e.g., Word2Vec, GloVe, BERT embeddings).
    - **LSTM Encoder**: Processes the sequence of embeddings and captures contextual meaning.
    - **Representation Layer**: Produces dense vector representations of queries and documents.
    - **Similarity Scoring**: Cosine similarity, dot product, or a learned function compares query and document representations.
    - **Ranking Layer**: Ranks documents based on similarity scores.
3. **Training**
    - LSTM models are trained on large-scale query-document pairs.
    - Objective: maximize the score of relevant documents and minimize that of irrelevant ones.
    - Loss functions: **contrastive loss, hinge loss, or cross-entropy loss**.
4. **Advantages**
    - Captures word order and semantic dependencies.
    - Learns query-document relationships automatically.
    - Handles synonyms, paraphrases, and context much better than classical models.
5. **Limitations**
    - Requires **large labeled datasets**.

- Training is **computationally expensive**.
- LSTMs are being replaced in practice by **Transformers (BERT, GPT, etc.)** which perform even better.

Example Flow

- Query: *"best deep learning books"*
- Document A: "Top resources for deep learning study" → LSTM captures context → High relevance score.
- Document B: "Library contains many programming books" → Lower relevance score.



WORD SENSE DISAMBIGUATION (WSD)

Word Sense Disambiguation (WSD) is the process of **determining the correct meaning (sense) of a word** in a given context when the word has multiple possible interpretations.
Example:

- *"He went to the **bank** to deposit money."* → **bank = financial institution**
- *"He sat on the **bank** of the river."* → **bank = river shore**

Approaches to WSD

1. **Knowledge-based Methods**

- Use lexical resources like **WordNet**, dictionaries, and thesauri.
- Algorithms:
  - *Lesk Algorithm*: Chooses the sense with the most word overlap between dictionary definitions and context.
  - *Semantic Similarity*: Selects the sense closest in meaning to surrounding words.

2. **Supervised Learning Methods**
   - Treat WSD as a classification task.
   - Steps:

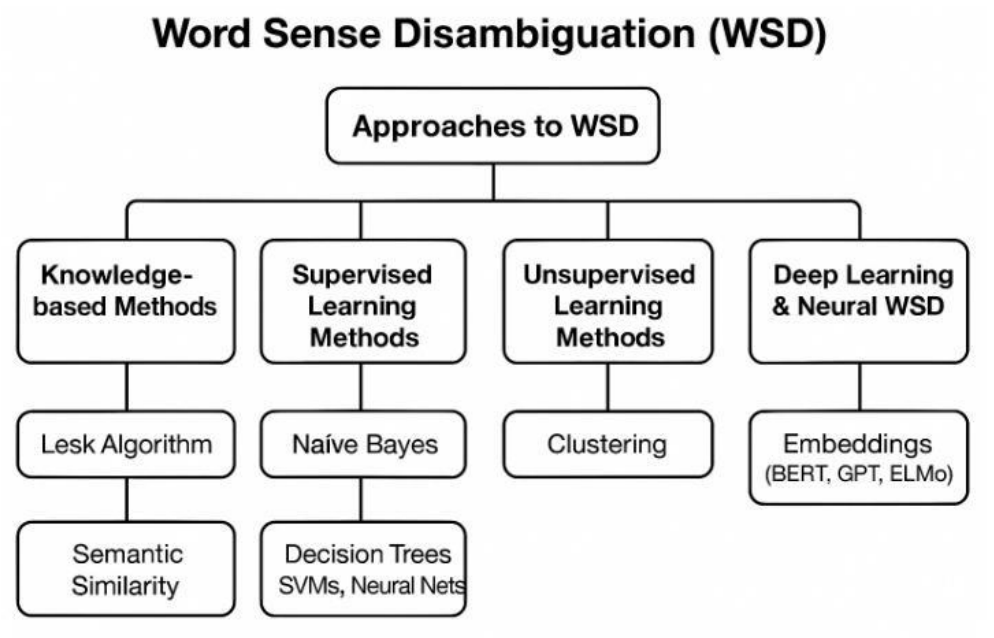
1. Extract features from the word's context (neighboring words, POS tags, etc.).
2. Train classifiers like Naïve Bayes, Decision Trees, SVMs, Neural Networks.
   - Limitation: Requires **large annotated datasets**.

3. **Unsupervised Learning Methods**
   - Cluster word occurrences into groups based on context similarity.
   - No labeled data needed.
   - Example: Using embeddings + clustering to group word senses.

4. **Deep Learning & Neural WSD**
   - Use embeddings (Word2Vec, GloVe, BERT).
   - Contextual embeddings (BERT, GPT, ELMo) **disambiguate senses dynamically** depending on context.
   - Example: *BERT automatically distinguishes "bank" in finance vs. river contexts.*

Applications of WSD

- **Machine Translation** (choosing correct word in target language).
- **Information Retrieval** (retrieving documents with correct sense).
- **Text Mining & NLP** (better semantic understanding).
- **Chatbots/Assistants** (improving intent understanding).



WORD SENSE DISAMBIGUATION (WSD) METHODS: SUPERVISED APPROACHES

Supervised WSD treats word sense disambiguation as a **classification problem**, where each occurrence of an ambiguous word is assigned to one of its possible senses.

Steps in Supervised WSD

1. **Data Preparation**
    - Requires a **sense-annotated corpus** (e.g., SemCor, OntoNotes).
    - Each ambiguous word in training data is labeled with the correct sense.
2. **Feature Extraction**
    - **Local Features**: Neighboring words, collocations.
    - **Global Features**: Sentence/paragraph-level words.
    - **Syntactic Features**: POS tags, dependency relations.
    - **Semantic Features**: Word embeddings (Word2Vec, GloVe, BERT).
3. **Model Training**
    - Train a classifier to predict the sense using extracted features.
    - Popular classifiers:
        - **Naïve Bayes**
        - **Decision Trees**

- **Support Vector Machines (SVMs)**
- **k-Nearest Neighbors (kNN)**
- **Neural Networks** (MLPs, CNNs, RNNs)

4. **Prediction**
    - Apply the trained model to unseen sentences to predict the correct sense of ambiguous words.

Example

Sentence: *"The fisherman sat on the **bank**."*

- Context features: {fisherman, sat, river}
- Classifier predicts: **bank = river shore**

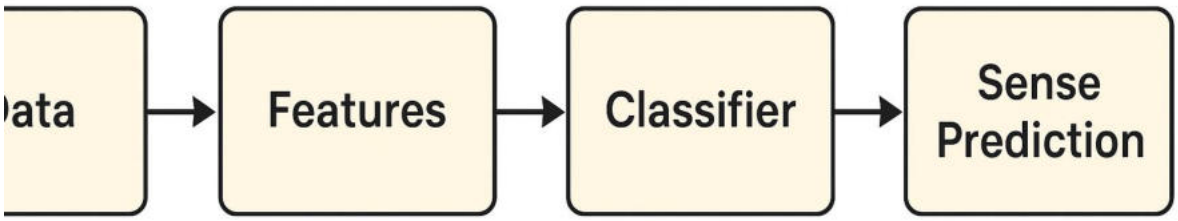
✅ **Advantages**

- High accuracy when **large annotated datasets** are available.
- Can adapt to **domain-specific senses**.

❌ **Limitations**

- **Expensive** to build sense-annotated corpora.
- Performance drops with **low-resource languages**.
- Classifiers may **overfit** on small datasets.

SUPERVISED WSD



WORD SENSE DISAMBIGUATION (WSD) – DICTIONARY-BASED APPROACHES

Dictionary-based (or **knowledge-based**) approaches exploit **lexical resources** such as **WordNet, Oxford Dictionary, or machine-readable dictionaries** to determine the correct sense of a word in context.
They do **not require annotated corpora** → useful for low-resource languages.

Major Dictionary-Based WSD Methods

1. Lesk Algorithm (1986)

- **Idea:** The correct sense of a word has the **highest overlap** between its dictionary definition (gloss) and the context words.
- **Steps:**
    1. Take all possible senses of the target word from the dictionary.
    2. For each sense, compare its gloss with the glosses of context words.
    3. The sense with maximum word overlap is chosen.
- **Example:**
  *"I went to the **bank** to deposit money."*
    - Gloss overlap with *money, deposit* → sense = *financial institution*.

2. Extended Lesk Algorithm

- Extends Lesk by considering **glosses of related words** (synonyms, hypernyms, hyponyms).
- Provides richer context and improves accuracy.

3. Semantic Similarity / Relatedness Methods

- Uses WordNet to compute **semantic distance** between senses.
- Example measures:
  - Path-based (shortest path in WordNet hierarchy).
  - Information content–based (Resnik, Lin, Jiang-Conrath measures).
- Chooses the sense most semantically similar to context words.

4. Selectional Preferences

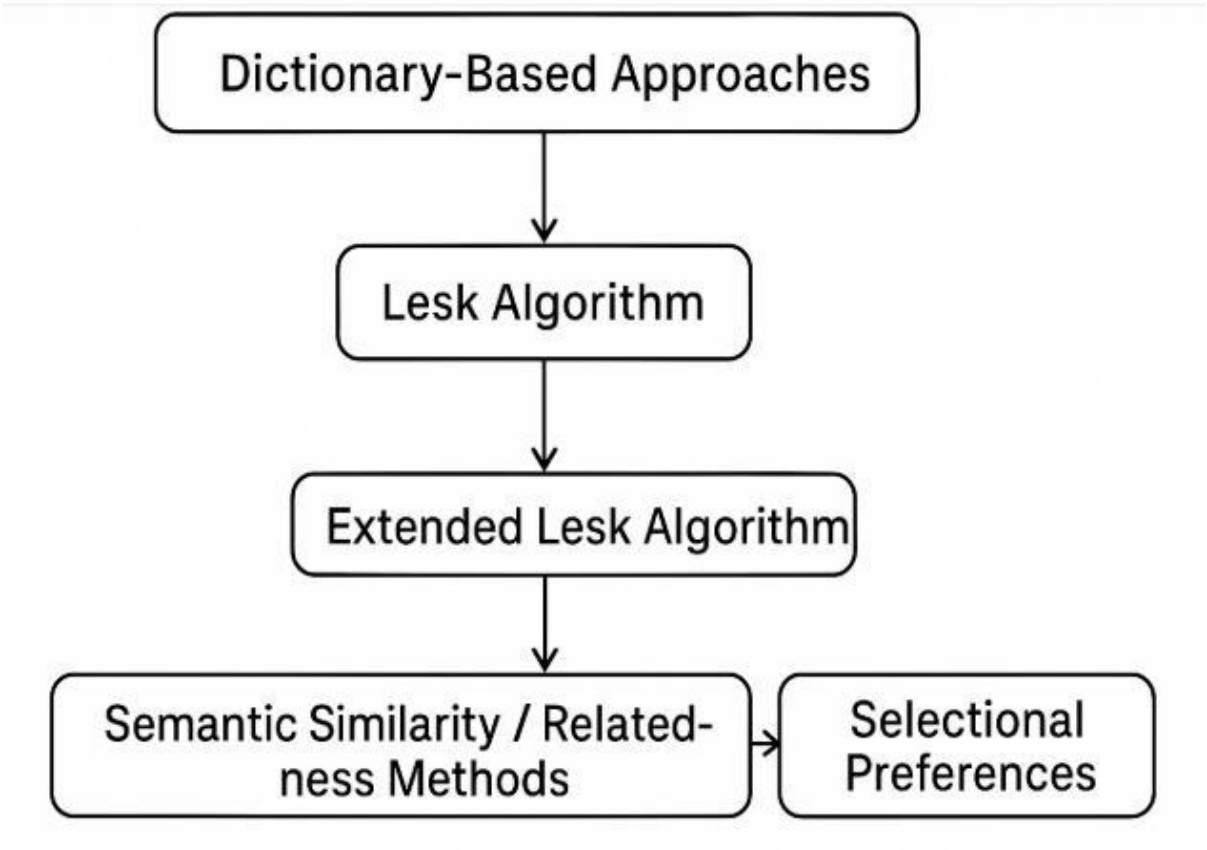
- Uses dictionary-based semantic restrictions.
- Example: *"He drank a **glass** of water."*
  - Verb *drink* expects a **liquid object** → sense of *glass* = container, not material.

✅ **Advantages**

- No need for large labeled corpora.
- Effective for languages with **rich dictionaries** (e.g., English with WordNet).
- Easy to implement.

