upGrad

Introduction to Natural Language Processing

Definition

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) concerned with the computational modelling and analysis of human language.

It aims to enable computers to **understand, interpret, and generate** natural language in a manner that is both meaningful and useful.

In essence, NLP bridges the gap between **human communication** and **machine understanding**, drawing upon concepts from **linguistics, computer science, and machine learning**.

Motivation

Human language is inherently **ambiguous, context-dependent, and dynamic**.

A single lexical item may possess multiple meanings; for instance, the word *"bank"* may refer either to a financial institution or to the side of a river.

Similarly, the structure of a sentence can yield different interpretations, as in *"I saw the man with the telescope"*, where the phrase *"with the telescope"* may modify either *"man"* or *"saw"*.

Moreover, idiomatic expressions such as *"break the ice"* convey meanings that cannot be deduced from the literal definitions of their constituent words.

Humans resolve such ambiguities effortlessly through contextual reasoning and prior knowledge.

Computers, however, require explicit models and algorithms to approximate this ability.

This fundamental challenge constitutes the central motivation for research and development in NLP.

Objectives of NLP

The principal objective of NLP is to make natural language **computationally processable**.

Its goals may be summarised as follows:

1. **Language Understanding:** To enable machines to extract and represent the meaning of linguistic input.
2. **Language Generation:** To produce coherent and contextually appropriate natural-language output from structured or unstructured data.
3. **Human–Computer Interaction:** To facilitate seamless communication between humans and computers using everyday language rather than formal programming commands.

Through these objectives, NLP seeks to emulate aspects of human linguistic competence within computational systems.

Linguistic Foundations

The study of language from a computational perspective requires attention to several interrelated linguistic levels.

Each level contributes distinct information necessary for accurate processing and interpretation.

Level	Focus	Illustration
Phonology	Sound patterns and pronunciation.	"Knight" and "night" are phonologically identical.
Morphology	Internal structure of words and morphemes.	*un* + *happy* + *ness* → *unhappiness*.
Syntax	Arrangement of words into grammatical sentences.	"The cat chased the dog" differs in meaning from "The dog chased the cat."
Semantics	Meaning of words and sentences.	"Light" may denote *illumination* or *not heavy*.
Pragmatics	Intended meaning and contextual use.	"Can you pass the salt?" functions as a request, not a query about ability.

A comprehensive NLP system must address all these levels to approximate human-like understanding.

The NLP Processing Pipeline

Language data undergoes a sequence of computational stages collectively termed the **NLP pipeline**:

1. **Data Acquisition:** Collection of text or speech from relevant sources such as articles, transcripts, or conversations.
2. **Data Cleaning:** Removal of extraneous symbols, typographical errors, and non-linguistic artefacts (e.g., HTML tags, emojis).
3. **Text Pre-processing:** Standardisation through operations such as tokenisation, normalisation, stemming, and lemmatisation.
4. **Feature Extraction:** Transformation of linguistic units into quantitative representations, using methods such as Bag-of-Words, Term Frequency–Inverse Document Frequency (TF–IDF), or word embeddings.
5. **Model Construction:** Application of statistical or machine-learning algorithms (e.g., Logistic Regression, Naïve Bayes, neural networks) to detect patterns within the data.
6. **Evaluation and Deployment:** Assessment of model performance and integration within practical systems such as chatbots, translation engines, or search interfaces.

This pipeline converts unstructured linguistic data into structured information amenable to computation.

Applications of NLP

Natural Language Processing underpins a wide spectrum of contemporary technologies:

Domain	Representative Example	Function of NLP
Information Retrieval	Web search queries such as "nearest pharmacy open now".	Interprets grammatical structure and intent rather than relying solely on keyword matching.
E-commerce	Product review analysis and recommendation.	Identifies synonyms and sentiment to enhance relevance of recommendations.
Customer Support	Automated chatbots on service websites.	Resolves lexical ambiguities and interprets user intent.
Healthcare	Processing of clinical notes.	Extracts key medical entities and relationships (e.g., symptoms, diagnoses).
Machine Translation	Systems such as Google Translate.	Maps linguistic structure and meaning across languages.
Voice-based Assistants	Siri, Alexa, or Google Assistant.	Converts speech to text, interprets intent, and generates responses.

These applications demonstrate NLP's pervasive influence across both consumer and industrial domains.

Approaches to NLP

Historically, two principal paradigms have guided NLP research:

1. **Rule-Based Systems:**
    a. Early NLP systems relied on manually crafted grammatical and lexical rules.
    b. They achieved high precision in restricted domains but lacked scalability and adaptability.

2. **Statistical and Data-Driven Systems:**
    a. With the advent of large corpora and increased computational power, probabilistic and machine-learning approaches became dominant.
    b. These systems learn linguistic patterns from data, offering flexibility and improved performance across diverse contexts.

Contemporary NLP integrates both paradigms, combining the **interpretability of rules** with the **robustness of data-driven learning**.

Challenges in NLP

Despite significant progress, several challenges persist:

- **Ambiguity:** Multiple plausible interpretations at lexical, syntactic, or semantic levels.
- **Context Sensitivity:** Dependence of meaning on situational or pragmatic context.
- **Idiomatic and Figurative Language:** Non-literal expressions resist rule-based interpretation.
- **Resource Scarcity:** Limited availability of annotated corpora for many languages.
- **Domain Adaptation:** Decline in model accuracy when applied outside the domain of training data.

These challenges continue to motivate ongoing research in computational linguistics and machine learning.

Applications of Natural Language Processing Across Industries

Introduction

Natural Language Processing has evolved from a theoretical domain of computational linguistics into a pervasive technology integrated across multiple sectors.

Its capacity to extract meaning, recognise intent, and generate coherent language has enabled a diverse set of industrial applications.

From search engines and healthcare systems to legal analytics and voice assistants, NLP forms the foundation of numerous modern intelligent systems.

Role of NLP in Practical Systems

NLP systems operate by analysing linguistic input — text or speech — to identify structure and meaning.

They employ a combination of **syntactic, semantic, and pragmatic** analysis to achieve human-like comprehension.

This capability allows organisations to automate tasks that previously required human linguistic interpretation, such as reading, summarising, or classifying large volumes of textual data.

The subsequent sections describe specific domains where NLP plays a transformative role.

Major Application Areas

Information Retrieval and Search Engines

One of the earliest and most widespread applications of NLP lies in **search technology**.

When a user types a query such as *"nearest pharmacy open now"*, the system does not merely match keywords; it interprets **intent**, **temporal context**, and **entity type** (a place, its operational hours, and proximity).

Techniques such as **query parsing**, **named entity recognition**, and **semantic search** enable the system to provide relevant and precise results.

Modern search engines such as Google and Bing employ large-scale language models that capture context and meaning beyond literal word matching.

E-commerce and Recommendation Systems

In the e-commerce domain, NLP facilitates product discovery, sentiment analysis, and personalised recommendations.

For instance, when a customer searches for *"phone with a good camera"*, the system analyses not only product descriptions but also user-generated content such as reviews and feedback.

Through **synonym detection**, **aspect-based sentiment analysis**, and **semantic similarity**, the system recognises equivalence between expressions such as *"excellent camera"* and *"great picture quality"*.

This linguistic intelligence ensures that recommendations align closely with user intent and experience.

Customer Service and Conversational Agents

Automated chatbots and virtual assistants deployed by banks, airlines, and retail companies rely fundamentally on NLP.

These systems interpret user queries expressed in natural language and produce contextually appropriate responses.

A key challenge in such applications lies in **word sense disambiguation**.

For example, the term *"charge"* may denote a payment, an accusation, or the act of powering a device.

The chatbot must infer the correct sense based on conversational context, often supported by intent classification models and dialogue management frameworks.

Healthcare and Clinical Text Analysis

In healthcare, NLP assists in structuring and interpreting unstructured medical text such as physician notes, pathology reports, and discharge summaries.

By identifying clinically relevant entities — for example, *"shortness of breath"*, *"chest pain"*, or *"hypertension"* — NLP systems support the extraction of patient conditions and treatment patterns.

Such systems employ **named entity recognition (NER)**, **relation extraction**, and **text summarisation** to aid clinical decision-making, epidemiological studies, and health record digitisation.

Machine Translation

Automated translation systems are among the most visible applications of NLP.

Systems such as **Google Translate** convert text from one natural language to another by capturing grammatical, semantic, and contextual equivalence.

Translation requires resolution of **ambiguity** and preservation of **naturalness**, ensuring that output sentences sound fluent rather than literal.

Contemporary translation engines employ **neural sequence-to-sequence architectures**, often enhanced by **attention mechanisms** and **transformer models**, to achieve high-quality translation performance across numerous languages.

Voice-Based Personal Assistants

Voice-enabled devices such as **Siri**, **Alexa**, and **Google Assistant** integrate multiple NLP components:

- **Automatic Speech Recognition (ASR)** to convert spoken language into text,
- **Natural Language Understanding (NLU)** to interpret user intent, and
- **Natural Language Generation (NLG)** to formulate appropriate verbal responses.

These assistants operate as multi-turn dialogue systems capable of maintaining context over interactions.

For example, when a user says, *"Remind me to call my mother at six"*, the system recognises entities (*person* and *time*), intent (*set reminder*), and context (*personal task management*), demonstrating end-to-end NLP integration.

Social Media and Sentiment Analysis

In social media analytics, NLP enables large-scale monitoring of public opinion and emotional tone.

Sentiment analysis algorithms classify textual content into categories such as *positive*, *negative*, or *neutral*.

By applying **lexicon-based** and **machine learning–based** methods, organisations assess consumer satisfaction, brand perception, and emerging trends.

Beyond simple polarity detection, advanced models capture **sarcasm**, **intensity**, and **domain-specific sentiment** variations.

Legal, Financial, and Policy Domains

In the legal and financial sectors, NLP supports the automated analysis of extensive textual corpora including contracts, judgements, and policy documents.

Tasks such as **clause extraction**, **named entity linking**, and **semantic similarity detection** expedite due diligence and compliance checks.

In government and policy research, NLP facilitates the analysis of parliamentary debates, legislative drafts, and public consultations to uncover patterns and sentiments in public discourse.

Significance

The ubiquity of NLP across industries underscores its role as a **general-purpose enabling technology**.

By transforming unstructured linguistic data into actionable insights, NLP contributes to automation, personalisation, and informed decision-making at scale.

Furthermore, the integration of **deep learning** and **large language models** has accelerated this impact, enabling real-time, context-aware, and multilingual processing capabilities previously unattainable through rule-based systems.

Linguistic Levels in Natural Language Processing

Introduction

Language is a complex, multi-layered system comprising sounds, words, structures, and meanings.

For computational models to process language effectively, they must analyse it at several distinct but interdependent linguistic levels.

Each level contributes a different aspect of understanding — from recognising the smallest sound unit to interpreting contextual meaning.

Collectively, these levels form the foundation of **linguistic analysis in Natural Language Processing (NLP)**.

Phonology

Phonology deals with the organisation of sounds in a language.

In speech-based NLP applications such as **Automatic Speech Recognition (ASR)** and **Text-to-Speech (TTS)** systems, phonological knowledge plays a critical role in mapping between written and spoken forms.

For instance, in English, the words *"knight"* and *"night"* are **homophones** — they sound identical but have different spellings and meanings.

Phonological processing helps systems recognise such equivalences and handle challenges such as **homonyms, accents, and pronunciation variations**.

Computational phonology involves creating **phonetic dictionaries**, developing **grapheme-to-phoneme conversion models**, and using **acoustic models** that account for variations in human speech.

This level forms the foundation for speech-enabled NLP tasks.

Morphology

Morphology is the study of the smallest meaningful units of language known as **morphemes**.

Morphemes can be of two types:

- **Free morphemes**, which can stand alone as words (*book*, *kind*, *play*).
- **Bound morphemes**, which must attach to other morphemes (*un-*, *-ness*, *-ed*).

Morphological analysis is essential for identifying how words are formed and how their internal components contribute to meaning.