❌ **Limitations**

- **Dictionary coverage problem** (not all senses well-defined).
- Struggles with **rare words or domain-specific terminology**.
- Performance often lower than supervised ML methods.


Dictionary-Based Approaches
Lesk Algorithm
Extended Lesk Algorithm
Semantic Similarity / Related-ness Methods
Selectional Preferences

UNIT IV: Machine Translation and Semantic Processing

Introduction to Machine Translation (MT), Language Divergence and Typology in MT Encoder-Decoder Model for Machine Translation, Translating in Low-Resource Scenarios, MT Evaluation Metrics and Techniques, Bias and Ethical Issues in NLP and Machine Translation, Semantic Analysis and First-Order Logic in NLP, Thematic Roles and Selectional Restrictions in Semantics, Word Senses and Relations Between Senses

MACHINE TRANSLATION AND SEMANTIC PROCESSING

- **Machine Translation** = Automatic cross-lingual text conversion.
- **Semantic Processing** = Understanding & representing meaning in language.
- In NLP, **semantic understanding is critical to high-quality MT**, along with other applications like QA, IR, and chatbots.

Machine Translation in NLP

Machine Translation (MT) is the task of **automatically converting text or speech from one natural language (source) into another (target)** using computational techniques.

Approaches to MT

1. **Rule-Based Machine Translation (RBMT)**
    - Relies on grammar rules, morphology, and bilingual dictionaries.
    - Works well for morphologically complex languages.
    - Example: Early systems like SYSTRAN.
2. **Statistical Machine Translation (SMT)**
    - Uses probabilities derived from bilingual corpora.
    - Example: Phrase-based models.
    - Limitation: Struggles with long-range dependencies and fluency.
3. **Neural Machine Translation (NMT)**
    - Employs deep learning (RNNs, LSTMs, Transformers).
    - Captures context and semantics effectively.

- Current systems: Google Translate, DeepL.

4. **Hybrid MT**
    - Combines rule-based, statistical, and neural methods for better accuracy.

Challenges in MT

- **Ambiguity:** Words with multiple meanings.
- **Idioms & Metaphors:** Hard to translate literally.
- **Word Order Differences:** Languages differ syntactically (e.g., English SVO vs Hindi SOV).
- **Low-Resource Languages:** Lack of large corpora.
- **Domain Adaptation:** A model trained on news text may fail in medical/technical domains.

Semantic Processing in NLP

Semantic Processing refers to techniques for **analyzing, representing, and understanding the meaning** of natural language.

Core Areas of Semantic Processing

1. **Lexical Semantics**
    - Word meaning and relations (synonymy, antonymy, hyponymy, polysemy).
    - Example: *big* ≈ *large*, *rose* → hyponym of *flower*.
2. **Word Sense Disambiguation (WSD)**
    - Selecting the correct sense of a word based on context.
    - Example: *bank* → riverbank vs financial institution.
3. **Semantic Role Labeling (SRL)**
    - Identifies semantic roles in a sentence.
    - Example: *Mary (agent) gave (action) John (recipient) a book (theme).*
4. **Compositional Semantics**
    - Meaning of a sentence derived from the meanings of words and their structure.

5. **Distributional Semantics**
    - Word meanings derived from usage patterns.
    - Implemented via **embeddings** (Word2Vec, GloVe, BERT).
6. **Semantic Networks & Ontologies**
    - Structured representations of concepts and relationships.
    - Examples: WordNet, ConceptNet, DBpedia.

Applications of Semantic Processing

- Improves **Machine Translation** by resolving ambiguities.
- **Question Answering Systems** (understanding intent).
- **Information Retrieval** (context-based search).
- **Chatbots & Dialogue Systems** (semantic understanding).
- **Text Summarization & Categorization.**

MT & Semantic Processing Relationship in NLP

- MT depends heavily on semantic processing.
- Example: Translating *"He went to the bank"* → needs **WSD** to decide if *bank* means financial institution or riverbank.
- Advanced **NMT systems (e.g., Transformer-based models)** leverage **semantic embeddings** (BERT, GPT) for accurate translation.

INTRODUCTION TO MACHINE TRANSLATION (MT)

Machine Translation (MT) is a core task in **Natural Language Processing (NLP)** that focuses on the **automatic translation of text or speech from one natural language (source language) into another (target language)** using computational methods.

Machine Translation is about teaching computers to "understand" and "translate" human languages automatically, with accuracy and fluency.

UNIT IV

The main goal of MT is to make information accessible across languages by removing the **language barrier**. Unlike human translators, MT systems rely on **linguistic rules, statistical models, or neural networks** to generate translations.

Key Features of MT

- Converts source language into target language without human intervention.
- Preserves **meaning** and **context**, not just words.
- Requires understanding of **syntax, semantics, morphology, and pragmatics** of languages.

Evolution of MT

1. **Rule-Based MT (RBMT):** Early systems using grammar rules and dictionaries.
2. **Statistical MT (SMT):** Based on probability models trained on bilingual corpora.
3. **Neural MT (NMT):** Uses deep learning models (e.g., sequence-to-sequence, Transformers) to capture context and semantics.

Applications of MT

- Online translators (Google Translate, DeepL).
- Multilingual communication in business and education.
- Cross-language information retrieval.
- Real-time speech translation (e.g., Skype, Zoom interpreters).

Challenges in MT

- **Ambiguity:** Words with multiple meanings.
- **Idiomatic Expressions:** Hard to translate literally.
- **Word Order Differences:** Languages vary syntactically.
- **Low-Resource Languages:** Lack of sufficient parallel corpora.

LANGUAGE DIVERGENCE AND TYPOLOGY IN MT ENCODER-DECODER MODEL FOR MACHINE TRANSLATION

- **Language Divergence** = structural/semantic differences across languages.

- **Typology** = systematic classification of these differences.
- The **Encoder–Decoder model (with attention/transformers)** in MT is powerful because it can **encode source meaning independent of order/morphology** and **decode appropriately** according to the target language's typological structure.

1. Language Divergence in MT

- **Definition:** Language divergence refers to the **differences across languages** in terms of grammar, morphology, word order, and semantic representation.
- **Examples of Divergence:**
  - **Word Order:** English (SVO: *She eats apples*) vs Hindi (SOV: *Woh seb khati hai*).
  - **Morphology:** English has limited inflections (*walk, walks*), while Turkish has rich morphology (*evlerinizden* = "from your houses").
  - **Lexical Gaps:** Some words/phrases in one language have no direct equivalent in another (e.g., German *Schadenfreude*).

👉 In MT, these divergences create **challenges for direct word-to-word translation**, requiring models to capture deeper syntactic and semantic structures.

2. Typology in MT

- **Definition:** Linguistic typology classifies languages based on **structural features** like word order, morphology, and syntax.
- **Importance for MT:**
  - Helps design better **pre-processing** and **alignment strategies**.
  - Guides model architecture when dealing with **morphologically rich or free-word-order languages**.
  - Provides insights for **transfer learning** across related languages.

3. Encoder–Decoder Model in MT

The **Encoder–Decoder framework** (foundation for Neural MT) is designed to handle these divergences.

- **Encoder:** Processes the **source language sentence** into a fixed-length or contextual representation (vector).

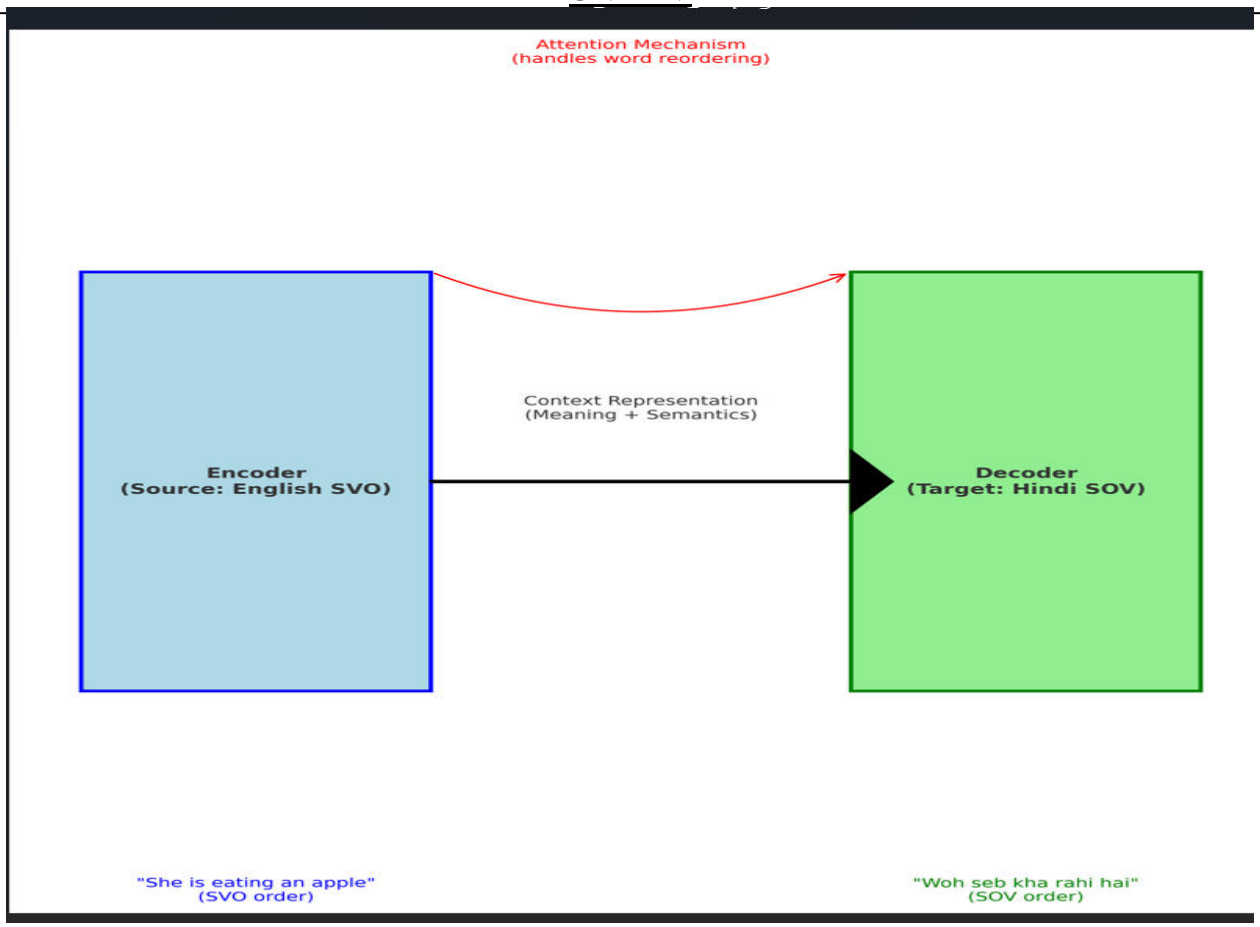
- **Decoder:** Generates the **target language sentence** step by step, conditioned on the encoded representation.
- **Attention Mechanism:** Helps the decoder focus on relevant parts of the source sentence during translation, especially important for long or structurally divergent sentences.

4. Role of Encoder–Decoder in Handling Divergence & Typology

- **Word Order Divergence:**
  - The encoder captures entire sentence context.
  - The decoder reorders words according to the target language syntax (e.g., English → Hindi).
- **Morphological Divergence:**
  - Encoders with **subword/byte-pair encoding (BPE)** handle complex word forms.
  - Useful for agglutinative languages like Finnish, Tamil, Turkish.
- **Semantic Divergence:**
  - Contextual embeddings (e.g., in Transformer-based models like BERT, GPT, mBART) ensure **meaning preservation**.
- **Typological Awareness:**
  - Multilingual NMT models benefit from typology by **sharing representations** across related languages (e.g., Romance languages: Spanish, Italian, French).
  - For highly divergent pairs (English–Chinese), the encoder–decoder needs stronger attention and larger training corpora.

5. Example

- English sentence: *She is eating an apple.* (SVO)
- Hindi translation: *Woh seb kha rahi hai.* (SOV)
- The **encoder** captures meaning in English order.
- The **decoder** rearranges words into Hindi order, respecting its typology.



TRANSLATING IN LOW-RESOURCE SCENARIOS

Translating in low-resource scenarios requires **creativity in data generation (back-translation, augmentation)** and **leveraging multilingual/transfer learning models** to compensate for the lack of parallel corpora.

1. What are Low-Resource Scenarios?

- A **low-resource language** is one that **lacks sufficient parallel corpora, linguistic resources, or digital text data** for training robust Machine Translation (MT) systems.
- Examples: Many African (Yoruba, Amharic), Indian (Kannada, Manipuri), and indigenous languages.
- **Problem:** Most MT approaches (especially Neural MT) require **millions of sentence pairs**, but low-resource languages often have only a few thousand.

2. Challenges in Low-Resource MT

- **Data Scarcity:** Very few parallel corpora for training.
- **Domain Mismatch:** Available data may not represent real-world usage.
- **Morphological Richness:** Low-resource languages are often highly inflected.
- **Divergence:** Structural and typological differences from high-resource languages.
- **Bias & Quality Issues:** Crowdsourced or automatically generated data may contain errors.

3. Approaches to Handle Low-Resource Translation

(a) Data Augmentation

- **Back-Translation:** Translate monolingual target text back into the source language to create synthetic parallel data.
- **Forward Translation:** Translate source monolingual data into target language using weak models.
- **Noise Injection & Paraphrasing:** Generate variations to expand limited data.

(b) Transfer Learning

- **Multilingual NMT (mNMT):** Train one model on multiple related languages so low-resource languages benefit from high-resource ones.
- **Zero-Shot Translation:** Use multilingual models to translate between two languages without direct training pairs (e.g., Hindi ↔ Tamil via English).

(c) Unsupervised MT

- Uses only **monolingual corpora** in both languages with techniques like:
    - Shared word embeddings (cross-lingual representations).
    - Iterative back-translation.

(d) Pretrained Language Models

- Use **large multilingual models** (e.g., mBART, XLM-R, mT5).
- Fine-tune them on small parallel datasets for the target low-resource language.

(e) Pivot-Based Translation

- Translate low-resource → high-resource → target.
- Example: Telugu → English → French.

(f) Community & Crowdsourcing

- Collecting translations from native speakers (e.g., via Wikipedia, local projects).

4. Real-World Examples

- **Google Translate** supports many low-resource languages using multilingual NMT.
- **Masakhane Project (Africa):** Community-driven NMT for African languages.
- **IndicNLP Project:** Building resources for Indian low-resource languages.

5. Summary

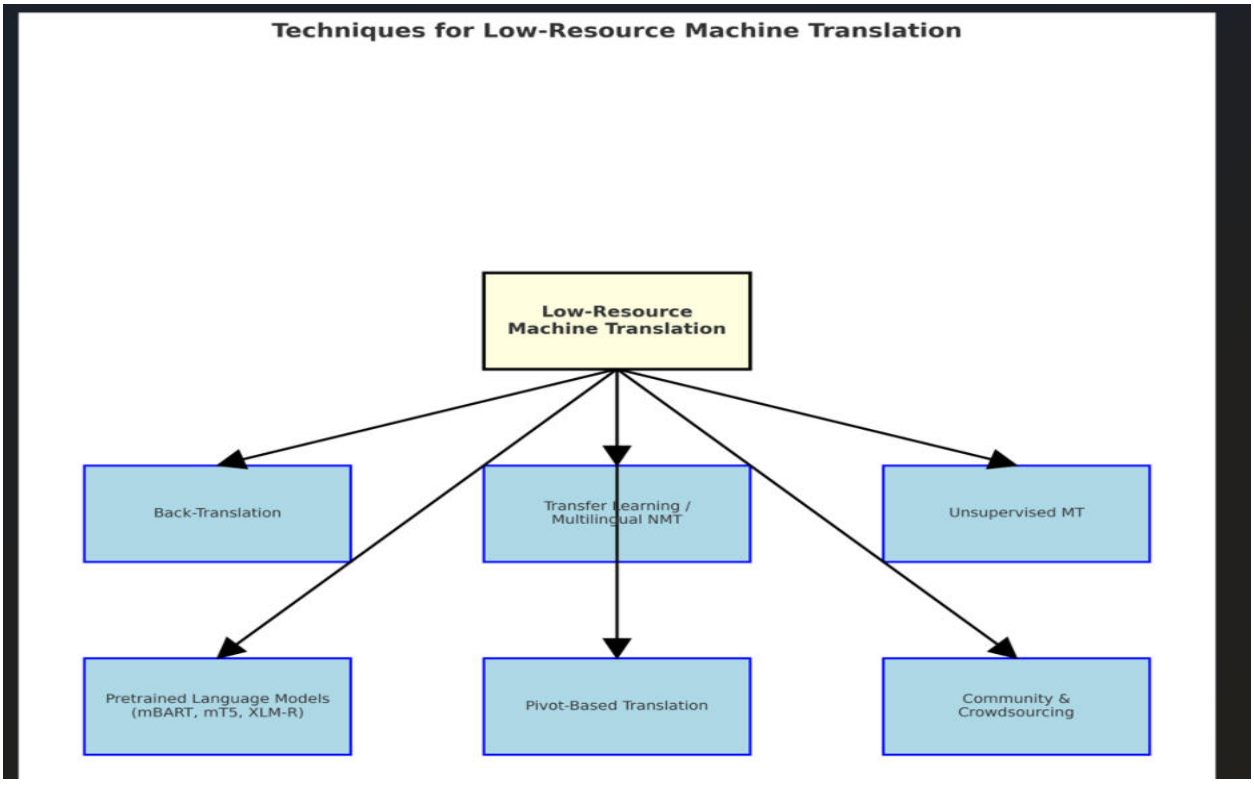
- **Low-resource translation** is a major challenge in NLP.
- Traditional NMT fails due to data scarcity.
- Modern strategies (transfer learning, back-translation, multilingual pretraining, pivoting) help overcome these limitations.
- Success depends not only on algorithms but also on **building more linguistic resources**.

High-Resource vs Low-Resource Machine Translation

- High-resource → straightforward supervised training.
- Low-resource → needs **creative approaches** like augmentation, transfer, and multilingual pretraining.

Aspect	High-Resource Languages	Low-Resource Languages
Data Availability	Large parallel corpora (millions of sentence pairs)	Very limited parallel corpora (few thousand or none)
Typical Approaches	Supervised Neural MT (Transformer, seq2seq) trained directly on parallel data	Transfer learning, multilingual NMT, pivoting, back-translation, unsupervised MT

Aspect	High-Resource Languages	Low-Resource Languages
Model Performance	High accuracy, fluent translations	Lower accuracy, may struggle with grammar and meaning preservation
Training Requirements	Requires massive compute and large-scale datasets	Requires data augmentation, synthetic data generation, and fine-tuning
Examples	English ↔ French, English ↔ German, Chinese ↔ English	Yoruba ↔ English, Telugu ↔ French, Amharic ↔ Swahili
Solutions	Direct end-to-end training with attention/transformer models	Back-translation, multilingual pretraining (mBART, mT5), crowdsourcing resources
Challenges	Handling domain adaptation, long sentences	Data scarcity, morphology, typological divergence, noisy/low-quality resources



UNIT IV

MT EVALUATION METRICS AND TECHNIQUES

1. Introduction

Machine Translation (MT) evaluation is the process of **assessing the quality of translated output** produced by MT systems.
It ensures translations are **accurate, fluent, and meaningful**, similar to human translations.

Two broad categories:

1. **Human Evaluation**
2. **Automatic Evaluation**

2. Human Evaluation Techniques

Human judges evaluate translation quality based on linguistic and semantic criteria.

- **Adequacy:** How well the translation preserves the meaning of the source text.
- **Fluency:** Grammatical correctness and naturalness of the target text.
- **Comprehensibility:** Ease with which a reader understands the translation.
- **Ranking/Pairwise Comparison:** Compare outputs of different systems and rank them.

✅ *Pros:* Captures nuance, idiomatic correctness.
❌ *Cons:* Time-consuming, costly, subjective.

3. Automatic Evaluation Metrics

Automatic metrics compare MT output against **reference translations** or evaluate quality without references.

(a) Reference-Based Metrics

1. **BLEU (Bilingual Evaluation Understudy):**
    - Compares n-grams (word sequences) between system output and reference translation.
    - Score: 0 (worst) → 1 (best).
    - Widely used, but doesn't capture meaning fully.
2. **NIST:**
    - Extension of BLEU; gives higher weight to *informative* n-grams.

3. **METEOR:**
    - Considers synonyms, stemming, and paraphrases.
    - More correlated with human judgment than BLEU.
4. **TER (Translation Edit Rate):**
    - Measures number of edits (insertions, deletions, substitutions, shifts) needed to transform MT output into reference translation.

(b) Semantic & Embedding-Based Metrics

1. **BERTScore:**
    - Uses contextual embeddings (BERT) to compare semantic similarity.
    - Better at capturing meaning than BLEU.
2. **MoverScore, COMET, BLEURT:**
    - Neural-based evaluation with strong correlation to human judgment.

(c) Reference-Free (Quality Estimation, QE)

- Predicts translation quality **without reference translations**.
- Uses machine learning to model adequacy/fluency directly.
- Useful for low-resource languages where references are scarce.

4. Comparative Summary

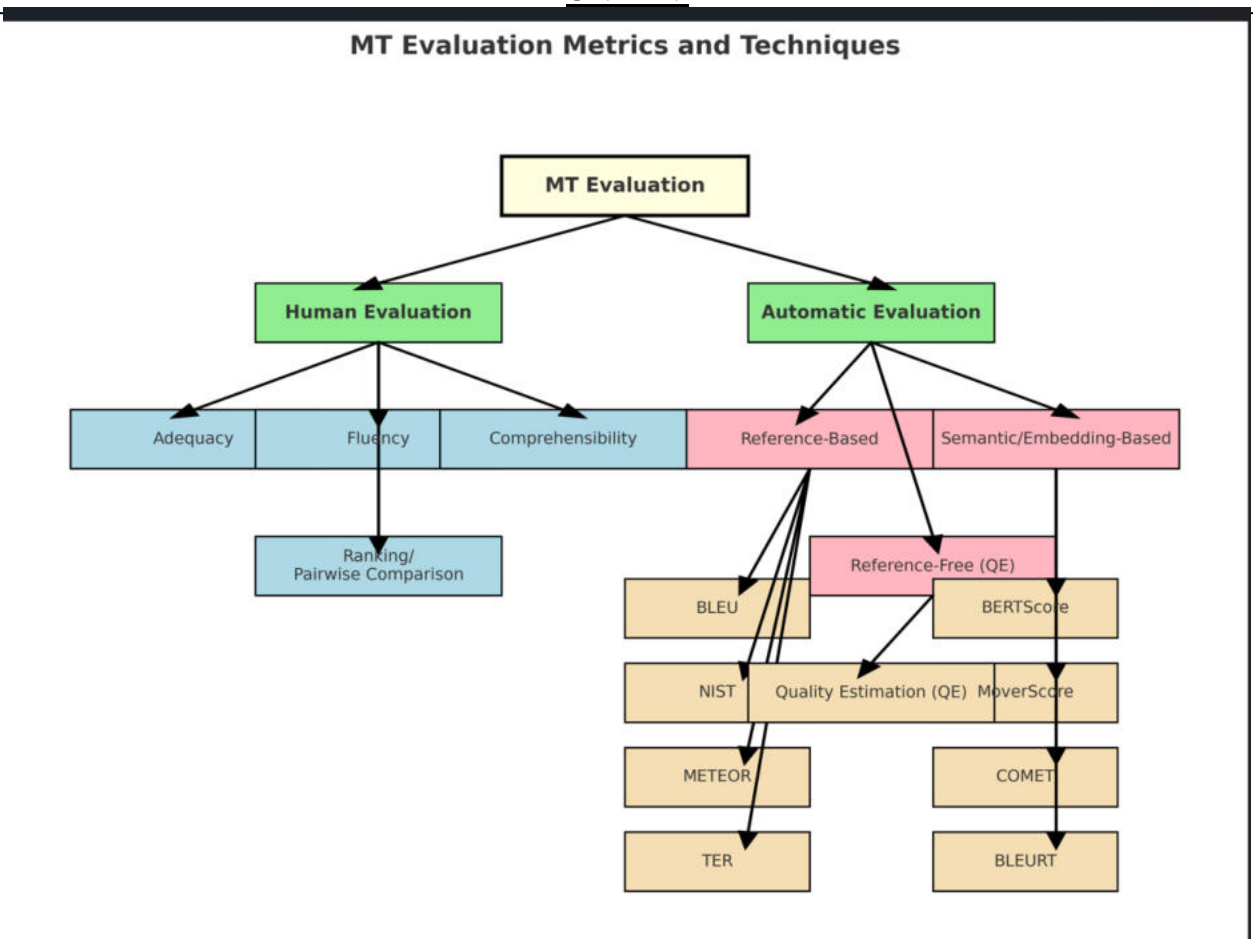
Metric/Technique	Type	Strengths	Weaknesses
Human Evaluation	Manual	Nuanced, captures context	Costly, subjective
BLEU	Automatic	Fast, standard, n-gram based	Ignores meaning, synonyms
NIST	Automatic	Considers informative n-grams	Similar limitations to BLEU
METEOR	Automatic	Synonyms, stemming, paraphrasing	Slower, language-dependent

Metric/Technique	Type	Strengths	Weaknesses
TER	Automatic	Measures effort to correct	May not capture fluency
BERTScore	Automatic	Semantic, contextual	Requires pretrained models
COMET / BLEURT	Automatic	Strong correlation with human judgment	Computationally expensive
QE (Quality Estimation)	Automatic, no reference	Works for low-resource	Less reliable without large training data

5. Conclusion

- **Human evaluation** = gold standard, but costly.
- **Automatic metrics** = fast, objective, scalable.
- Trend: Moving from **surface-level metrics (BLEU, METEOR)** to **semantic/neural metrics (BERTScore, COMET)** for better correlation with human judgment.