For example, in the word *"unhappiness"*:

- Prefix: *un-* → denotes negation.
- Root: *happy* → carries the core meaning.
- Suffix: *-ness* → converts the adjective into a noun.

In NLP, morphological analysis supports:

- **Stemming and lemmatisation**, which reduce word variants to a common root form.
- **Part-of-speech tagging**, where morphological endings (e.g., "-ed", "-ing") help determine grammatical category.
- **Vocabulary reduction**, enhancing model efficiency by grouping related word forms.

Syntax

Syntax refers to the structural arrangement of words within a sentence.

It governs how words combine to form phrases and how phrases combine to form sentences according to grammatical rules.

Syntactic analysis helps machines determine relationships such as **subject–verb–object** and identify grammatical dependencies.

For example:

- "The dog chased the cat." → Subject (*dog*), Verb (*chased*), Object (*cat*).
- "The cat chased the dog." → Same words, different meaning due to structural rearrangement.

Computational syntax forms the basis for:

- **Parsing algorithms** (e.g., constituency and dependency parsing).
- **Grammar checking and correction systems**.
- **Machine translation**, where accurate syntactic mapping is critical for preserving meaning across languages.

Syntactic rules are often represented through **context-free grammars (CFGs)**, which specify how sentences can be derived from smaller linguistic units.

Semantics

Semantics addresses the meaning of words and how these meanings combine in sentences.

It aims to represent linguistic expressions in a way that computers can reason about them.

Two primary aspects of semantics are:

1. **Lexical Semantics:** Deals with the meaning of individual words and their relationships (synonymy, antonymy, hyponymy).
    a. Example: *"Big"* and *"large"* are synonyms; *"hot"* and *"cold"* are antonyms.
2. **Compositional Semantics:** Concerns how the meanings of individual words combine to form the meaning of a larger unit (e.g., a sentence).

Semantic analysis in NLP enables:

- **Word Sense Disambiguation (WSD):** Determining the correct meaning of a word in context (e.g., *"bat"* = animal or sports equipment).
- **Semantic Role Labelling:** Identifying roles such as agent, object, and instrument within a sentence.
- **Text Similarity and Entailment:** Determining whether one sentence implies another.

Effective semantic modelling is crucial for applications such as **machine translation**, **information retrieval**, and **question answering**.

Pragmatics

Pragmatics extends beyond literal meaning to consider speaker intention, conversational context, and shared world knowledge.

It explains how meaning is shaped by **social and situational context**.

For instance:

- *"Can you open the door?"* is not a question about ability but a polite directive.

- *"It's cold in here."* often implies a request to close a window rather than a factual statement.

Computational pragmatics is central to **dialogue systems**, **chatbots**, and **virtual assistants**, where systems must interpret indirect requests, politeness strategies, or conversational implicatures.

Techniques such as **intent detection**, **context tracking**, and **discourse modelling** enable systems to capture this higher level of meaning.

Interdependence of Linguistic Levels

The five linguistic levels are interdependent rather than sequentially isolated.

For example, successful **semantic interpretation** depends upon correct **syntactic parsing**; **morphological analysis** informs both **syntax** and **semantics**; and **pragmatic understanding** requires inputs from all prior levels.

Therefore, effective NLP systems integrate these layers into a **hierarchical processing architecture**, ensuring that meaning is consistently derived from structure.

The Natural Language Processing Pipeline

Introduction

Having understood the linguistic foundations of language processing, it becomes essential to examine how these concepts are operationalised within computational systems.

The **Natural Language Processing (NLP) pipeline** refers to the structured sequence of stages through which raw language data is transformed into meaningful, actionable information.

Each stage performs a distinct function — from text acquisition and cleaning to feature representation and model deployment — together enabling machines to analyse and generate human language effectively.

Overview of the Pipeline

An NLP pipeline typically comprises the following six stages:

1. **Data Collection**
2. **Data Cleaning**
3. **Text Preprocessing**
4. **Feature Extraction**
5. **Model Building**
6. **Evaluation and Deployment**

These stages are modular but interdependent; the accuracy and efficiency of later stages depend heavily on the precision and quality of earlier ones.

A schematic overview is shown below:

Raw Text → Cleaning → Preprocessing → Feature Representation → Modelling → Deployment

Data Collection

The first step in any NLP workflow involves the acquisition of language data.

The quality and representativeness of the collected data directly influence the performance of subsequent models.

Sources of Data

Language data may be obtained from:

- **Structured repositories:** such as digital libraries, news archives, or annotated corpora.
- **Unstructured text sources:** such as blogs, social media posts, or transcribed conversations.
- **Speech recordings:** later converted to text using Automatic Speech Recognition (ASR) systems.

Considerations

- **Ethical compliance:** Data must be collected in accordance with privacy, copyright, and consent guidelines.
- **Domain relevance:** The corpus should reflect the linguistic patterns of the intended application (e.g., medical, legal, or conversational domains).
- **Diversity:** A balanced dataset ensures robustness across dialects, styles, and registers.

Data Cleaning

Raw text data is typically noisy and inconsistent.

The purpose of data cleaning is to remove extraneous artefacts that may hinder accurate processing.

Common Cleaning Operations

1. **Removal of non-linguistic elements:** such as HTML tags, URLs, emojis, and special characters.
2. **Standardisation of text format:** including lowercasing (when appropriate), normalising spacing, and correcting encoding issues.
3. **Spelling correction:** particularly in user-generated or transcribed data, using methods such as **edit distance** or **noisy channel models**.
4. **Handling missing or corrupted data:** by filtering or imputing suitable replacements.

The outcome of this stage is a **clean and consistent text corpus**, ready for systematic linguistic analysis.

Text Preprocessing

Preprocessing prepares textual input for computational modelling by decomposing it into analyzable units and simplifying linguistic variation.

It ensures that the machine learning model receives text in a uniform, structured form.

Key Preprocessing Steps

1. **Tokenisation:**

Segmentation of text into individual units or *tokens* — typically words, phrases, or sentences.

   a. Example: “Natural Language Processing is fascinating.” → [Natural, Language, Processing, is, fascinating]

2. **Stopword Removal:**

Elimination of high-frequency function words (e.g., *the*, *is*, *at*) that carry limited semantic value for many analytical tasks.

   a. However, care must be taken not to remove meaningful negatives such as *not* or *never*, which can alter sentiment.

3. **Normalization:**
   a. Standardising forms such as expanding contractions (“don’t” → “do not”), removing diacritics, and converting to lowercase.
   b. Special attention is needed when case conveys meaning (e.g., *Apple* the company vs. *apple* the fruit).
4. **Stemming and Lemmatization:**
   a. **Stemming** reduces words to their base stem by truncation (e.g., *running* → *run*).
   b. **Lemmatization** uses morphological and lexical analysis to derive the true base form (e.g., *better* → *good*).
   c. Lemmatization is more accurate, while stemming is computationally faster.

Together, these steps convert heterogeneous natural text into structured, analyzable input for computational models.

Feature Extraction

Once text is preprocessed, it must be represented numerically, as machine learning algorithms operate on quantitative features.

Feature extraction refers to this conversion of linguistic data into mathematical form while preserving essential semantic information.

Traditional Representations

1. **Bag-of-Words (BoW):**

Represents documents as unordered collections of words, disregarding grammar and order. Each word’s frequency is stored as a feature.

   a. Advantage: Simple and interpretable.
   b. Limitation: Ignores contextual meaning and word relationships.

2. **Term Frequency–Inverse Document Frequency (TF–IDF):**

Weighs words according to their importance by balancing **local frequency** (term occurrence within a document) against **global rarity** (inverse frequency across the corpus).

This reduces the influence of common but uninformative words.

Contextual Representations

Traditional models treat words independently, which limits semantic understanding.

Modern NLP instead uses **distributed representations** or **word embeddings**, where words are represented as dense vectors in a continuous space.

- **Word2Vec** and **GloVe** capture semantic similarity by learning co-occurrence patterns.

For instance, the vectors for *king* and *queen* are closely aligned, and the relation *king – man + woman ≈ queen* emerges naturally.

- These embeddings enable algorithms to infer meaning based on proximity in the vector space.

Thus, feature extraction bridges the gap between textual symbols and numerical computation.

Model Building

After text has been preprocessed and converted into numerical representations, the next step involves constructing a computational model capable of identifying patterns, relationships, or predictive signals within the data.

Model building in NLP refers to the process of **training algorithms** to learn from linguistic data and perform tasks such as classification, clustering, or sequence labelling.

A. Classical Machine Learning Approaches

Before the advent of deep learning, most NLP tasks were successfully handled through **statistical and probabilistic models**.

These models rely on **feature-based representations** such as Bag-of-Words (BoW) or TF–IDF and require well-engineered linguistic features.

Logistic Regression

Logistic Regression models the probability that a given document belongs to a particular class by applying a **sigmoid (logistic) function** to a linear combination of its features.

It is widely used for tasks such as sentiment analysis and topic categorisation.

The model learns **weights** for each feature, indicating the strength and direction of its influence on the prediction.

Regularisation terms (L1 or L2) are used to prevent overfitting by penalising extreme weight values.

Hidden Markov Models (HMMs)

For sequential tasks — such as **Part-of-Speech (POS) tagging**, **Named Entity Recognition (NER)**, or **speech recognition** — **Hidden Markov Models** capture dependencies between consecutive words or tags.

An HMM treats the observed words as emissions from hidden states (e.g., grammatical categories) and learns transition probabilities between those states.

B. Workflow for Model Building

Regardless of the algorithm employed, model construction follows a systematic process:

1. **Feature Selection:** Identify and select the most informative linguistic features (e.g., word frequencies, n-grams, POS tags).
2. **Training:** Fit the model to a training corpus by optimising its parameters or probability distributions.
3. **Validation:** Evaluate performance on a held-out validation set to tune hyperparameters and prevent overfitting.
4. **Testing:** Assess the generalisation of the final model on unseen data.
5. **Interpretation:** Analyse feature importance or model coefficients to understand linguistic patterns captured by the model.

C. Transition to Deep Learning

While classical methods remain foundational and interpretable, their reliance on manual feature engineering limits scalability.

Recent advances in **deep learning** — particularly models such as **RNNs, CNNs, and Transformers** — automate feature extraction by learning hierarchical representations directly from raw text.

These architectures will be studied in detail in subsequent modules.

For now, it is sufficient to understand that **modern NLP integrates both paradigms**:

statistical interpretability from classical models and representational richness from neural models.

Evaluation and Deployment

Once a model is trained, it must be systematically evaluated to determine its **effectiveness, generalisability, and reliability** before deployment.

Evaluation ensures that the system not only fits the training data but also performs accurately on unseen inputs — a core requirement for any practical NLP application.

A. Model Evaluation Principles

Evaluation in NLP is grounded on two complementary goals:

1. **Measuring Predictive Performance:** How accurately does the model predict or classify linguistic inputs?
2. **Understanding Model Behaviour:** Why does the model perform the way it does, and under what conditions does it fail?

Both aspects are crucial for building trustworthy systems.

B. Quantitative Evaluation Metrics

Depending on the nature of the NLP task, different metrics are applied:

Metric	Definition / Purpose	Common Use Cases

Accuracy	Ratio of correct predictions to total predictions.	Overall classification tasks.
Precision	Proportion of predicted positives that are actually correct.	Spam detection, information extraction.
Recall	Proportion of actual positives correctly identified by the model.	Medical or legal text retrieval, where missing a relevant case is costly.
F1-Score	Harmonic mean of precision and recall; balances the two.	Tasks requiring equal emphasis on both correctness and completeness.
Confusion Matrix	Tabular summary of true vs. predicted classes.	Diagnostic tool for classification models.
Perplexity	Measures how well a probability model predicts a sample.	Language modelling and sequence prediction.
BLEU / ROUGE Scores	Compare generated text with reference outputs.	Machine translation and summarisation.

In academic and industrial practice, multiple metrics are used simultaneously to provide a holistic assessment.

C. Qualitative Evaluation and Error Analysis

Quantitative scores alone may obscure underlying model weaknesses.

Hence, **qualitative evaluation** — examining model outputs, misclassifications, or edge cases — provides deeper insight.

1. **Error Categorisation:** Identify whether errors arise from preprocessing, feature extraction, or modelling assumptions.
2. **Confusion Pattern Analysis:** Observe which classes are frequently mislabelled (e.g., *positive* vs. *neutral* sentiment).
3. **Lexical and Contextual Failures:** Examine where models misinterpret idioms, negations, or domain-specific terminology.
4. **Human Evaluation:** In tasks such as summarisation or dialogue generation, human judges assess fluency, coherence, and relevance.

Such analysis informs iterative improvement of both data and algorithms.

D. Overfitting, Underfitting, and Validation

- **Overfitting:** The model learns training data too precisely, capturing noise rather than general patterns. It performs well on training data but poorly on unseen data.
    - *Mitigation strategies:* Regularisation, cross-validation, dropout (for neural models), and increasing data diversity.
- **Underfitting:** The model is too simple to capture underlying relationships.
    - *Mitigation strategies:* Feature enrichment or using more expressive models.

Cross-validation, particularly *k-fold validation*, is widely used to assess generalisability across multiple data subsets.

E. Model Interpretability and Fairness

An increasingly important aspect of evaluation involves ensuring that NLP systems are **transparent, unbiased, and ethically sound**.

Analysts should examine:

- Whether features disproportionately affect certain classes or demographic groups.
- How model decisions can be explained in human-understandable terms (e.g., feature weights, SHAP or LIME explanations).
- Whether data bias may propagate into predictions.

Robust evaluation thus extends beyond accuracy to accountability.

F. Deployment and Continuous Monitoring

Once a model satisfies evaluation criteria, it is integrated into operational environments such as search engines, chatbots, or recommendation systems.

However, deployment marks the beginning — not the end — of the evaluation process.

- **Monitoring:** Continuous tracking of model performance over time ensures stability amid changing language trends (e.g., emerging slang, new entities).
- **Feedback Loops:** User interactions can be collected to fine-tune and retrain the model periodically.
- **Scalability and Efficiency:** Deployed systems must balance accuracy with latency and computational cost.

Sustained performance in deployment requires **active maintenance** and **responsible data governance**.

Rule-Based and Statistical Approaches to Natural Language Processing

Introduction

Approaches to Natural Language Processing have evolved from explicit, rule-driven methods to data-driven statistical models.

Both paradigms aim to enable computers to interpret and generate human language, but they differ in how linguistic knowledge is represented and applied.

Understanding these two foundational approaches provides insight into the progression of NLP systems from handcrafted logic to machine-learned intelligence.

Rule-Based Approach

The **rule-based approach** relies on predefined grammatical, lexical, and semantic rules created by linguists or domain experts.

Each rule captures a known pattern of language structure or meaning, allowing the system to process text deterministically.

Key Features

- Built upon **explicit linguistic knowledge** such as grammars, dictionaries, and pattern rules.
- Operates through **if–then logic** or formal grammar frameworks.
- Produces **consistent and explainable** outputs.

Example

A simple syntactic rule may state:

Noun Phrase → Determiner + Adjective (optional) + Noun

This enables identification of phrases like *"the red car"* or *"a tall building."*

Strengths

- High **precision** in restricted domains where linguistic patterns are well understood.
- **Transparency**: decisions are traceable to human-authored rules.
- Useful when **data is limited** or expert knowledge dominates (e.g., grammar checkers, domain-specific parsers).

Limitations

- **Time-intensive** to build and maintain large rule sets.
- Struggles with **ambiguity, idioms, and exceptions**.
- Lacks **scalability** to new domains or unseen language forms.

Statistical (Data-Driven) Approach

The **statistical approach** emerged as large text corpora and computational resources became available.

Instead of manually coding linguistic rules, models **learn patterns from data** by estimating probabilities or parameters that best explain observed examples.

Core Idea

Language exhibits statistical regularities — certain words, sequences, or structures occur more frequently than others.

By analysing these patterns, algorithms can predict or classify new text without explicit human rules.

Typical Workflow

1. **Collect and label** data (e.g., sentences with part-of-speech tags or sentiment labels).
2. **Extract features** such as word frequencies or TF–IDF scores.
3. **Train a model** (e.g., Naïve Bayes, Logistic Regression, or Support Vector Machine) to learn associations between features and outcomes.
4. **Test** on unseen data to evaluate generalisation.

Applications

- **Spam filtering**: learns word patterns common in spam messages.
- **Sentiment analysis**: classifies reviews as positive or negative based on learned word weights.
- **Information retrieval**: ranks documents by statistical relevance to a query.

Strengths

- Automatically **adapts** to new data and domains.
- Handles **noise and variation** better than rigid rule systems.
- Scales efficiently with larger datasets.

Limitations

- Requires **substantial labelled data** for effective training.
- **Less interpretable** than rule-based systems; difficult to trace individual decisions.
- May **inherit bias** or error patterns from training data.

Comparison of Approaches

Aspect	Rule-Based NLP	Statistical NLP

Knowledge Source	Linguistic expertise and handcrafted rules.	Empirical data and statistical inference.
Adaptability	Low – must modify rules manually.	High – retrain with new data.
Interpretability	Fully explainable.	Often opaque or abstract.
Data Requirement	Minimal.	Large annotated corpora.
Best suited for	Well-defined, controlled language tasks.	Large-scale, diverse language tasks.

Modern systems often **combine** both: rules provide structure and constraints, while statistical models handle variability and uncertainty.

Applications of Rule-Based and Statistical NLP

Introduction

Rule-based and statistical approaches, though conceptually different, have both found widespread use in real-world Natural Language Processing systems.

Their practical applications demonstrate how each paradigm suits specific contexts — rule-based systems excel where structure and precision are required, while statistical systems dominate tasks involving variability and scale.

In many cases, modern NLP solutions integrate both methods to achieve reliability and flexibility.

Rule-Based NLP Applications

A. Grammar and Spelling Checkers

Rule-based systems form the backbone of many traditional **grammar correction tools**.

They employ predefined syntactic and morphological rules to identify common language errors such as subject–verb disagreement or incorrect article usage.

For example:

Rule: *If a singular noun follows, use "is" instead of "are."*

Input: "She are going" → Correction: "She is going."

Such systems provide **high precision** in formal contexts and are particularly effective for controlled languages.

B. Information Extraction

In domains like **legal**, **medical**, or **financial text processing**, rule-based systems can be designed to detect structured information using patterns or regular expressions.

For instance, a rule might extract all dates from a report or identify patient IDs from a record.

Because domain-specific texts often follow fixed formats, rule-based methods ensure reliability and interpretability.

C. Dialogue Systems and Chatbots (Early Models)

Before the advent of statistical learning, early conversational agents like **ELIZA (1966)** and **PARRY (1972)** relied entirely on rule-based pattern matching.

These systems used simple keyword-based templates to simulate understanding — for example:

User: "I feel sad."

System: "Why do you feel sad?"

While limited in intelligence, such systems laid the foundation for human–computer dialogue.

Statistical NLP Applications

A. Machine Translation

Statistical Machine Translation (SMT) systems, such as earlier versions of **Google Translate**, learn translation probabilities from large bilingual corpora.

They automatically discover phrase correspondences and generate the most probable target-language output.

For example, if "good morning" frequently aligns with "bonjour" in training data, the model learns this mapping without explicit rules.

This approach enabled large-scale translation across multiple languages.

B. Sentiment Analysis

In **opinion mining** and **review analysis**, statistical models learn to associate linguistic features (e.g., words, n-grams, emojis) with sentiment categories like *positive* or *negative*.

A Logistic Regression or Naïve Bayes model, for instance, learns that words such as *great* or *excellent* often indicate positive sentiment, while *poor* or *terrible* indicate the opposite.

Such models are widely deployed in e-commerce and social media monitoring.

C. Text Classification and Spam Detection

In email filtering, statistical classifiers use word frequencies and contextual probabilities to distinguish between *spam* and *non-spam* messages.

The system automatically adapts to new patterns (e.g., emerging spam keywords) by retraining on updated datasets, offering scalability beyond what static rules could achieve.

D. Speech Recognition and Tagging

Tasks like **Part-of-Speech tagging** and **Automatic Speech Recognition (ASR)** rely on probabilistic models that learn sequential patterns in language.

For example, given a sequence of words, a statistical model predicts the most likely tag (noun, verb, etc.) or corresponding sound pattern.

These models demonstrate the strength of data-driven approaches in handling uncertainty and variation.

Hybrid and Modern Implementations

Contemporary NLP systems frequently combine **rule-based precision** with **statistical adaptability**.

For instance:

- A **chatbot** may use rules for intent detection (e.g., recognising “book flight”) and a statistical model for language generation.
- A **grammar checker** might employ statistical ranking to prioritise the most likely corrections among multiple rule-based suggestions.

This hybridisation achieves both interpretability and robustness, reflecting the complementary nature of both paradigms.

Lexical Processing

Introduction

Lexical Processing represents the **first structured stage** in the Natural Language Processing (NLP) pipeline.

It focuses on processing **words and their linguistic forms**, ensuring consistency and interpretability before syntactic or semantic analysis.

The goal is to convert unstructured text into linguistically meaningful and standardised tokens.

Lexical processing typically includes:

1. **Text Normalisation** – standardising text forms.
2. **Tokenisation** – dividing text into words or segments.
3. **Stopword Removal** – filtering uninformative words.
4. **Morphological Analysis** – identifying roots and affixes.
5. **Spell Correction** – detecting and fixing typographical or contextual errors.
6. **Stemming and Lemmatization** – reducing words to base or dictionary forms.
7. **Lexical Resource Mapping** – linking words to structured linguistic databases such as WordNet or gazetteers.

Each of these operations enhances the quality of text representation for higher levels of NLP.

Text Normalisation

Purpose

Language as it appears “in the wild” is messy.

Texts gathered from social media, emails, or scanned documents often contain **inconsistencies** such as irregular capitalisation, misspellings, mixed encodings, emojis, and formatting noise.

These inconsistencies fragment the vocabulary and mislead computational models.

Text normalisation is the process of converting such heterogeneous text into a **uniform, machine-interpretable form**.

By enforcing consistency, it ensures that semantically identical expressions —

for example *“Covid-19”*, *“covid 19”*, and *“COVID19”* — are recognised as the same token.

Normalisation is therefore a **foundational prerequisite** for tokenisation, feature extraction, and model training.

Key Operations in Normalisation

Operation	Objective	Illustration
Case Conversion	Reduces variation caused by capitalisation.	“Natural Language” → “natural language”
Punctuation & Symbol Handling	Removes or selectively retains symbols. Punctuation like commas or periods may be dropped unless needed for sentence boundaries.	“C++ programming” → keep “++”; “Hello!!!” → “hello”
Number Standardisation	Brings all numeric mentions to a common form or token.	“ten kg” → “10 kg” or “<NUM> kg”
Contraction Expansion	Converts shortened forms to their full equivalents for better parsing.	“don’t” → “do not”; “I’ll” → “I will”
Accent & Diacritic Removal	Removes language-specific marks to ensure uniform encoding.	“résumé” → “resume”; “café” → “cafe”
Whitespace & Encoding Fixes	Eliminates redundant spaces and standardises text encoding (e.g., UTF-8).	“data science” → “data science”

Advanced Normalisation Tasks

1. **Slang and Abbreviation Expansion**

Replace informal or shortened forms with their full equivalents:

“u” → “you”; “btw” → “by the way”.

Particularly useful for social-media text mining.

2. **Unicode and Emoji Handling**

Depending on the application, emojis may either be removed or mapped to textual descriptors

(😊 → "smile").

3. **Text Standardisation Pipelines**

In large-scale NLP systems, these steps are implemented as rule-based or regex pipelines that convert raw input into a canonical, lower-noise version before tokenisation.