MT evaluation combines **human judgment** and **automatic metrics** (BLEU, METEOR, BERTScore, etc.) to ensure translations are both **accurate and fluent**.



BIAS AND ETHICAL ISSUES IN NLP AND MACHINE TRANSLATION

1. Introduction

- NLP and MT systems are widely used (e.g., Google Translate, chatbots, voice assistants).
- However, since they are trained on **large-scale data** collected from the internet, they often **inherit and amplify human biases** present in the data.
- These biases lead to **ethical challenges** like unfair treatment, stereotypes, and misinformation.

2. Sources of Bias in NLP & MT

1. **Data Bias:**
    - Training corpora may overrepresent some languages, cultures, or social groups.

- Example: More data for English and French → better translation quality than for Telugu or Yoruba.

2. **Representation Bias:**
   - Word embeddings capture stereotypes.
   - Example: *doctor → he, nurse → she* associations.
3. **Algorithmic Bias:**
   - Models may optimize for accuracy but ignore fairness.
   - Example: MT translating gender-neutral Turkish sentence *"O bir doktor" → "He is a doctor"* (incorrect gender assumption).
4. **Socio-Cultural Bias:**
   - Systems may fail with culturally specific idioms or dialects.

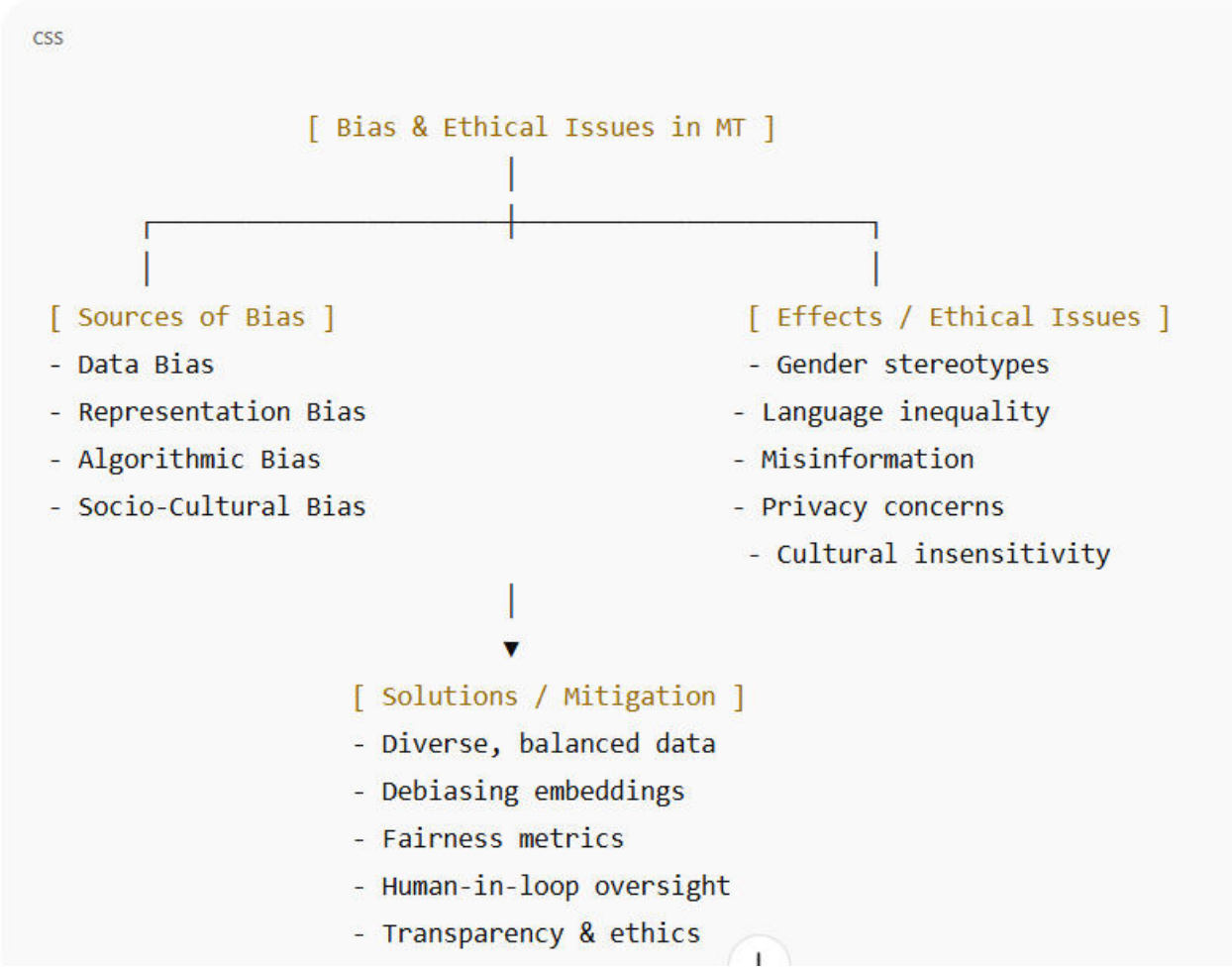
3. Ethical Issues in MT and NLP

1. **Gender & Stereotype Reinforcement:**
   - Translations can reinforce gender roles (*engineer → he*, *teacher → she*).
2. **Language Inequality (Digital Divide):**
   - High-resource languages get accurate MT; low-resource ones are neglected.
   - Leads to exclusion of communities from digital participation.
3. **Misinformation & Trustworthiness:**
   - Poor translations in healthcare/legal contexts may cause harm.
4. **Privacy Concerns:**
   - Using personal text for training without consent can violate privacy.
5. **Cultural Insensitivity:**
   - Incorrect translations of sensitive terms can cause offense or misunderstanding.
6. **Accountability & Transparency:**
   - Users may not know why a system made a biased or wrong translation.

4. Mitigation Strategies

- **Bias-Aware Data Collection:** Balanced, diverse corpora covering different dialects, genders, and contexts.
- **Debiasing Techniques in Embeddings:** Modify word embeddings (e.g., gender-neutral embeddings).
- **Fairness Metrics in MT:** Evaluate translations not only for accuracy (BLEU) but also for fairness and neutrality.
- **Human-in-the-Loop Systems:** Involve human review for sensitive domains (healthcare, law).
- **Transparency & Explainability:** Use Explainable AI (XAI) to show why a translation was produced.
- **Ethical Guidelines & Policies:** Industry and academia must enforce ethical standards.
- **Bias and ethics** in NLP/MT are critical because models influence billions of users daily.
- Problems include **gender stereotypes, language inequality, cultural insensitivity, and privacy risks**.
- Solutions require **better data practices, fairness-aware algorithms, and human oversight** to build **trustworthy and inclusive MT systems**.

Bias in MT arises from **data, algorithms, and cultural factors**, leading to ethical risks like **stereotyping, inequality, and misinformation**.
Ethical MT requires **fair data, debiasing methods, transparency, and accountability**.



SEMANTIC ANALYSIS AND FIRST-ORDER LOGIC IN NLP

- **Semantic Analysis** → Extracts **meaning** from text.
- **FOL** → Provides a **formal, logic-based way** to represent that meaning for reasoning.

Semantic Analysis is about understanding the **meaning** of text beyond its structure. It deals with mapping natural language into **machine-readable representations**.

Key Tasks in Semantic Analysis:

1. **Word Sense Disambiguation (WSD):** Resolving meaning of words in context
    - *Example:* "bank" → river bank vs financial bank
2. **Semantic Role Labeling (SRL):** Identifying roles in sentences
    - *Example:* "John ate an apple" → John = Agent, Apple = Patient

3. **Thematic Relations:** Relations like *cause, effect, instrument, location*
4. **Semantic Parsing:** Converting sentences into formal structures (logic-based or graph-based)
5. **Compositional Semantics:** Meaning of a sentence = combination of meanings of its words + grammar

🔢 First-Order Logic (FOL) in NLP

FOL provides a **formal representation of meaning** in natural language using:

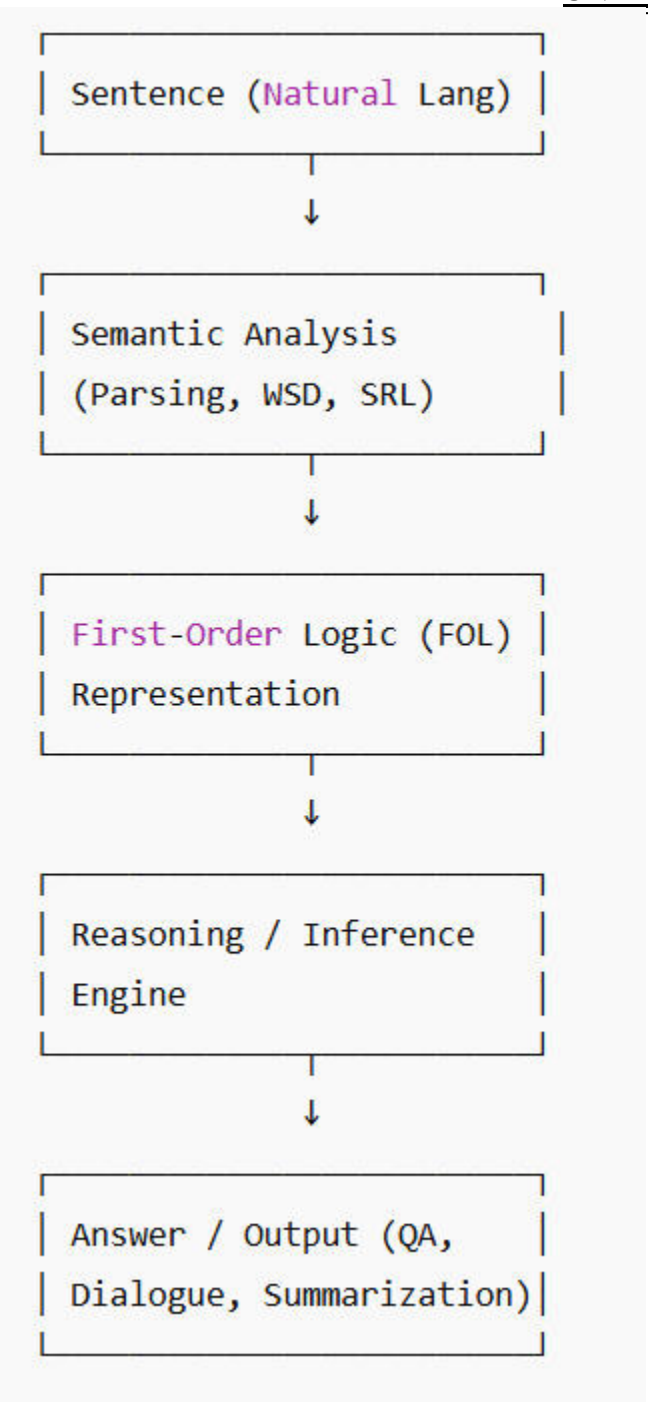
- **Constants:** Specific objects (e.g., *John, Apple*)
- **Predicates:** Properties or relations (e.g., *Loves(John, Mary)*)
- **Quantifiers:**
  - ∀ (for all) → Universal quantification
  - ∃ (there exists) → Existential quantification

Example Conversions:

- Sentence: *"All humans are mortal."*
  FOL: ∀x [Human(x) → Mortal(x)]
- Sentence: *"Some cats are black."*
  FOL: ∃x [Cat(x) ∧ Black(x)]
- Sentence: *"John loves Mary."*
  FOL: Loves(John, Mary)

⚙️ Applications in NLP

- **Question Answering (QA):** Convert question → FOL → match with KB facts
- **Information Retrieval:** Logical forms help filter relevant facts
- **Text Summarization:** Identify semantic structures and relations
- **Dialogue Systems:** Represent intents and reasoning in logic form



THEMATIC ROLES AND SELECTIONAL RESTRICTIONS IN SEMANTICS

- **Thematic roles** assign **semantic functions** to sentence participants.
- **Selectional restrictions** impose **plausibility checks** on those roles.