Illustrative Example

Input Text:

"I LUV Python !! #CodingIsLife 🩶 2025"

Normalisation Process:

1. Lowercase → "i luv python !! #codingislife 💔 2025"
2. Expand slang → "i love python !! #codingislife 💔 2025"
3. Remove excess punctuation → "i love python #codingislife 💔 2025"
4. Handle emoji and number → "i love python #codingislife heart <NUM>"

Output → "i love python codingislife heart <NUM>"

Why It Matters

- **Reduces Sparsity:** Different spellings or cases no longer appear as separate tokens.
- **Improves Model Generalisation:** Models learn linguistic patterns rather than formatting noise.
- **Enhances Search & Matching:** Keyword-based systems retrieve results regardless of capitalisation or punctuation differences.
- **Facilitates Multilingual Processing:** Consistent encoding allows scripts and diacritics to be handled uniformly across languages.

Tokenisation

Purpose

After text is normalised, the next task is to divide it into **meaningful linguistic units** known as *tokens*.

A token usually corresponds to a **word**, **subword**, or **sentence**, depending on the level of analysis.

Tokenisation is crucial because almost every NLP algorithm—from frequency counts to neural embeddings—operates on tokens rather than raw character streams.

It acts as the **bridge** between raw text and structured linguistic data.

Definition

Tokenisation is the process of segmenting continuous text into smaller components (*tokens*) that carry meaning and can be independently analysed.

Mathematically, tokenisation may be viewed as a mapping

$$T: S \rightarrow \{t_1, t_2, \ldots, t_n\}$$

where S is a text string and t_i are the extracted tokens.

Levels of Tokenisation

Level	Description	Example
Word-Level	Splits on spaces and punctuation; the most common form.	"Language models are powerful." → [language, models, are, powerful]
Subword-Level	Breaks complex or unseen words into smaller meaningful parts (useful for morphologically rich languages or neural models).	"unhappiness" → [un, happy, ness]
Sentence-Level	Divides paragraphs into sentences based on punctuation and syntactic cues.	"It rained. Roads flooded." → [Sentence 1, Sentence 2]
Character-Level	Treats each character as a token (rare; used in handwriting or low-resource NLP).	"cat" → [c, a, t]

Approaches to Tokenisation

1. **Rule-Based / Regular-Expression Tokenisers**
    a. Use predefined delimiters such as spaces, commas, or punctuation.
    b. Example: NLTK's TreebankWordTokenizer.
    c. Advantage – Simple and fast; Limitation – Fails with exceptions like *"U.S.A."* or *"e-mail"*.
2. **Statistical Tokenisers**
    a. Learn likely boundaries from data (common in languages without spaces).
    b. Example: Word-boundary prediction using character n-grams or CRFs for Chinese.
3. **Subword Algorithms (Neural-Era)**
    a. Break words into smaller, reusable units based on frequency.
    b. **Byte-Pair Encoding (BPE)** and **WordPiece** (used in BERT) are popular.
    c. Example: "unbelievable" → [un, believ, able].
    d. Benefits: Handles unseen words; reduces vocabulary explosion.

Challenges in Tokenisation

1. **Ambiguous Boundaries** – Some languages (e.g., Chinese, Thai) lack spaces, making segmentation non-trivial.
2. **Multiword Expressions** – "New York City" should remain a single token in entity recognition tasks.

3. **Punctuation Ambiguity** – Periods in abbreviations ("Dr.", "U.S.") or decimals ("3.14") should not split sentences.
4. **Contractions** – "don't" may be one or two tokens depending on task; expanded form ("do", "not") preferred in sentiment analysis.
5. **URLs and Emojis** – Require special handling; often replaced with placeholders such as <URL> or <EMOJI>.

Examples

Example 1 – Word Tokenisation

"Tokenisation in NLP is important!"

→ [Tokenisation, in, NLP, is, important]

Example 2 – Subword Tokenisation

"tokenisation" → [to, ken, isation] (using BPE)

"anti-discrimination" → [anti, dis, crimination]

Example 3 – Sentence Tokenisation

"Mr. John went home. He slept."

→ ["Mr. John went home.", "He slept."]

The period after "Mr." is not treated as a boundary.

Why Tokenisation Matters

- **Vocabulary Control:** Defines the granularity of word representation.
- **Downstream Efficiency:** Smaller, consistent tokens improve model learning.
- **Language Adaptation:** Enables models to handle morphological variation.
- **Interpretability:** Tokens correspond to human-readable linguistic units.

Stopword Removal

Purpose

Not all words in a text contribute equally to its meaning.

Certain **high-frequency function words** — such as *the*, *is*, *at*, *of*, *for*, *to*, and *an* — occur frequently across all kinds of text but carry little discriminative value.

They mainly serve **grammatical functions**, helping sentence construction rather than conveying content.

For most NLP tasks, these words add computational overhead without significantly improving meaning representation.

Stopword removal refers to the process of filtering out such terms so that only **content-bearing words** remain for further analysis.

Definition

Stopwords are words that occur very frequently in a language but contribute minimally to the contextual meaning or discriminative power of a document.

Removing them helps focus on semantically informative content such as nouns, verbs, and adjectives.

Examples

Original Sentence	After Stopword Removal
"The movie was not very good."	"movie not good"
"Data is the new oil of the digital economy."	"Data new oil digital economy"

The key idea is to retain **keywords** that define meaning and drop filler words that repeat across all documents.

Why Remove Stopwords?

1. **Dimensionality Reduction**
    a. Reduces vocabulary size by removing extremely common tokens.
    b. Results in faster computation and smaller vector representations.
2. **Improved Focus on Meaningful Words**
    a. Models learn from words that actually differentiate documents or sentiments.
3. **Reduced Noise in Frequency-Based Models**

   a. In Bag-of-Words (BoW) or TF–IDF representations, common stopwords dominate counts and distort relevance.
4. **Simplified Matching in Search Systems**
   a. In information retrieval, ignoring stopwords ("the", "of", "in") allows better keyword matching.

Challenges and Considerations

While stopword removal is widely used, **it is not universally beneficial**.

Its effectiveness depends on the task, language, and data domain.

A. Task Sensitivity

- In **sentiment analysis**, words such as *"not"*, *"never"*, or *"no"* carry critical polarity information and **must be retained**.

Example:

  - "The movie was not good." → removing *not* changes the meaning completely.
- In **text summarisation or question answering**, stopwords may be useful for syntactic or contextual completeness.

B. Domain Dependence

- Generic lists (like NLTK's English stopwords) may not suit specialised domains.

In biomedical text, for instance, words such as *"patient"*, *"disease"*, or *"study"* may occur very frequently but still hold semantic importance.

Thus, a **domain-specific stopword list** should be curated.

C. Language Dependence

- Stopwords differ across languages (e.g., *le*, *la*, *les* in French; *der*, *die*, *das* in German).

Language-specific lists are necessary to maintain accuracy.

Morphological Analysis

Purpose

After tokenisation and stopword filtering, the next step in lexical processing is to understand **the internal structure of words**.

Every word in a language can often be broken down into smaller meaning-bearing units.

This study of word structure is known as **morphology**, and the computational procedure of analysing it is called **Morphological Analysis**.

The main goal is to identify **roots (stems)** and **affixes (prefixes, suffixes, infixes)** so that all inflected or derived forms of a word can be recognised as related.

Definition

Morphology is the branch of linguistics that studies the internal structure of words and how they are formed from smaller meaningful elements called **morphemes**.

Morphological Analysis in NLP refers to the process of segmenting a word into its constituent morphemes and determining their grammatical roles.

Morphemes and Their Types

A **morpheme** is the smallest unit of meaning or grammatical function in a language.

Each word can be formed from one or more morphemes.

Type	Definition	Example
Free Morpheme	Can stand alone as a word.	*book*, *kind*, *happy*
Bound Morpheme	Cannot stand alone; must attach to a root.	*un-*, *-ness*, *-ed*
Root / Stem	Core part carrying primary meaning.	*play* in *playing*
Prefix	Added before the root to alter meaning.	*un-* in *unhappy*
Suffix	Added after the root to mark tense or form.	*-ing*, *-ed*, *-ness*
Infix / Circumfix	Inserted or wrapped around the root (less common in English; frequent in Semitic or Asian languages).	*um* in Tagalog *s-um-ulat* ("write")

Examples of Word Formation

Word	Morphological Breakdown	Meaning Change
Unhappiness	un- + happy + -ness	"Not happy" → "state of not being happy"
Replaying	re- + play + -ing	"Play again"
Kindness	kind + -ness	"State or quality of being kind"
Misunderstanding	mis- + understand + -ing	"Act of wrongly understanding"

Functions of Morphological Analysis in NLP

1. **Vocabulary Normalisation**

   a. Groups together word variants (*plays*, *played*, *playing* → *play*).
   b. Reduces data sparsity in feature-based models.
2. **Part-of-Speech Tagging Support**
   a. Identifies grammatical clues such as tense (*walked* → past tense), number (*cats* → plural noun), or degree (*happier* → comparative adjective).
3. **Semantic Clarity**
   a. Helps systems understand relationships between words with shared roots (*create*, *creator*, *creation*).
4. **Downstream Utility**
   a. Aids in stemming, lemmatisation, and spell correction — ensuring consistent handling of word forms.

Computational Perspective

In computational linguistics, morphological analysis can be performed using **two primary approaches**:

A. Rule-Based Morphological Analysers

- Use manually crafted grammar rules or morphological dictionaries.
- Suitable for morphologically rich languages (e.g., Hindi, Arabic, Turkish).
- Example rule:

If word ends with "-ing" and root exists in dictionary → classify as *present participle*.

Pros: Linguistically accurate.

Cons: Labour-intensive and language-specific.

B. Statistical or Machine-Learned Analysers

- Use annotated corpora to learn affix patterns and segmentation.
- Example: A model trained on word-tag pairs learns that "-ed" indicates past tense with high probability.
- Probabilistic models (like Conditional Random Fields) or neural sequence models can identify morpheme boundaries from large datasets.

Pros: Scalable, language-adaptable.

Cons: Requires labelled data.

Example Workflow

Input Word: *Unbelievable*

1. Identify prefix: *un-*
2. Identify root: *believe*
3. Identify suffix: *-able*

Output: [Prefix: Negation] + [Root: believe] + [Suffix: expressing ability]

Meaning: "Not able to be believed."

Applications of Morphological Analysis

Application	How Morphology Helps
Information Retrieval	Groups inflected forms under one term. Searching "run" also retrieves "running", "ran".
Machine Translation	Preserves tense and agreement when translating between morphologically different languages.
Speech Recognition	Identifies variants of words pronounced differently (e.g., *played*, *playing*).
Text-to-Speech Systems	Correct pronunciation of derived forms by identifying root and affix patterns.

Spell Correction - Edit (Levenshtein) Distance

Concept

In spelling correction, we need a way to measure **how different** two words are.

The **Edit Distance**, also known as the **Levenshtein Distance**, is one of the most fundamental metrics for this purpose.

It quantifies the **minimum number of single-character operations** required to transform one string into another.

The allowed operations are:

1. **Insertion (I):** Add one character.
2. **Deletion (D):** Remove one character.
3. **Substitution (S):** Replace one character with another.

For example:

flaw → lawn

- Delete "f" (1 operation)
- Insert "n" at the end (1 operation)

→ **Edit Distance = 2**

Formal Definition

Let two strings be $a = a_1 a_2 \ldots a_m$ and $b = b_1 b_2 \ldots b_n$.

Define $M(i, j)$ as the edit distance between the first i characters of a and the first j characters of b.

The recursive formulation is:

$$M(i,j) = \begin{cases} 0, & i = 0; j = 0 \\ i, & j = 0 \\ j, & i = 0 \\ \min \begin{pmatrix} M[i,j-1] \\ M[i-1,j] \\ M[i-1,j-1] \end{pmatrix} + 1, & a[i] \neq b[j] \\ M[i-1,j-1], & a[i] = b[j] \end{cases}$$

Example

Let's compute the Levenshtein distance between:

$a =$ kitten

$b =$ sitting

Operation	Step	Word
Substitute **k** → **s**	1	*sitten*
Substitute e → i	2	*sittin*
Insert **g**	3	*sitting*

$$EditDist(kitten, sitting) = 3$$

Example Table

The algorithm fills a matrix $M[i,j]$ of size $(m+1) \times (n+1)$, where each cell represents the cost of aligning prefixes of the two strings.