1. Thematic Roles (Semantic Roles)

Thematic roles (also called **semantic roles**) describe how **participants** in an event are related to the **verb (predicate)**.
They capture the **who did what to whom, when, where, and how** in a sentence.

Common Thematic Roles:

- **Agent** – the doer of an action
  Example: John (Agent) opened the door.
- **Experiencer** – entity experiencing a state/emotion
  Example: Mary (Experiencer) felt happy.
- **Theme/Patient** – entity affected or moved by an action
  Example: John ate *an apple* (Theme).
- **Instrument** – object used to perform an action
  Example: He cut the bread with *a knife* (Instrument).
- **Beneficiary** – entity for whom an action is performed
  Example: She baked a cake for *her friend* (Beneficiary).
- **Location** – where the action happens
  Example: The party is at *the park* (Location).
- **Goal / Source** – endpoint or starting point of movement
  Example: He traveled *to Paris* (Goal) from *London* (Source).

2. Selectional Restrictions

Selectional restrictions are **semantic constraints** imposed by verbs (predicates) on their arguments.
They ensure that the arguments are **semantically compatible** with the verb.

Examples:

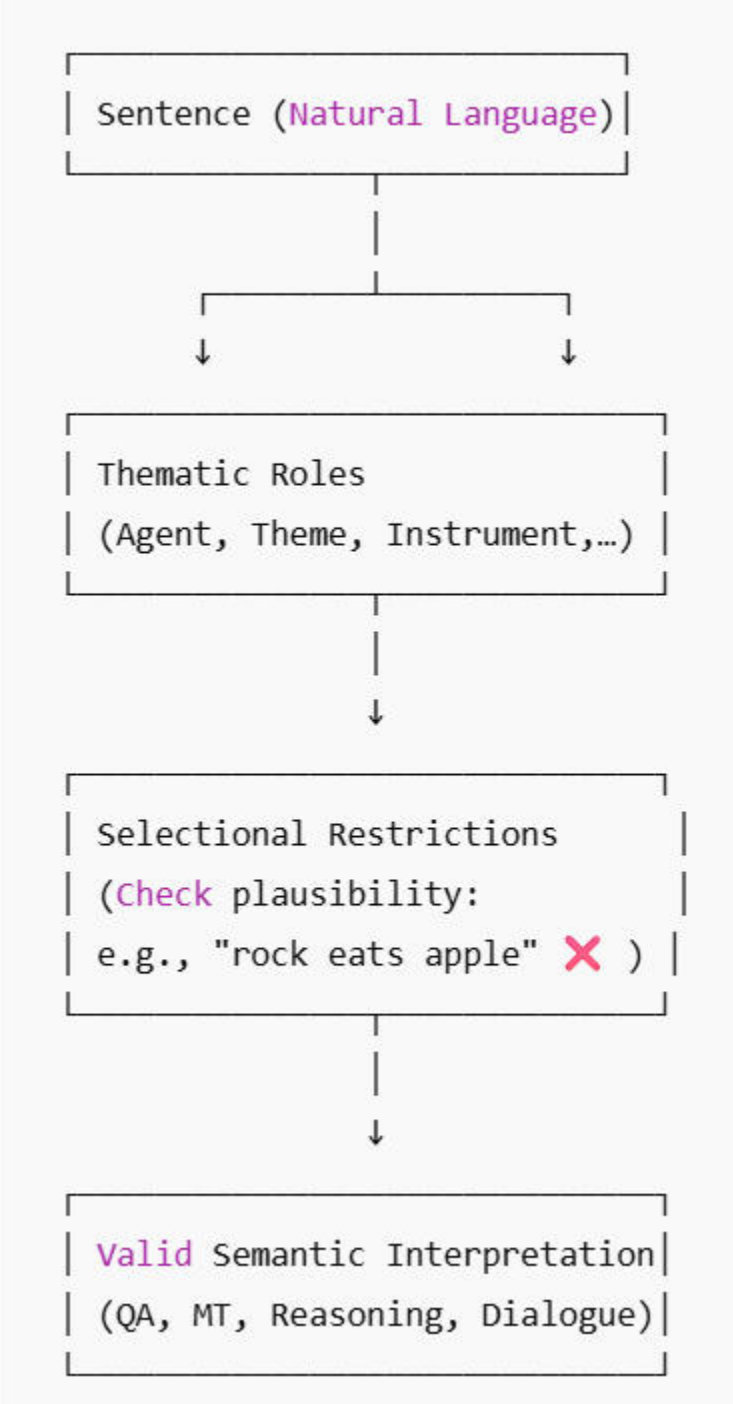
- *"The boy ate an apple."* ✅ (boy = animate Agent, apple = edible Theme)
- *"The rock ate an apple."* ❌ (rock cannot be an animate Agent of "eat")
- *"She drank water."* ✅ (drink requires liquid Theme)
- *"She drank a chair."* ❌ (chair violates selectional restrictions)

Why important?

- Prevents **nonsensical interpretations** in NLP.
- Helps in **Word Sense Disambiguation** (WSD).
- Useful in **semantic parsing, question answering, and MT**.

3. Thematic Roles + Selectional Restrictions in NLP

- Thematic roles help determine **relations between entities**.
- Selectional restrictions **validate plausibility** of those relations.
- Together, they guide **semantic interpretation** of natural language.



WORD SENSES AND RELATIONS BETWEEN SENSES

- **Word senses** = different meanings of a word.

- **Relations between senses** = structured links (synonymy, antonymy, hypernymy, etc.) that organize lexical knowledge.

1. Word Senses

- A **word sense** is a specific meaning of a word in a given context.
- Many words are **polysemous** (multiple related senses) or **homonymous** (unrelated senses).

Example:

- Word: **bank**
  - Sense 1: *financial institution* → "I deposited money in the bank."
  - Sense 2: *side of a river* → "They sat on the river bank."

Word Sense Disambiguation (WSD) is the NLP task of identifying which sense of a word is used in context.

2. Relations Between Word Senses

Lexical semantics defines several **semantic relations** between senses, many of which are captured in **WordNet**.

a) Synonymy (same or similar meaning)

- *big* ↔ *large*
- Used in **thesaurus-based IR**.

b) Antonymy (opposite meaning)

- *hot* ↔ *cold*
- Common in **sentiment analysis**.

c) Hyponymy / Hypernymy (IS-A relation)

- Hyponym: *rose* is a type of *flower*.
- Hypernym: *flower* is a super-category of *rose*.
- Used in **taxonomy construction**.

d) Meronymy / Holonymy (Part-Whole relation)

- Meronym: *wheel* is part of a *car*.

- Holonym: *car* has *wheels*.

e) Polysemy (multiple related senses)

- *head* → of a person, a company, a bed.

f) Homonymy (different, unrelated senses)

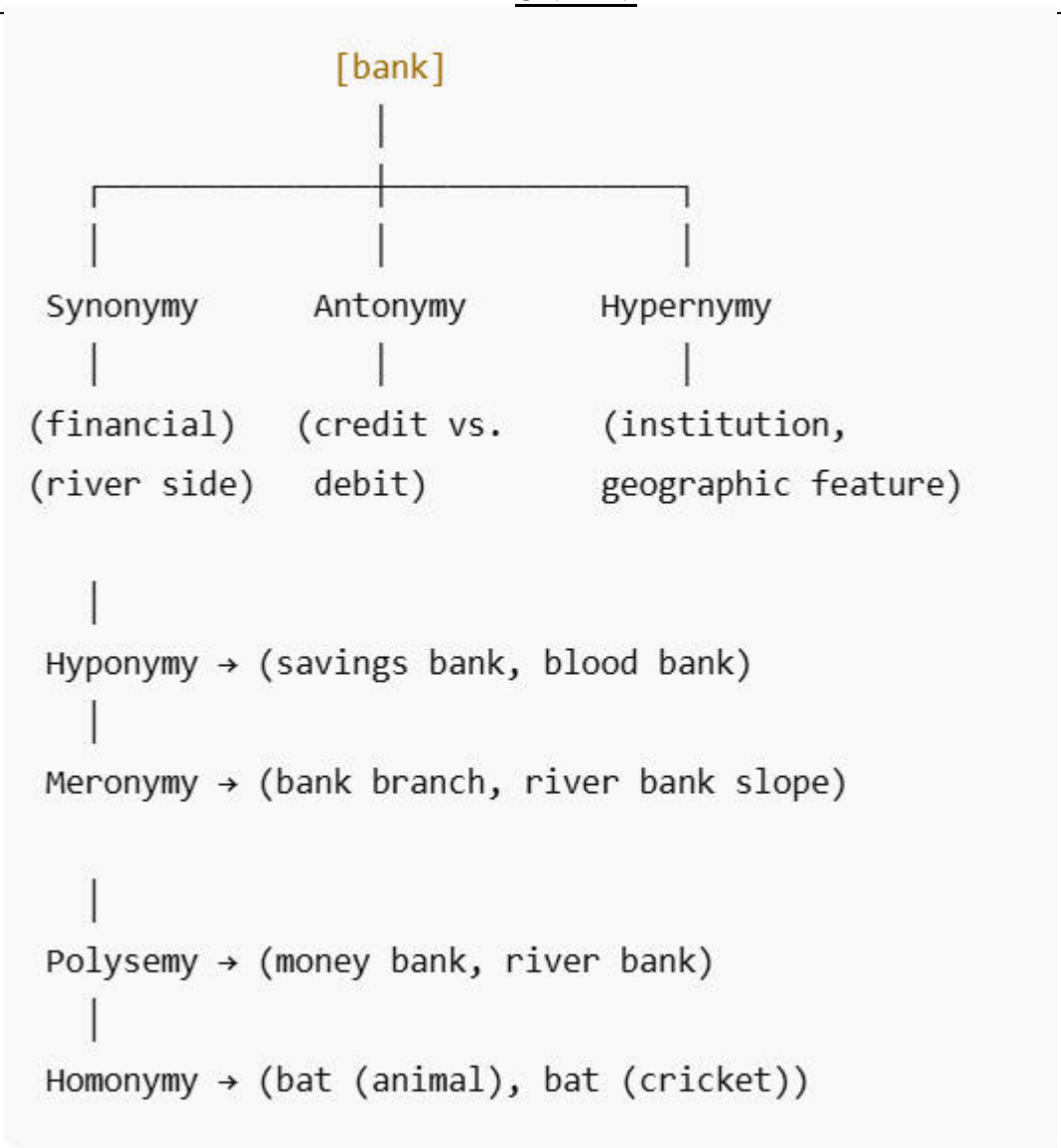
- *bat* → an animal, a cricket bat.

g) Troponymy (manner relation between verbs)

- *walk* ↔ *stroll* (to walk in a leisurely manner).

3. Importance in NLP

- **WSD**: Disambiguating polysemy in translation, QA.
- **MT**: Correct sense selection → accurate translation.
- **IR/Search**: Synonymy and hypernymy expand queries.
- **Ontologies**: Relations form the basis of semantic networks.


[bank]
Synonymy
Antonymy
Hypernymy
(financial)
(river side)
(credit vs.
debit)
(institution,
geographic feature)
Hyponymy → (savings bank, blood bank)
Meronymy → (bank branch, river bank slope)
Polysemy → (money bank, river bank)
Homonymy → (bat (animal), bat (cricket))

UNIT V: Speech Processing and Advanced NLP Models

Speech Fundamentals: Phonetics and Acoustic Phonetics, Digital Signal Processing in Speech Analysis, Feature Extraction in Speech: Short-Time Fourier Transform (STFT), Mel-Frequency Cepstral Coefficients (MFCC) and Perceptual Linear Prediction (PLP), Hidden Markov Models (HMMs) in Speech Recognition.