Example (simplified for *flaw* → *lawn*):

	#	l	a	w	n
#	0	1	2	3	4
f	1	1	2	3	4

l	2	1	2	3	4
a	3	2	1	2	3
w	4	3	2	1	2

The final cell gives the distance (2).

Probabilistic Spell Correction – The Noisy Channel Model

Motivation

While edit distance measures how *similar* two words are, it does not account for **word likelihood** or **context**.

For example:

speling → could be *spelling* or *spieling*

Both are one edit apart, but *spelling* is far more common.

Hence, we need a way to combine:

- **How likely it is that a word was mistyped** (error model), and
- **How common or probable the correct word is** in the language (language model).

This leads us to the **Noisy Channel Model (NCM)** — a probabilistic framework for choosing the *most likely intended word* given a misspelling.

The Core Idea

We assume the user wanted to type a word w, but due to noise (keyboard slips, OCR errors, etc.), we observed a corrupted form x.

Our goal is to infer the most probable true word w^* given the observed word x.

$$w^* = \underset{w \in V}{\arg max} P(w/x)$$

Applying **Bayes' Theorem**:

$$P(w/x) = \frac{P(x/w).P(w)}{P(x)}$$

Since $P(x)$ is constant for all candidates, we can ignore it for ranking:

$$w^* = \arg\max_{w \in V} P(x/w).P(w)$$

Interpreting the Two Components

In the Noisy Channel Model, we estimate the probability of the intended word (w) given the observed misspelling (x) as:

$$w* = \arg\max P\,(x/w) * P(w)$$

This depends on two key components — the **Language Model** and the **Error Model**. Both contribute complementary information.

Component	Symbol	Description / Meaning	Estimated From	Role in Correction
Language Model	$P(w)$	Prior probability of the word w appearing in natural text. Represents how likely a word is in general usage.	Large text corpora (word frequencies, n-gram models)	Prefers common or linguistically plausible words (e.g., "spelling" > "spieling")
Error Model	$P(x/w)$	Likelihood that intended word w produced misspelling x. Captures typical human or system error patterns.	Confusion matrices of typing/OCR errors.	Prefers corrections consistent with typical error patterns ("ie" ↔ "ei", etc.).

Example

For observed word x = "speling":

Candidate (w)	P(w) (from corpus)	P(x|w)	P(x|w)*P(w)
spelling	0.0012	0.5	**0.0006**
spieling	0.0001	0.5	0.00005
selling	0.0011	0.05	0.000055

Interpretation:

- "spelling" is both frequent (high P(w)) and a plausible misspelling (high P(x|w)).
- "spieling" is rare → low prior probability.
- "selling" is frequent but unlikely typo → low error probability.

Thus:

$$w* = \arg\max \quad P(x \mid w) * P(w)$$

or equivalently, in logarithmic form for stability:

$$w^* = \arg\max \quad [\log P(x/w) + \log P(w)]$$

Building the Model

A. The Language Model $P(w)$

Represents the likelihood of a word's natural occurrence in the language.

It can be estimated from corpus frequencies:

$$P(w) = \frac{count(w)}{\sum_{w'} count(w')}$$

Example: If "receive" appears 1,000 times in a corpus of 10 million words:

$$P(receive) = \frac{1000}{10^7} = 10^{-4}$$

Higher frequency words are more likely intended corrections.

B. The Error Model $P(x/w)$

Represents how likely it is that a true word w produces a misspelled form x.

Derived from confusion matrices of character-level edits:

Error Type	Example	Likelihood P(x/w)
Substitution	e → i	0.3
Deletion	missing a letter	0.2
Insertion	extra character	0.1
Transposition	th → ht	0.15

This model captures common keyboard or typographical patterns.

C. Combined Scoring

Final score combines the two probabilities:

$$Score(w) = \log P(x/w) + \log P(w)$$

Log probabilities are used for numerical stability (addition instead of multiplication).

The candidate with the **highest total score** is selected as the correction.

Implementation Intuition

Algorithm Outline:

1. Generate candidate words $C(x)$ within edit distance ≤ 2.
2. For each candidate w:
   a. Compute $P(x/w)$ from error model.
   b. Compute $P(w)$ from corpus frequency.
3. Return:

$$w^* = \underset{w \in C(x)}{\arg\max} \; P(x/w).P(w)$$

Example

Word	Edit Distance	P(w)	P(x/w)	Score
receive	1	0.002	0.6	0.0012
recipe	2	0.0008	0.3	0.00024
deceive	2	0.0004	0.3	0.00012

Predicted Correction: *receive*

Relationship to Edit Distance

- Edit distance provides the **mechanism** to identify plausible candidates (those within small edit distance).
- The Noisy Channel Model provides the **probabilistic ranking** among those candidates.

Together, they form a robust two-step system:

1. **Candidate Generation:** Using Edit Distance.
2. **Candidate Ranking:** Using the Noisy Channel Model.

Advantages

- Probabilistically sound; accounts for both frequency and error patterns.
- More accurate than pure edit-distance correction.
- Can generalise to contextual and real-word errors.
- Scalable for large dictionaries and corpora.

Limitations

- Requires large corpora for accurate $P(w)$.
- Needs confusion data for realistic $P(x/w)$.

- Cannot handle deep semantic context (e.g., homophones) without additional models.

Probabilistic Spell Correction – The Noisy Channel Model

Purpose

After text has been normalised, tokenised, and corrected for spelling, different inflected or derived forms of the same word may still appear separately.

From a computational perspective, words such as *connect*, *connected*, *connecting*, and *connection* share the same core meaning and should ideally map to a common base form.

Stemming and **Lemmatization** achieve this by reducing words to their **root** or **lemma**, thereby lowering vocabulary size and improving linguistic consistency.

Term	Definition	Example Output

Stemming	Rule-based removal of prefixes/suffixes to obtain an approximate root.	*studies → studi*
Lemmatization	Linguistically informed reduction to canonical dictionary form.	*studies → study*

Both techniques are key for tasks such as information retrieval, text classification, topic modelling, and semantic analysis.

Stemming

Definition

Stemming is a heuristic process that removes or truncates affixes from words using simple pattern-matching rules, without considering grammatical context.

The output (the **stem**) may not always be a valid English word.

Examples:

playing → play *studies → studi*

Porter Stemmer (Rule-Based Example)

The **Porter Algorithm** (1980) applies successive rewrite rules in five stages.

Step	Rule Example	Illustration
1	Remove plural / -ed / -ing endings	“agreed” → “agree”
2	Remove derivational suffixes	“sadness” → “sad”
3	Simplify double suffixes	“relational” → “relate”
4	Remove final -ion / -ous / etc.	“effective” → “effect”
5	Drop terminal -e if redundant	“hopeful” → “hope”

Advantages

- Fast and language-specific.
- Reduces sparsity for search or indexing tasks.

Limitations

- May yield non-words (*studi*).
- Over-stems (*universe*, *university* → *univers*).
- Under-stems (*connect*, *connection*).
- Language-dependent rule creation.

Lemmatization

Lemmatization identifies the **dictionary base form** (*lemma*) of a word using morphological analysis and part-of-speech (POS) information.

Mathematically:

- f : (word, POS) → lemma *

Examples

Word	POS Tag	Lemma
running	Verb	run
studies	Noun	study
better	Adjective	good
was	Verb	be

Unlike stemming, lemmatization distinguishes grammatical roles:

meeting → *meet* (verb) *meeting* → *meeting* (noun)

Advantages

- Produces valid dictionary words.
- Handles irregular inflections (*went* → *go*).
- Context-aware through POS information.
- Ideal for semantic and translation tasks.

Comparison

Aspect	Stemming (Rule-Based)	Lemmatization (Linguistic)

Method	Simple affix stripping	Morphological + POS analysis
Speed	Fast	Slower
Output Validity	Often non-word	Always valid word
Context Awareness	None	High
Accuracy	Moderate	High
Use Case	Search / IR engines	MT, QA, semantic analysis

Illustrative Example

Sentence: *"The boys are playing happily and studied hard."*

Word	Stemmed	Lemma
boys	boy	boy
playing	play	play
happily	happi	happy
studied	studi	study

Applications

- **Search Engines:** Group morphological variants under one query term.
- **Topic Modelling (LDA):** Reduce redundancy for clearer topics.
- **Machine Translation:** Ensure grammatical agreement across languages.
- **Question Answering:** Normalise wording between Q & A pairs.
- **Text Classification:** Lower feature dimensionality.

Part-of-Speech (POS) Tagging

Definition

Part-of-Speech (POS) Tagging is the process of assigning each word in a sentence its grammatical category — such as **noun**, **verb**, **adjective**, or **adverb** — based on both its **definition** and **context**.

It is the first step in **syntactic analysis**, helping computers understand **how words function together** to form meaningful sentences.

Example

Sentence:

The quick brown fox jumps over the lazy dog.

Token	POS Tag
The	DT
quick	JJ
brown	JJ
fox	NN
jumps	VBZ
over	IN
the	DT
lazy	JJ
dog	NN

Legend:

- DT → Determiner
- JJ → Adjective

- NN → Noun (Singular)
- VBZ → Verb, 3rd-person singular
- IN → Preposition

Objective

$$\text{Given a sequence of words } W = w_1, w_2, \ldots, w_n$$

$$\text{determine the most likely sequence of POS tags } T = t_1, t_2, \ldots, t_n$$

Formally:

$$T^* = \arg\max_T \quad P(T/W)$$

This probability can be approached through rules or statistical models, which you'll study in upcoming sections.

Why POS Tagging Matters

Purpose	Explanation
Grammatical Understanding	Identifies sentence structure (subject, verb, object).
Word Sense Disambiguation	Distinguishes meanings (e.g., "book" as a noun vs. verb).
Information Extraction	Helps locate entities and relationships.
Semantic Analysis	Provides input for higher-level understanding.
Downstream NLP Tasks	Essential for translation, summarisation, and question answering.

Challenges in POS Tagging

Challenge	Example / Description
Ambiguity	"can" → may be *noun* or *verb* depending on context.
Unknown Words	Newly coined or domain-specific words.
Multi-word Expressions	"New York" → should be treated as a single noun.
Context Sensitivity	"flies" → noun in *"Time flies fast"*, verb in *"Flies eat fruit."*

Example Tag Sets

Tag Set	Sample Tags	Usage
Penn Treebank	NN, NNS, VB, VBD, JJ, RB, IN, PRP	Standard English tagging scheme.
Universal POS	NOUN, VERB, ADJ, ADV, PRON, DET, ADP, CCONJ, NUM, PART	Simplified and multilingual (used in spaCy / UD).

Rule-Based POS Tagging

Definition

Rule-Based POS Tagging is one of the earliest approaches to assigning parts of speech.

It uses a predefined set of **lexical resources** (like dictionaries or lexicons) and **handcrafted grammatical rules** to determine the most likely POS tag for each word based on its form and neighbouring context.

The system relies on **two main components:**

1. A **Lexicon** – containing words and their possible tags.
2. A **Rule Base** – containing disambiguation rules that select the correct tag using syntactic or morphological clues.

Example: Basic Rule Structure

A rule typically follows a pattern like:

If previous word's tag = "TO" then tag current word as "Verb (VB)".

or

If word ends with "-ly" then tag as "Adverb (RB)".

These deterministic rules capture linguistic regularities that frequently appear in English grammar.

Example Workflow

Let's tag the sentence:

Can you can the can?

1. Lexicon lookup:
    a. "can" → {Noun, Verb, Modal}
2. Apply context rules:
    a. Rule 1: If a word follows a pronoun → tag as **Verb (VB)**
    b. Rule 2: If preceded by "the" → tag as **Noun (NN)**

Resulting tags:

Can (MD) you (PRP) **can (VB)** the (DT) **can (NN)** ?

Rule-Based Tagging Process

Step	Description
1. **Lexical Lookup**	Retrieve possible POS tags from a dictionary.
2. **Morphological Analysis**	Use word endings, prefixes, and inflections (e.g., "-ed", "-ing").
3. **Contextual Rule Application**	Apply grammatical rules based on neighbouring tags.
4. **Disambiguation**	When multiple tags fit, apply highest-priority rule.

Sample Heuristic Rules

Rule Type	Example	Explanation

Suffix Rule	If word ends in "-ing", tag as **Verb (VBG)**	"playing"
Prefix Rule	If word starts with "un-", tag as **Adjective (JJ)**	"unhappy"
Context Rule	If word follows "to", tag as **Verb (VB)**	"to play"
Word Form Rule	If word is capitalised mid-sentence, tag as **Proper Noun (NNP)**	"John"
Lexical Rule	Use dictionary to resolve fixed tags.	"the" → DT, "is" → VBZ

Example: Simple Rule-Based Output

Sentence:

The old man the boats.

Word	Possible Tags	Selected Tag	Rationale
The	DT	DT	Determiner
old	JJ / NN	NN	Functions as noun in this context
man	NN / VB	VB	Verb form selected via rule: "after noun → verb"
the	DT	DT	Determiner
boats	NN	NN	Plural noun

Final Interpretation:

"The old (people) man (operate) the boats."

Advantages

- High interpretability and transparency.
- Effective for small domains or well-defined grammar.
- Easy to update or debug by linguists.
- Requires no large annotated corpora.

Limitations

Aspect	Challenge
Manual Effort	Requires many handcrafted rules.

Scalability	Hard to maintain for large vocabularies or multi-lingual data.
Ambiguity Resolution	May fail when context contradicts multiple rules.
No Probability Handling	Cannot represent uncertainty or frequency information.

Statistical POS Tagging

Definition

Statistical POS Tagging is a **data-driven approach** that assigns grammatical tags to words based on **probabilities learned from annotated text corpora**.

Instead of relying on manually written rules, it learns which tags are most likely for given words — and which tags usually follow each other — based on real usage in language data.

In simple terms:

The tagger looks at how frequently a tag appears for a word, and how tags tend to occur together in sentences, then chooses the **most likely sequence of tags** for a new sentence.

Core Intuition

Each word can have multiple possible tags — for example,

"book" → Noun (*a novel*) or Verb (*to reserve*).

A statistical tagger uses two kinds of probabilities to decide which one fits best:

Type	What It Captures	Example
Tag Probability	$P(tag/previous\ tag)$	How likely one tag follows another.
Word Probability	$P(word/tag)$	How likely a word appears with a specific tag.

The final decision combines both intuitions:

which tag fits the word **and** which tag fits the **sentence context**.

How It Works (Conceptually)

Step	Description
1. **Training**	The model learns probabilities from a tagged corpus.
2. **Prediction**	For each new sentence, it estimates possible tags and their likelihoods.
3. **Sequence Choice**	It selects the tag combination that has the **highest overall probability** of being correct.

In practice, algorithms compute this efficiently (e.g., using methods like Viterbi decoding internally),

but in your course, the focus is on **understanding how probabilities guide the tagging process**, not on the algorithmic details.

Example

Sentence:

Time flies like an arrow.

Possible tags:

- "Time" → Noun or Verb
- "flies" → Noun or Verb
- "like" → Preposition or Verb

Based on learned probabilities:

- *Determiner (DT)* often followed by *Noun (NN)*.
- *Noun (NN)* often followed by *Verb (VBZ)*.

Thus, the most probable tagging sequence is:

Time (NN) flies (VBZ) like (IN) an (DT) arrow (NN)

Why It Works Better Than Rule-Based Tagging

Aspect	Rule-Based	Statistical
Basis	Fixed linguistic rules	Probabilities learned from data
Learning	Manual	Automatic
Adaptability	Limited	Improves with more data
Handling Ambiguity	Deterministic	Chooses most likely option
Scalability	Hard for large corpora	Suitable for big datasets

Advantages

- Learns from examples instead of manual rules.
- Adapts to real-world text usage.
- Resolves ambiguity using context and frequency.
- Works across large vocabularies and multiple domains.

Limitations

Aspect	Challenge
Data Dependency	Requires large tagged corpora for training.
Unknown Words	Performs poorly on unseen or rare words.
Local Context Only	Often considers only nearby tags (limited context).
Interpretability	Less transparent than rule-based systems.

Hidden Markov Model (HMM) Tagging

Definition

A **Hidden Markov Model (HMM)** is a **probabilistic framework** used for sequence labelling tasks such as **Part-of-Speech (POS) Tagging**.

It models language as a sequence of **hidden states** (POS tags) that generate **observable outputs** (words).

The idea is that we don't see the part-of-speech tags directly — they are *hidden* —

but we can infer them based on the *observable sequence of words*.

Conceptual Structure

An HMM for POS tagging consists of:

Component	Represents	Example
Hidden States	POS tags (NN, VB, JJ, etc.)	NN → VB → DT → NN
Observations	The actual words in the sentence	"Dogs bark at night"
Transition Probabilities	How likely one tag follows another	P(VB | NN): verbs often follow nouns
Emission Probabilities	How likely a word is given a tag	P(bark | VB): "bark" is common as verb
Initial Probabilities	Probability of a tag starting a sentence	P(DT) — sentences often begin with determiners

The model predicts the tag sequence that best explains the observed words.

Key Intuition

- Each word depends **on its own tag** (emission).
- Each tag depends **on the previous tag** (transition).
- The tagger combines both patterns to assign the most likely tag sequence.

Tag sequence chosen = $\{the\ one\ with\ highest\ overall\ probability\ of\ transitions\ \times\ emissions\}$

Example

Sentence: *Time flies like an arrow.*

Word	Possible Tags	Most Likely Tag (Based on HMM)
Time	NN / VB	NN
flies	NNS / VBZ	VBZ
like	IN / VB	IN
an	DT	DT
arrow	NN	NN

Resulting sequence:

Time (NN) flies (VBZ) like (IN) an (DT) arrow (NN)

The HMM prefers this tagging because:

- *DT* → *NN* transitions are common.
- *VBZ* (verb) often follows *NN* (noun).
- *IN* → *DT* → *NN* forms a familiar phrase pattern.

Why HMMs Work Well

- Capture **contextual dependencies** between tags.
- Handle **ambiguity** statistically, based on likelihood.
- Learn probabilities automatically from **annotated corpora**.
- Provide a **mathematically grounded** model for sequential text.

Advantages

- Data-driven yet interpretable.
- Performs well on moderate-sized datasets.
- Handles unseen sequences using probability smoothing.
- Foundation for many later sequence models (CRFs, RNNs).

Limitations

Aspect	Challenge
Markov Assumption	Only considers nearby tags (short context).
Emission Independence	Ignores influence of surrounding words.

Sparse Data	Rare tag–word combinations lower performance.
Unknown Words	Unseen tokens hard to predict.

Transformation-Based POS Tagging (Brill Tagger)

Definition

Transformation-Based POS Tagging (TBL), also known as the **Brill Tagger** (named after Eric Brill, 1992), is a **hybrid approach** that combines the strengths of both **rule-based** and **statistical** methods.

It begins with a simple **initial tagging** (often statistical or lexicon-based) and then **learns correction rules** from data to iteratively improve accuracy.

In essence, it is:

"Rule-based learning guided by statistical evidence."

Core Idea

1. Start with a **baseline tagging** — e.g., assign the most frequent tag for each word (like NN for unknown words).
2. **Compare** this tagging to the **correct tags** in a training corpus.
3. **Learn transformation rules** that would fix the errors (e.g., "If a word is tagged as NN but follows TO, change it to VB").
4. **Apply** the most effective rules sequentially to improve tagging results.

This creates a series of transformations that gradually convert the initial rough tagging into highly accurate tagging.

Example: Transformation Rule

Before Rule	Context	After Rule	Explanation
NN (noun)	If previous tag = TO	VB (verb)	"to play", "to run"
VB (verb)	If next tag = DT	NN (noun)	"run the company"
JJ (adjective)	If previous tag = RB	VB (verb)	"quickly running"

Each rule refines specific mistakes made by the initial tagger, leading to steady improvement.

Algorithm Steps

Step	Description
1. **Initial Tagging**	Start with basic tagger (e.g., most frequent tag).
2. **Error Detection**	Compare with gold-standard corpus to find mis-tags.
3. **Rule Generation**	Create candidate transformation rules to fix those errors.
4. **Rule Scoring**	Measure each rule's effectiveness (how many errors it corrects vs introduces).
5. **Rule Application**	Apply the best rule to the corpus.
6. **Iteration**	Repeat until no significant improvement is possible.

The output is a **ranked list of transformation rules**, applied in sequence during tagging.

Example

Sentence:

Can you can the can?

Initial Tagging (frequency-based):

Can (NN) you (PRP) can (NN) the (DT) can (NN)

Transformation Rules Learned:

- Rule 1: If word = "can" and previous tag = PRP → change NN → VB
- Rule 2: If word = "can" and next tag = DT → change NN → VB

Final Output:

Can (MD) you (PRP) can (VB) the (DT) can (NN) ☑

Characteristics

Feature	Description
Type	Error-driven learning (hybrid)
Learning	Induces rules automatically from tagged data
Transparency	Human-readable rules
Base Model	Often uses statistical or lexicon-based initialization
Performance	Competitive with probabilistic taggers on medium data

Advantages

- Combines **rule clarity** with **data-driven learning**.
- Produces **interpretable correction rules**.
- Works well with **moderate-sized corpora**.
- Avoids black-box behavior — linguists can inspect learned rules.

Limitations

Aspect	Challenge
Training Time	Iterative rule learning can be slow.
Data Requirement	Needs labelled data for rule induction.
Local Context Focus	Rules often rely on nearby tags, not long-range context.

Error Propagation	Incorrect early rules can affect later results.

Shallow Parsing (Chunking)

Definition

Shallow Parsing, also known as **Chunking**, is the process of identifying and grouping words into short, meaningful **phrases** (chunks) such as noun phrases (NP), verb phrases (VP), or prepositional phrases (PP), without constructing a full syntactic parse tree.

It focuses on phrase-level structure rather than the complete sentence hierarchy.

Example:

"The quick brown fox jumps over the lazy dog."

→ [NP The quick brown fox] [VP jumps] [PP over [NP the lazy dog]]

Purpose

Shallow parsing helps identify **local grammatical structure** and **phrase boundaries** that are useful for tasks like entity recognition, relation extraction, and question answering.

Goal	Description
Identify phrase-level chunks	Groups tokens into NP, VP, PP, etc.
Simplify structure	Avoids full parsing complexity.
Capture relationships	Reveals which words "belong together" in context.
Support higher NLP tasks	Acts as input for parsing, NER, or sentiment models.

Key Concept

Shallow parsing builds upon **POS tagging** and uses regular patterns of POS tags to form phrases.

Example Pattern Rules (Simplified):

- NP → DT JJ* NN+ → determiner + optional adjectives + one or more nouns
- VP → VB.* → any verb form
- PP → IN NP → preposition followed by noun phrase

Illustration:

Sentence: "The little boy ate an apple."

POS tags: DT (The) | JJ (little) | NN (boy) | VBD (ate) | DT (an) | NN (apple)

Chunks: [NP The little boy] [VP ate] [NP an apple]

Example Output

Input:

"The quick brown fox jumps over the lazy dog."

Chunked Output:

[NP The quick brown fox]

[VP jumps]

[PP over [NP the lazy dog]]

Chunk Representation

Chunks are often represented using **IOB (Inside–Outside–Beginning)** notation.

Token	POS	Chunk Tag
The	DT	B-NP
quick	JJ	I-NP
brown	JJ	I-NP
fox	NN	I-NP
jumps	VBZ	B-VP
over	IN	B-PP
the	DT	B-NP
lazy	JJ	I-NP
dog	NN	I-NP

- **B-** → Beginning of a chunk
- **I-** → Inside a chunk
- **O** → Outside any chunk

Advantages

- Simple, interpretable, and fast.
- Effective for local grammatical grouping.
- Provides meaningful phrase-level information without full parsing.
- Useful as a preprocessing step for information extraction and NER.