SPEECH PROCESSING AND ADVANCED NLP MODELS

1. Introduction

- **Speech Processing** bridges human spoken language and computational systems.
- It covers **speech recognition, speech synthesis, speaker identification, and spoken dialogue systems**.
- Advanced NLP models (deep learning, transformers, etc.) are now integrated with speech to power **voice assistants, MT, and multimodal AI**.

2. Speech Processing

a) Speech Recognition (ASR – Automatic Speech Recognition)

- Converts spoken input into text.
- Pipeline:
  - **Acoustic Modeling** → maps audio signals to phonemes.
  - **Language Modeling** → predicts word sequences.
  - **Decoding** → selects the most likely sentence.
- **Example:** Google Speech API, Alexa.

b) Speech Synthesis (TTS – Text-to-Speech)

- Converts text into natural-sounding speech.
- Methods:
  - Concatenative synthesis (unit selection).
  - Parametric synthesis (HMM-based).
  - Neural TTS (WaveNet, Tacotron).

c) Speaker Recognition

- **Speaker Identification** → "Who is speaking?"
- **Speaker Verification** → "Is this person who they claim to be?"
- Used in security, banking apps.

d) Challenges in Speech Processing

- Noise, accents, dialects.
- Low-resource languages.
- Code-switching (mix of languages).

3. Advanced NLP Models

a) Deep Learning in NLP

- RNNs, LSTMs, and GRUs → capture sequential dependencies.
- Used in **speech recognition, MT, sentiment analysis**.

b) Transformer Models

- Replace recurrence with **self-attention mechanism**.
- Examples: **BERT, GPT, T5, BART**.
- Advantages: parallelization, better handling of context.

c) Large Language Models (LLMs)

- Pre-trained on massive corpora.
- Capabilities: text generation, translation, reasoning, speech-to-text integration.
- Examples: **GPT-4, PaLM, LLaMA**.

d) Speech + NLP Integration

- End-to-end **speech-to-text translation** (e.g., Meta's SeamlessM4T, OpenAI Whisper).
- Multimodal models: handle **speech + text + image** together.

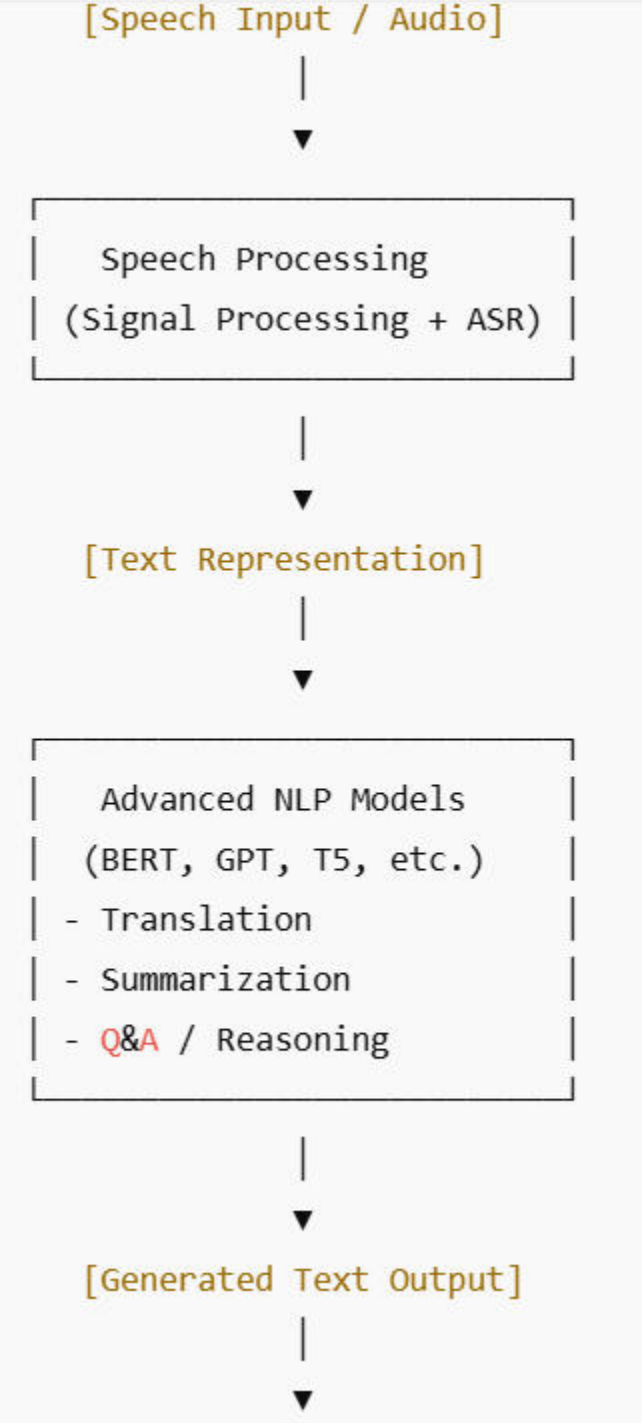
4. Applications

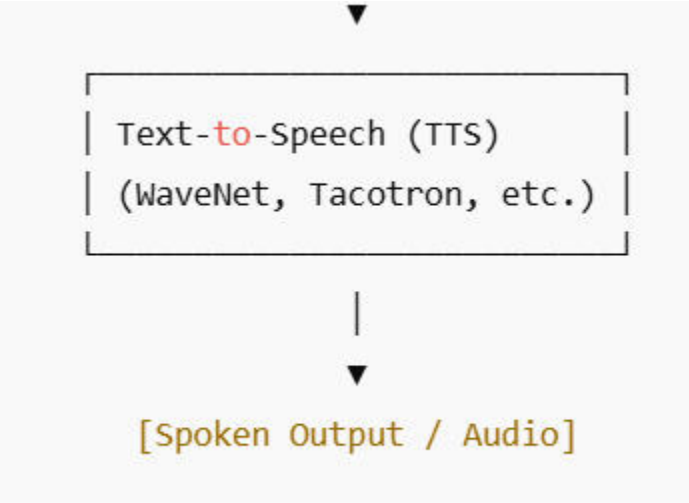
- Voice assistants (Siri, Alexa, Google Assistant).

- Real-time speech translation (Skype Translator, Google Translate).
- Healthcare dictation systems.
- Accessibility tools (screen readers, speech therapy).

5. Conclusion

- Speech Processing enables **human-computer interaction through spoken language**.
- Advanced NLP models (Transformers, LLMs) push speech technologies to new levels.
- Future: **multilingual, multimodal, and context-aware speech systems**.


[Speech Input / Audio]
Speech Processing
(Signal Processing + ASR)
[Text Representation]
Advanced NLP Models
(BERT, GPT, T5, etc.)
- Translation
- Summarization
- Q&A / Reasoning
[Generated Text Output]



SPEECH FUNDAMENTALS

Speech fundamentals form the base for all **speech and language technologies**. Understanding **production, signal properties, and types of sounds** helps in building efficient speech-based NLP systems.

1. Introduction

- Speech is the **primary mode of human communication**.
- In **Speech Processing**, understanding speech fundamentals is essential for tasks like **ASR (Automatic Speech Recognition), TTS (Text-to-Speech), and Speaker Recognition**.

2. Speech Production Process

- Human speech is produced through the **vocal tract system**:
    1. **Lungs** → provide airflow.
    2. **Vocal cords (glottis)** → vibrate to produce voiced sounds.
    3. **Articulators** (tongue, lips, teeth, palate) → shape sounds into phonemes.

3. Key Properties of Speech Signal

- **Time-domain features:** waveform, amplitude.
- **Frequency-domain features:** spectrum, formants.
- **Pitch (F0):** perceived as tone of voice, related to vocal cord vibration.

- **Formants:** resonant frequencies of the vocal tract.
- **Phonemes:** smallest speech units (e.g., /p/, /a/, /t/).

4. Speech Signal Characteristics

- **Analog in nature**, continuous wave.
- **Digitization:** speech is sampled & quantized to store in digital form.
- **Sampling rate:** commonly 8 kHz (telephony) or 16 kHz (ASR).
- **Spectrogram:** visual representation (time vs. frequency vs. energy).

5. Types of Speech Sounds

- **Voiced sounds** → vocal cords vibrate (e.g., /a/, /b/).
- **Unvoiced sounds** → no vibration (e.g., /s/, /f/).
- **Vowels** → open vocal tract, periodic.
- **Consonants** → constricted vocal tract, can be noisy.

6. Challenges in Speech Fundamentals

- **Coarticulation** → overlap of sounds.
- **Accents & dialects** → variations in pronunciation.
- **Background noise** → distorts signal.
- **Speaker variability** → pitch, speed, style differences.

7. Applications

- **Speech recognition (ASR).**
- **Voice synthesis (TTS).**
- **Forensic analysis & speaker identification.**
- **Language learning tools.**

PHONETICS AND ACOUSTIC PHONETICS

UNIT V

Phonetics provides the foundation for studying speech sounds, while **acoustic phonetics bridges linguistics and signal processing** by analyzing sound wave properties. This is essential for **ASR, TTS, and other speech-based NLP systems**.

1. Introduction

- **Phonetics** is the study of **speech sounds** – how they are produced, transmitted, and perceived.
- **Acoustic phonetics** is a branch of phonetics focusing on the **physical properties of speech sounds** as sound waves.

2. Branches of Phonetics

1. **Articulatory Phonetics**
    - Studies how speech sounds are produced by the **vocal organs** (tongue, lips, vocal cords).
    - Example: how /p/ differs from /s/.
2. **Acoustic Phonetics**
    - Examines the **physical characteristics of sound waves**.
    - Properties studied: **frequency, amplitude, duration, formants**.
3. **Auditory Phonetics**
    - Studies how humans **perceive and process speech sounds** using the auditory system.

3. Acoustic Properties of Speech Sounds

- **Frequency (Pitch):** rate of vocal cord vibration (measured in Hertz).
- **Intensity (Loudness):** energy of the sound wave.
- **Duration:** length of the sound.
- **Formants:** resonant frequencies that distinguish vowels.
- **Spectrogram:** visual tool showing time, frequency, and intensity.

4. Phonetic Units

- **Phoneme:** smallest unit of sound that changes meaning (*pat* vs. *bat*).

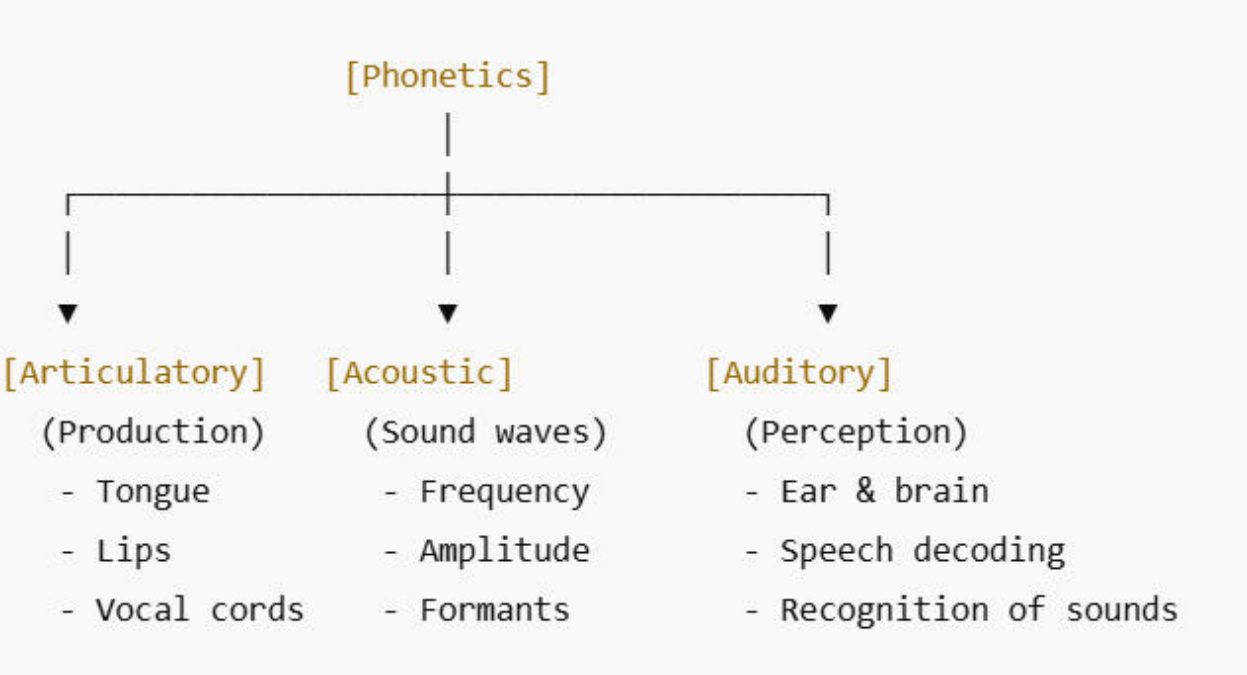
- **Allophone:** variation of a phoneme that doesn't change meaning (*top* vs. *stop*).