Limitations

Aspect	Challenge
Limited Depth	Does not produce a full hierarchical structure.
Ambiguity Handling	May misidentify phrase boundaries in complex sentences.
Rule Sensitivity	Depends on carefully crafted POS tag patterns.
No Long-Range Dependencies	Cannot represent nested or recursive phrases.

Applications

- **Named Entity Recognition (NER):** Identifies noun phrases containing entity names.
- **Relation Extraction:** Links entities through verbs or prepositions.
- **Information Retrieval:** Improves indexing by grouping meaningful phrases.
- **Speech Recognition:** Detects phrase boundaries for better prosody.
- **Machine Translation:** Aligns source and target phrase structures.

Context-Free Grammar (CFG)

Definition

A **Context-Free Grammar (CFG)** is a formal system used to describe the **syntactic structure** of a language.

It defines a set of **production rules** that describe how sentences (strings of words) can be generated from a start symbol, using **non-terminal** and **terminal** symbols.

In NLP, CFGs are used to represent how words and phrases combine to form grammatically correct sentences.

Components of a CFG

A Context-Free Grammar is represented as a 4-tuple:

$$G = (V, \Sigma, R, S)$$

where:

Symbol	Meaning	Example
V	Set of non-terminal symbols	{S, NP, VP, PP}
Σ	Set of terminal symbols (words)	{the, cat, sleeps}
R	Set of production rules	S → NP VP
S	Start symbol	Sentence (S)

Example Grammar

Let's define a simple grammar for English sentences:

Rule	Description
S → NP VP	A sentence is a noun phrase followed by a verb phrase
NP → DT NN	A noun phrase is a determiner + noun
VP → VB NP	A verb phrase is a verb + noun phrase
DT → "the"	Determiner
NN → "cat"	Noun
VB → "chased"	Verb

Example Derivation:

Sentence: *The cat chased the cat.*

S

→ NP VP

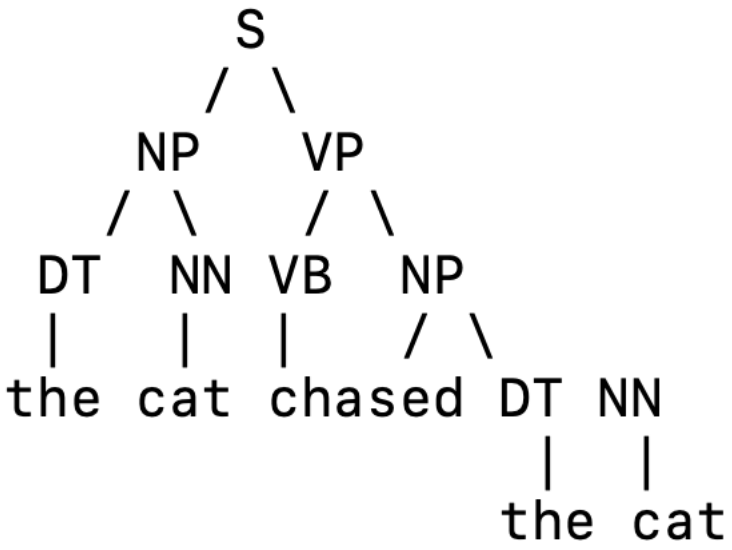
→ (DT NN) (VB NP)

→ the cat chased (DT NN)

→ **the cat chased the cat**

Parse Tree Representation

CFGs can be visualised as **parse trees**, showing how words and phrases are structured hierarchically.



This tree shows the syntactic composition:

- NP = "the cat"
- VP = "chased the cat"
- S = NP + VP

Types of Symbols

Symbol Type	Represents	Example
Terminal	Actual words (tokens).	"dog", "runs", "blue"
Non-terminal	Categories or phrases.	NP, VP, PP, ADJP
Start Symbol	Root of the sentence.	S
Production Rule	How symbols expand.	NP → DT NN

Applications of CFGs in NLP

Application	Description
Parsing	Used to generate parse trees and analyse sentence structure.

Grammar Checking	Validates whether a sentence follows syntactic rules.
Question Answering	Identifies syntactic relationships between words.
Machine Translation	Ensures grammatical correspondence between source and target languages.
Speech Recognition	Models valid syntactic patterns of spoken input.

Advantages

- Provides a **formal and interpretable** representation of syntax.
- Easy to express hierarchical and recursive structures.
- Foundation for many modern **syntactic parsers**.

Limitations

Aspect	Challenge
Ambiguity	A single sentence may have multiple valid parse trees.
Coverage	Real-world language has exceptions and irregularities beyond strict CFGs.
Scalability	Large grammars become complex to manage.
Context Sensitivity	Cannot capture dependencies that rely on external context.

Constituency Parsing

Definition

Constituency Parsing is the process of analysing a sentence and representing it as a **hierarchical structure of nested constituents (phrases)**, according to a **Context-Free Grammar (CFG)**.

Each node in the structure represents a **constituent** — a word or group of words that act as a single unit within the sentence.

In simple terms: it shows how smaller phrases combine to form larger ones, until they make a complete sentence.

Core Idea

Every sentence can be broken down into **phrases (constituents)** like:

- Noun Phrase (NP)
- Verb Phrase (VP)
- Prepositional Phrase (PP)
- Adjective Phrase (ADJP)
- Adverb Phrase (ADVP)

A **constituent parser** uses CFG rules to identify how these phrases are structured within the sentence.

Example

Sentence:

The quick brown fox jumps over the lazy dog.

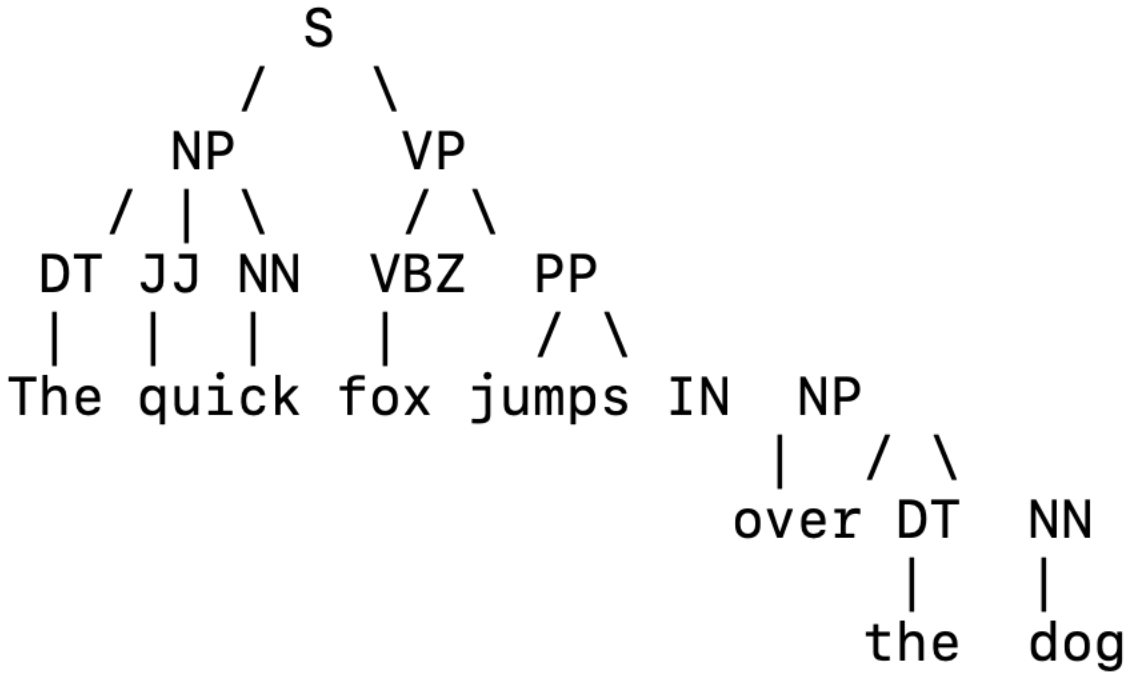
POS Tags:

DT (The), JJ (quick), JJ (brown), NN (fox), VBZ (jumps), IN (over), DT (the), JJ (lazy), NN (dog)

Constituents Identified:

- NP → The quick brown fox
- VP → jumps over the lazy dog
- PP → over the lazy dog
- NP → the lazy dog

Example Parse Tree



This shows how words form phrases and how those phrases combine to form the full sentence.

Constituent Types

Constituent	Meaning	Example
NP (Noun Phrase)	Functions as a subject/object.	"The old man"
VP (Verb Phrase)	Describes an action.	"is reading a book"
PP (Prepositional Phrase)	Expresses relation or position.	"on the table"
ADJP (Adjective Phrase)	Describes a noun.	"very tall"
ADVP (Adverb Phrase)	Describes a verb.	"quite quickly"

How Constituency Parsing Works

1. **Input:** A sentence + a grammar (CFG).
2. **Process:** The parser applies grammar rules recursively to identify valid phrase combinations.
3. **Output:** A hierarchical **parse tree** representing sentence structure.

Ambiguity in Constituency Parsing

A single sentence may have **multiple valid parse trees** (known as *syntactic ambiguity*).

Example:

I saw the man with the telescope.

- Interpretation 1: "I used a telescope to see the man." → PP modifies *saw*
- Interpretation 2: "I saw the man who had a telescope." → PP modifies *man*

Both parses are syntactically valid — ambiguity is resolved later using context or semantics.

Advantages

- Produces **clear hierarchical structure** for understanding syntax.
- Useful for grammar checking, paraphrasing, and translation.
- Enables identification of **phrase boundaries** and **relationships**.
- Forms the theoretical base for **semantic role labelling** and **semantic parsing**.

Limitations

Aspect	Challenge
Computationally Expensive	Building trees for long sentences is resource-intensive.

Ambiguity Explosion	Sentences may have many possible trees.
Grammar Dependency	Requires well-defined CFG rules.
Limited Context Sensitivity	Cannot fully capture meaning or long-distance relations.

Applications

Application	Purpose
Grammar Checking	Identifies syntactic errors.
Machine Translation	Aligns syntactic structures between languages.
Question Answering	Helps identify subjects, predicates, and objects.
Information Extraction	Recognises entities and their relationships.
Text Summarisation	Identifies key syntactic components.

Dependency Parsing

Definition

Dependency Parsing is the process of analysing the grammatical structure of a sentence by establishing **relationships (dependencies)** between "head" words and their dependents.

Unlike **Constituency Parsing**, which focuses on *phrases*, dependency parsing focuses on **word-to-word relations** that directly express the *syntactic function* of each word in the sentence.

It represents a sentence as a **directed graph** of words, where each arrow (dependency) shows which word depends on which.

Core Idea

Every word (except the root) in a sentence depends on another word — usually the word it modifies or complements.

The word on which others depend is called the **head**, and the linking words are called **dependents**.

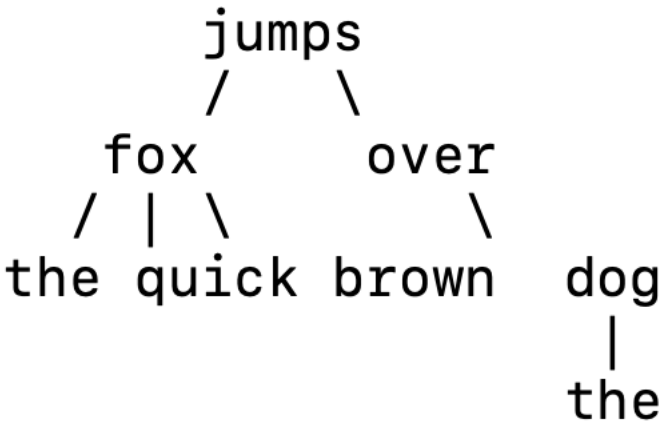
Example:

Sentence: *The quick brown fox jumps over the lazy dog.*

Head	Dependent	Relation
jumps	fox	subject (nsubj)
jumps	over	prepositional modifier (prep)
over	dog	object of preposition (pobj)
dog	the	determiner (det)
fox	the	determiner (det)
jumps	ROOT	root of the sentence

Visual Representation

Here, **"jumps"** is the **root verb** — all other words attach directly or indirectly to it.