5. Applications in NLP & Speech Processing

- **Automatic Speech Recognition (ASR):** converts speech to text.
- **Text-to-Speech (TTS):** synthesizes natural-sounding speech.
- **Speaker Identification:** analyzing unique phonetic-acoustic features.
- **Language Learning Tools:** pronunciation training.

6. Challenges in Acoustic Phonetics

- **Coarticulation:** overlap of sounds in continuous speech.
- **Noise & distortions:** affect accuracy of acoustic features.
- **Accent & dialect variation.**



DIGITAL SIGNAL PROCESSING IN SPEECH ANALYSIS

DSP is the backbone of modern **speech analysis**. By transforming speech into digital form and extracting features, it enables powerful applications in **ASR, TTS, speaker recognition, and healthcare AI**.

1. Introduction

- **Digital Signal Processing (DSP)** involves applying mathematical and computational techniques to analyze, modify, and synthesize speech signals.
- In **speech analysis**, DSP helps extract features from speech for recognition, synthesis, and enhancement.

2. Why DSP in Speech?

- Speech is an **analog waveform** → converted into **digital form** for processing.
- DSP enables:
    - Noise reduction
    - Compression
    - Feature extraction (for ASR, speaker ID)
    - Enhancement of intelligibility

3. Steps in Speech Signal Processing

1. **Speech Acquisition**
    - Microphone records speech (analog).
    - **Sampling** converts it to digital (e.g., 16 kHz, 44.1 kHz).
    - **Quantization** approximates amplitude into discrete levels.
2. **Pre-Processing**
    - **Pre-emphasis filter** → boosts high frequencies.
    - **Framing** → divide signal into short frames (10–30 ms).
    - **Windowing (Hamming/Hanning)** → reduce discontinuities at frame edges.
3. **Spectral Analysis**

- **Fourier Transform (FFT):** converts time domain → frequency domain.
- **Spectrograms:** visualize energy distribution across frequency & time.
- **Formant Analysis:** resonances used in vowel classification.

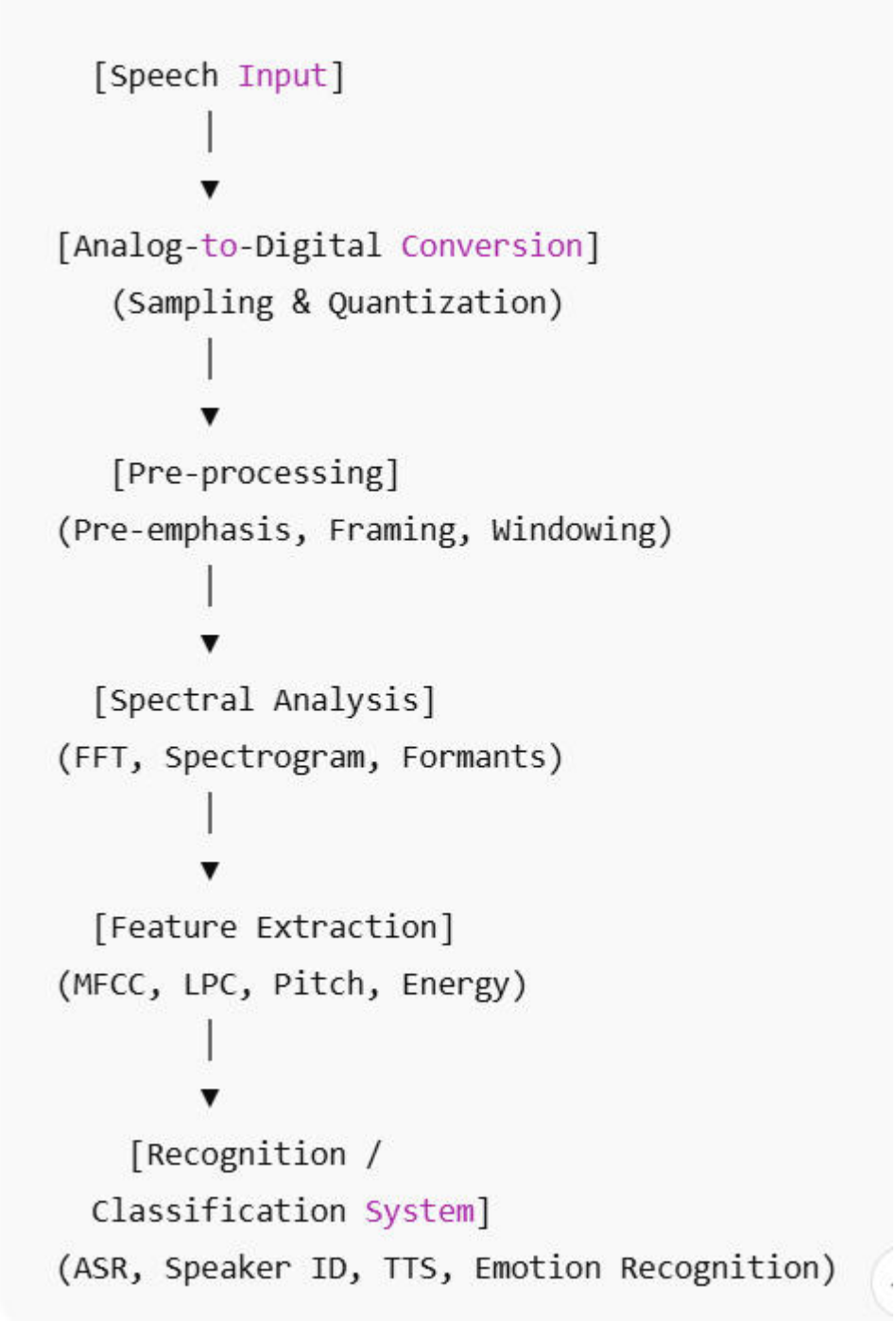
4. **Feature Extraction**
   - **Mel-Frequency Cepstral Coefficients (MFCCs):** widely used in ASR.
   - **Linear Predictive Coding (LPC):** models speech production.
   - **Pitch & Energy estimation:** for prosody analysis.
5. **Post-Processing / Applications**
   - **Automatic Speech Recognition (ASR)**
   - **Speaker Identification & Verification**
   - **Speech Synthesis (TTS)**
   - **Emotion Recognition**

4. Challenges in DSP for Speech

- **Background Noise** (affects accuracy).
- **Variability in speakers** (age, accent, health).
- **Real-time processing** requirements.

5. Applications in NLP & AI

- Virtual Assistants (Alexa, Siri, Google Assistant).
- Forensic speaker identification.
- Real-time translation systems.
- Medical diagnostics (speech-based Parkinson's detection).



FEATURE EXTRACTION IN SPEECH

Feature extraction transforms **raw, complex speech data** into **compact, discriminative representations**, making it the **core of speech analysis and NLP-based applications**.

1. Introduction

- **Feature Extraction** in speech processing refers to transforming raw speech signals into a set of **compact, discriminative, and meaningful parameters**.
- Goal: represent speech **efficiently** for tasks like **ASR (Automatic Speech Recognition), TTS (Text-to-Speech), Speaker Identification, and Emotion Recognition**.

2. Why Feature Extraction?

- Speech signals are **high-dimensional and redundant**.
- Features reduce complexity while preserving **linguistic and speaker information**.
- Good features must be:
  - **Robust** to noise and channel variations.
  - **Discriminative** between speakers and phonemes.
  - **Compact** for fast processing.

3. Common Speech Features

(a) Spectral Features

1. **Mel-Frequency Cepstral Coefficients (MFCCs)**
   - Most widely used.
   - Mimic human auditory perception (mel scale).
   - Extract spectral envelope for phoneme recognition.
2. **Linear Predictive Coding (LPC)**
   - Models the vocal tract as a filter.
   - Captures formant structure of speech.
3. **Perceptual Linear Prediction (PLP)**
   - Similar to LPC but incorporates psychoacoustic models.

(b) Prosodic Features

1. **Pitch (Fundamental Frequency, F0):** conveys tone, stress, and intonation.

2. **Energy (Intensity):** loudness; useful in emotion recognition.
3. **Duration / Speaking Rate:** helps in rhythm and language modeling.

(c) Temporal & Other Features

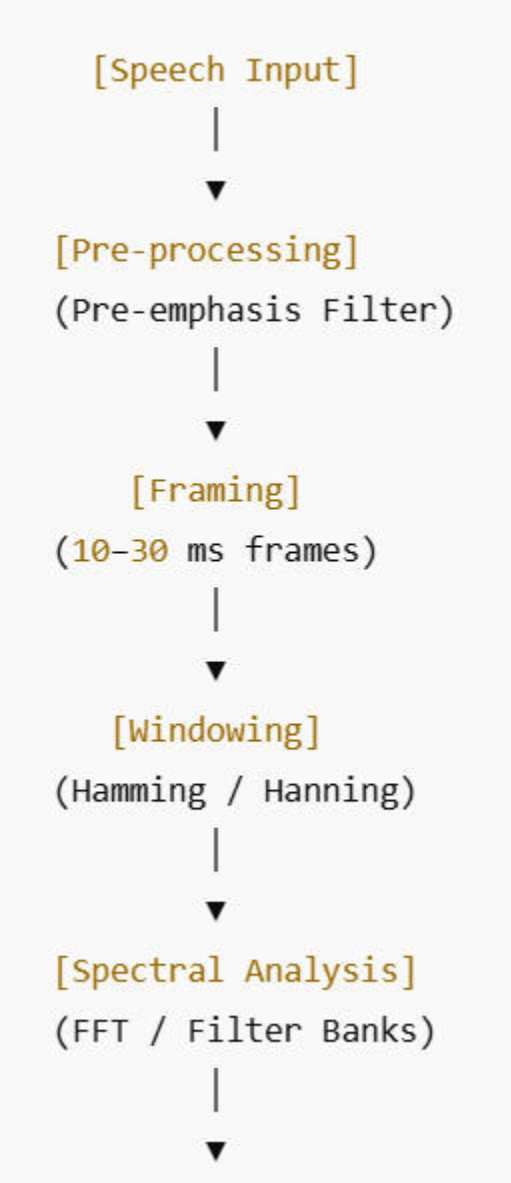
- **Delta & Delta-Delta Coefficients:** represent changes in MFCCs over time.
- **Zero Crossing Rate (ZCR):** useful for voiced/unvoiced classification.
- **Formant Frequencies:** resonant frequencies distinguishing vowels.

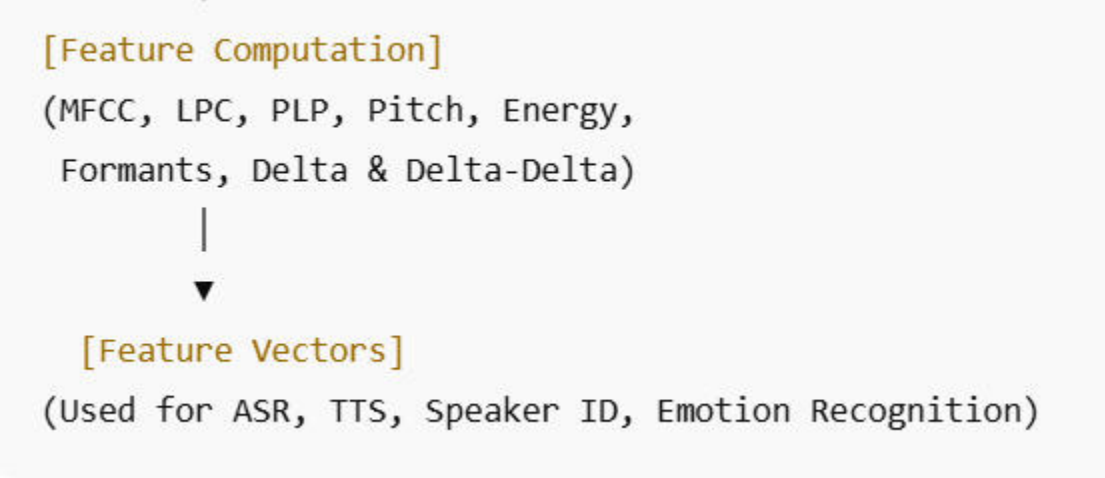
4. Feature Extraction Pipeline

1. **Speech Acquisition** → Recording through microphone.
2. **Pre-emphasis** → Boost high frequencies.
3. **Framing** → Divide signal into small segments (10–30 ms).
4. **Windowing** → Apply Hamming/Hanning window.
5. **Spectral Analysis** → FFT or filter banks.
6. **Feature Computation** → MFCC, LPC, pitch, energy.

5. Applications

- **ASR:** Phoneme and word recognition.
- **Speaker Identification:** Using LPC/MFCC for voiceprints.
- **Emotion Detection:** Using prosodic features like pitch and energy.
- **Language Processing:** Intonation and rhythm analysis.


[Speech Input]
[Pre-processing]
(Pre-emphasis Filter)
[Framing]
(10–30 ms frames)
[Windowing]
(Hamming / Hanning)
[Spectral Analysis]
(FFT / Filter Banks)



SHORT-TIME FOURIER TRANSFORM (STFT)

The Short-Time Fourier Transform (STFT) is a fundamental DSP tool in speech analysis, enabling time–frequency analysis of non-stationary signals like human speech.

1. Introduction

- Speech is a **non-stationary signal** (its frequency content changes over time).
- The **Fourier Transform (FT)** gives frequency information but loses time resolution.
- The **Short-Time Fourier Transform (STFT)** solves this by analyzing **small segments (windows)** of the signal, providing **time–frequency representation**.

2. STFT Concept

- Divide the speech signal into **short overlapping frames** (e.g., 20–40 ms).
- Apply a **window function** (Hamming/Hanning).
- Perform **Fourier Transform** on each frame.
- Result: **Spectrogram** → a 2D plot (time vs frequency vs magnitude).

3. STFT Mathematical Definition

For signal $x(t)$:

$$STFT\{x(t)\}(\tau,\omega) = \int_{-\infty}^{\infty} x(t) \cdot w(t-\tau) \cdot e^{-j\omega t} dt$$

Where:

- $w(t-\tau)$ = window function centered at time τ.
- ω = angular frequency.

4. Steps in STFT

1. **Speech Input** → continuous-time signal.
2. **Framing & Windowing** → isolate short segment.
3. **Apply Fourier Transform** → extract frequency components.
4. **Repeat across frames** → capture how spectrum changes over time.
5. **Visualize** → Spectrogram (used widely in speech recognition & phonetics).

5. Applications in Speech Processing

- **Speech Recognition (ASR):** extracting time–frequency features.
- **Speaker Identification:** capturing vocal tract signatures.
- **Emotion Recognition:** analyzing prosodic variations.
- **Music/Speech Separation:** separating overlapping sources.
- **Noise Reduction & Enhancement:** filtering noise in time–frequency space.

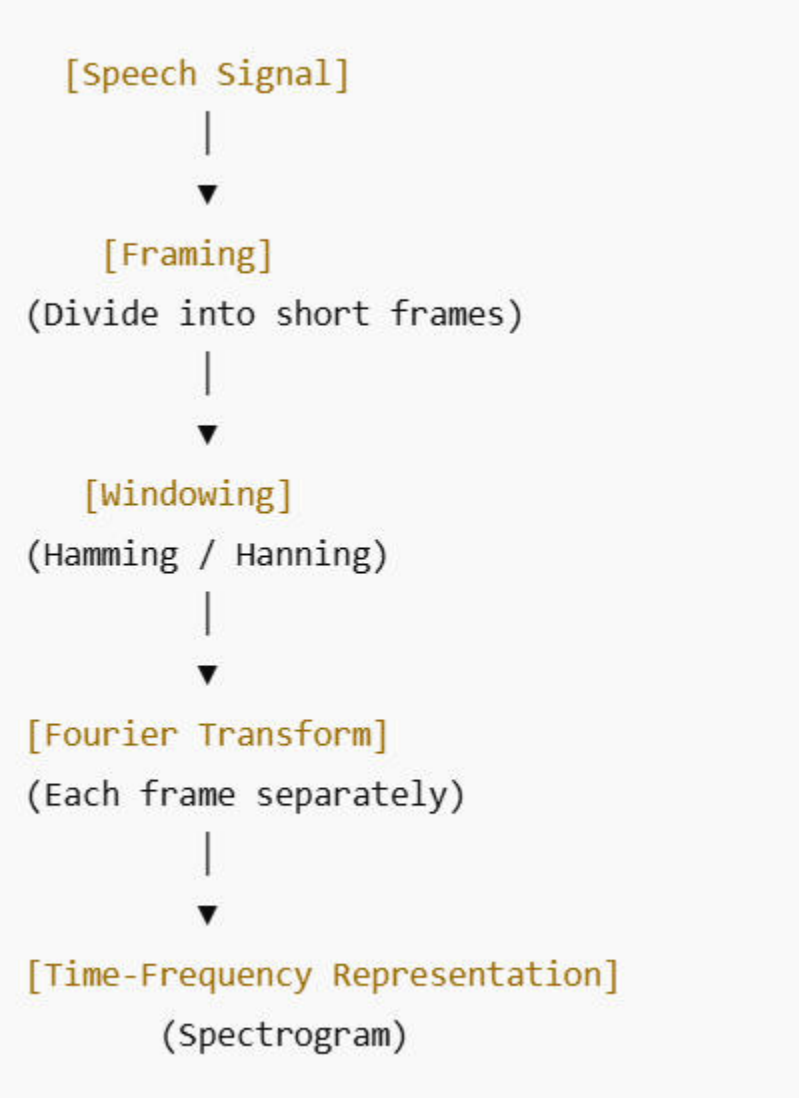
6. Advantages & Limitations

✅ **Advantages:**

- Provides **both time and frequency information**.
- Useful for analyzing **non-stationary signals** like speech.

⚠️ **Limitations:**

- **Fixed resolution**: trade-off between time and frequency (uncertainty principle).
- Choice of **window size** is critical:
  - Small window → better time resolution, poor frequency resolution.
  - Large window → better frequency resolution, poor time resolution.



MEL-FREQUENCY CEPSTRAL COEFFICIENTS (MFCC) AND PERCEPTUAL LINEAR PREDICTION (PLP)

- **MFCC = widely used, simple, effective, but noise-sensitive.**

- **PLP = psychoacoustic, more noise-robust, better for practical ASR.**

1. Introduction

Speech recognition requires **compact, discriminative, and perceptually motivated features**. Two popular methods are:

- **MFCC** – based on the Mel-scale of human hearing.
- **PLP** – based on auditory perception and linear prediction.

2. Mel-Frequency Cepstral Coefficients (MFCC)

Concept:

- Mimics how humans perceive sound frequencies.
- Uses a **Mel-scale filter bank** that emphasizes low frequencies (where human hearing is more sensitive).
- Produces **cepstral coefficients** representing speech spectrum compactly.

Steps:

1. **Pre-emphasis** – boost high frequencies.
2. **Framing & Windowing** – short segments (20–40 ms).
3. **FFT** – convert to frequency domain.
4. **Mel Filter Bank** – apply triangular filters spaced on Mel scale.
5. **Logarithm** – mimic human loudness perception.
6. **DCT (Discrete Cosine Transform)** – decorrelate features, keep 12–13 MFCCs.

Applications:

- Automatic Speech Recognition (ASR).
- Speaker Identification.
- Emotion Recognition.

3. Perceptual Linear Prediction (PLP)

Concept:

- Developed by **Hermansky (1990)**.
- Based on **psychoacoustic principles** of human hearing.
- Uses **Linear Prediction Analysis** but modifies the spectrum to match auditory perception.

Steps:

1. **Critical Band Analysis** – filters based on Bark scale.
2. **Equal Loudness Pre-emphasis** – compensates for ear sensitivity.
3. **Intensity Loudness Compression** – cube-root compression.
4. **LP Analysis** – models vocal tract with all-pole filter.
5. **Cepstral Coefficients** – converted from LP coefficients.

Applications:

- ASR under noisy conditions.
- Robust speech recognition systems.

4. Comparison: MFCC vs PLP

Aspect	**MFCC**	**PLP**
Scale	Mel scale (human pitch perception)	Bark scale (critical bands of hearing)
Compression	Logarithm (log energy)	Cube-root (loudness perception)
Basis	Cepstral (DCT of log energies)	Linear Prediction (LP + perceptual model)
Robustness	Sensitive to noise	More robust in noisy environments
Applications	ASR, Speaker ID, Emotion recog.	ASR, esp. noisy speech recognition

MFCC Pipeline

```
CSS

[Speech Signal] → [Pre-emphasis] → [Framing & Windowing]
      → [FFT] → [Mel Filter Bank] → [Log] → [DCT] → [MFCCs]
```

PLP Pipeline

```
CSS

[Speech Signal] → [Critical Band Analysis]
      → [Equal Loudness] → [Intensity Compression]
      → [LP Analysis] → [Cepstral Coefficients]
```

HIDDEN MARKOV MODELS (HMMS) IN SPEECH RECOGNITION

HMMs were the **foundation of modern speech recognition**, providing the statistical framework for modeling phoneme sequences.
While **deep learning has surpassed HMMs**, many systems still use **hybrid HMM-DNN models** in real-world ASR.

1. Introduction

- **Hidden Markov Model (HMM):** A probabilistic model used to represent **sequential data** such as speech.
- Speech is a **time-varying signal** where sounds (phonemes) unfold over time.
- HMM captures **temporal dynamics** + **probabilistic state transitions**.

2. Why HMM for Speech?

- Speech has **variability** (speaker, rate, noise).
- Phonemes are not directly observable; they are **hidden states**.
- Acoustic signals are observable outputs generated by these hidden states.

- HMM provides a **mathematical framework** to model:
  - **State transitions** (sequence of phonemes).
  - **Observation likelihoods** (acoustic features).

3. Structure of HMM

- **States (S):** Represent phonemes or sub-phonemes (e.g., S = {s1, s2, ..., sn}).
- **Transitions (A):** Probabilities of moving from one state to another.
- **Observations (O):** Acoustic feature vectors (MFCCs, PLPs).
- **Emission Probabilities (B):** Likelihood of observing feature vector given a state.
- **Initial Probabilities (π):** Probability distribution of starting states.

Mathematically:

HMM = (A, B, π)

- **A** = state transition probabilities
- **B** = observation probability distribution
- **π** = initial state distribution

4. Key Problems in HMM

1. **Evaluation Problem:**
   - Given model (λ) and observation sequence (O), compute P(O|λ).
   - Solved using **Forward Algorithm**.
2. **Decoding Problem:**
   - Find most likely state sequence for given observation sequence.
   - Solved using **Viterbi Algorithm**.
3. **Training Problem:**
   - Adjust model parameters (A, B, π) to maximize P(O|λ).
   - Solved using **Baum-Welch Algorithm (EM algorithm)**.

---

5. Application in Speech Recognition

Pipeline:

```java
Speech Signal → Feature Extraction (MFCC/PLP) → HMM Modeling → Decoding → Recognized Words
```

- **Acoustic Model (HMM):** Maps feature vectors to phonemes.
- **Lexicon (Pronunciation Dictionary):** Maps phonemes to words.
- **Language Model (n-grams):** Provides word sequence probability.

---

6. Example: Word Recognition

Suppose we want to recognize the word "**cat**":

- HMM states represent phonemes: **/k/ /æ/ /t/**.
- Each phoneme modeled with 3 HMM states (beginning, middle, end).
- Input speech is converted into MFCC vectors.
- HMM computes the **most probable state sequence** that generated the observation.
- Result: recognized word.

7. ASCII Diagram of HMM in Speech

```rust
States (Hidden):   s1 ----> s2 ----> s3 ----> s4 ----> s5
                     ↘       ↘       ↘       ↘
Observations:       o1      o2      o3      o4 ... (MFCC vectors)
```

8. Advantages & Limitations

✅ **Advantages:**

- Captures temporal dynamics of speech.
- Provides efficient algorithms (Forward, Viterbi).
- Widely used in ASR before deep learning.

⚠ **Limitations:**

- Assumes **Markov property** (current state depends only on previous).
- Observation distributions often Gaussian → limited expressiveness.
- Struggles with long-term dependencies.
- Replaced by **Deep Neural Networks (DNN-HMM hybrids, end-to-end models like RNNs/Transformers)**.