Dependency Relations

Relation	Description	Example
nsubj	Nominal subject	"The dog runs." → (runs, dog)
dobj	Direct object	"She eats apples." → (eats, apples)
iobj	Indirect object	"She gave him a gift." → (gave, him)
det	Determiner	"The cat." → (cat, the)
amod	Adjectival modifier	"red ball." → (ball, red)
advmod	Adverbial modifier	"runs quickly." → (runs, quickly)
prep	Prepositional modifier	"sits on chair." → (sits, on)
pobj	Object of preposition	"on the chair." → (on, chair)

Comparison: Constituency vs. Dependency Parsing

Aspect	Constituency Parsing	Dependency Parsing
Focus	Phrase structure	Word-to-word relations
Representation	Tree of phrases (NP, VP, PP)	Directed graph of dependencies
Output	Constituency tree	Dependency tree
Grammar Basis	CFG	Dependency Grammar
Usefulness	Good for phrase-level analysis	Good for relation extraction and semantics

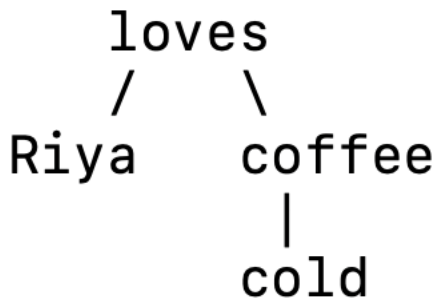
Example (Dependency Output)

Sentence:

Riya loves cold coffee.

Dependencies (simplified):

Head	Dependent	Relation
loves	Riya	nsubj
loves	coffee	dobj
coffee	cold	amod



How Dependency Parsing Works

1. **Input:** Tokenised and POS-tagged sentence.
2. **Process:** Identify the head–dependent relationships using dependency grammar rules or machine learning models.
3. **Output:** A **dependency tree** that captures the syntactic structure through binary relations.

Types of Dependency Parsers

Type	Approach
Rule-Based	Uses hand-written linguistic rules to assign dependencies.
Transition-Based	Builds the tree incrementally by predicting next relations.
Graph-Based	Finds the most probable dependency tree globally using scoring.
Neural Parsers	Learn directly from data using embeddings and neural networks.

Advantages

- Provides **direct syntactic relations** between words.
- More compact and semantically meaningful than phrase structure trees.
- Useful for **information extraction**, **relation detection**, and **semantic parsing**.
- Easier to apply in machine translation and question answering systems.

Limitations

Aspect	Challenge
Ambiguity	Same sentence can yield multiple dependency graphs.
Complex Constructions	Difficult to represent coordination and ellipsis.
Cross-Linguistic Variability	Languages differ in dependency rules.
Interpretability	Requires linguistic expertise to interpret fine-grained relations.

Applications

- **Relation Extraction:** Find subject–verb–object triples.
- **Information Retrieval:** Identify entities and their actions.
- **Semantic Role Labelling:** Understand roles like agent, object, location.
- **Chatbots / QA Systems:** Map syntax to intent structure.
- **Grammar Checking:** Detect dependency mismatches and agreement errors.

Syntax Trees vs. Parse Trees

Definition

A **Syntax Tree** (also called a **Parse Tree**) visually represents the **syntactic structure** of a sentence according to a given grammar.

It shows **how words combine into phrases and clauses**, and how these components relate hierarchically to form a complete sentence.

However, depending on the focus and level of abstraction, we distinguish between two related forms:

Type	Focus	Purpose
Parse Tree	Derivation according to grammar rules	Shows rule-based generation from grammar
Syntax Tree	Abstract syntactic structure	Shows grammatical roles and hierarchy

Parse Tree

A **Parse Tree** explicitly represents **grammar rule applications** step-by-step — how a sentence is *derived* from a **Context-Free Grammar (CFG)**.

Example:

Grammar Rules

S → NP VP

NP → DT NN

VP → VB NP

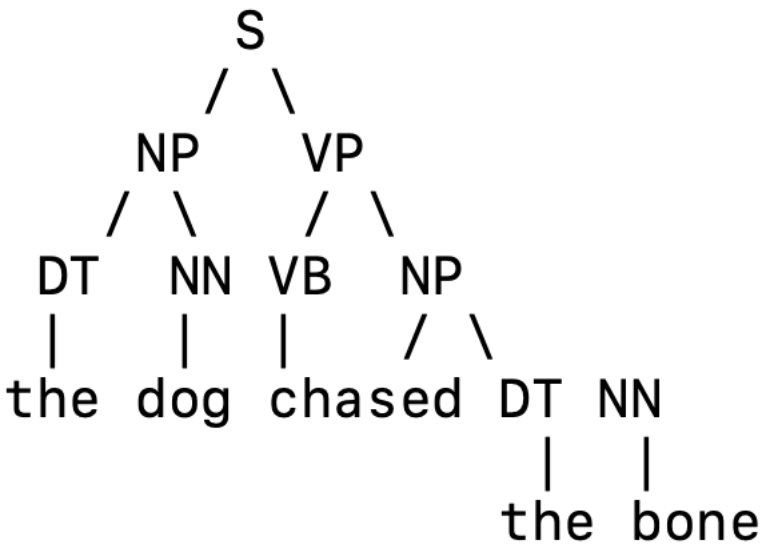
DT → “the”

NN → “dog” | “bone”

VB → “chased”

Sentence: “The dog chased the bone.”

Parse Tree (Rule-Derivation View):



Here, every node corresponds directly to a grammar rule.

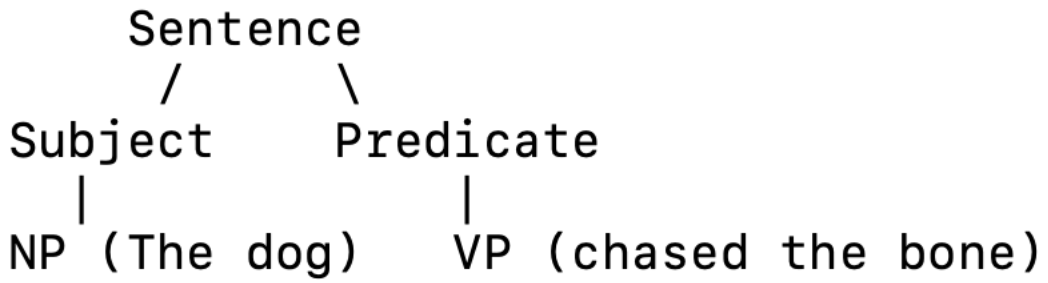
It’s **closely tied to CFG derivations** and helps linguists or parsers understand rule-based sentence generation.

Syntax Tree

A **Syntax Tree** (often more abstract) represents **grammatical relations and categories**, not just derivation steps.

It may **omit intermediate grammar rules** and highlight **functional roles** like *subject*, *predicate*, *object*, etc.

Example (Abstracted View):



This is simpler and **focuses on the logical structure** rather than grammar expansion.

Comparison: Parse Tree vs. Syntax Tree

Aspect	Parse Tree	Syntax Tree
Representation	Based on grammar rule derivations	Based on grammatical relationships
Structure	Includes all intermediate non-terminals	Abstracts away non-essential nodes
Purpose	To trace how grammar generates a sentence	To show grammatical and semantic relations
Level of Detail	More detailed (step-by-step rules)	More general (role-focused)
Used In	Parsing algorithms, CFG validation	Linguistic analysis, visualization
Example Focus	NP → DT NN	Subject → NP

Relation Between the Two

- Every **parse tree** implies one **syntax tree** — but not vice versa.
- Syntax trees are often **simplified representations** derived from parse trees.
- In NLP practice:
  - Constituency and dependency parsers often **output syntax trees**.
  - Grammar-based parsers (CFG-based) **generate parse trees**.

Illustrative Example

Sentence: *The cat sat on the mat.*

Representation Type	Illustration
Parse Tree (Grammar Rules)	S → NP VP → (DT NN) (VBD PP) → the cat sat (IN NP) → on (DT NN) → on the mat
Syntax Tree (Functional Roles)	[S [NP The cat] [VP sat [PP on [NP the mat]]]]

Applications

Tree Type	Usage
Parse Tree	Used for teaching grammars, syntactic parsing, and rule validation.
Syntax Tree	Used for downstream NLP tasks (relation extraction, semantics).

Advantages

- Both provide **interpretable hierarchical representations** of language structure.
- Useful for **semantic role identification** and **syntactic error detection**.
- Facilitate **machine translation** and **grammar-based generation**.

Limitations

Aspect	Challenge
Ambiguity	A sentence may have multiple valid trees.
Complexity	Long sentences lead to large and dense trees.
Context Sensitivity	Trees reflect grammar, not deeper meaning or pragmatics.

Syntax Trees vs. Parse Trees

Definition

Deep Parsing is the process of analysing a sentence to capture **complete syntactic and semantic structures**, including **hierarchical phrase relationships**, **dependencies**, and **grammatical functions** such as subject, object, and modifiers.

It goes beyond surface-level or phrase-level parsing to provide a **comprehensive structural and meaning-oriented representation** of the sentence.

Deep parsing seeks to understand *how* a sentence is built and *what it means*, combining syntax with semantics.

Core Concept

While **shallow parsing** focuses on chunks (like NPs or VPs), and **constituency/dependency parsing** focuses on structure,

deep parsing integrates **syntactic, semantic, and functional** information into a unified analysis.

It attempts to answer:

- *Who did what to whom?*
- *When, where, and how did the action occur?*
- *What roles do the entities play in the sentence?*

Example

Sentence:

Riya gave her friend a gift yesterday.

Aspect	Captured by Deep Parser
Syntactic Structure	Subject = “Riya”; Verb = “gave”; Objects = “friend”, “gift”
Semantic Roles	Agent = Riya, Recipient = friend, Theme = gift
Temporal Modifier	“yesterday” (time)
Dependency Links	gave → friend (iobj), gave → gift (dobj), gave → yesterday (advmod)

The deep parser thus builds both a **syntactic tree** and a **semantic role graph** that together express meaning.

Comparison with Other Parsing Levels

Aspect	Shallow Parsing	Constituency/Dependency Parsing	Deep Parsing
Focus	Phrase detection	Structure and relations	Syntax + Semantics
Output	NP, VP, PP chunks	Parse trees or dependency trees	Full structural + semantic analysis
Detail	Limited	Intermediate	Comprehensive
Goal	Identify phrase boundaries	Show grammatical links	Extract meaning and roles

Output Representation

Deep parsers typically generate **graph-like representations** that combine:

- **Phrase structure** (who is connected to whom)
- **Semantic roles** (what functions they serve)
- **Feature annotations** (number, gender, tense, aspect)

Example (Simplified Semantic Representation):

give(Riya, gift, friend)

time(give, yesterday)

This expresses:

Riya (agent) gave (action) a gift (theme) to her friend (recipient) yesterday (time).

How Deep Parsing Works

1. **Tokenisation and POS Tagging**

Break the sentence into words and assign parts of speech.

2. **Syntactic Parsing**

Generate constituency or dependency trees.

3. **Semantic Role Labelling (SRL)**

Identify who did what to whom, when, where, and how.

4. **Feature Enrichment**

Add attributes such as number, gender, tense, and aspect.

5. **Logical / Graph Representation**

Produce a rich, machine-readable structure combining grammar and meaning.

Advantages

- Provides **complete structural and semantic understanding**.
- Enables advanced NLP applications like information extraction, question answering, and dialogue systems.
- Bridges **linguistic rules and statistical learning**.
- Supports **logical inference and reasoning** on text.

Limitations

Aspect	Challenge
Computational Complexity	Deep models are resource-intensive.
Ambiguity Resolution	Requires contextual and world knowledge.
Training Data	Needs richly annotated corpora (e.g., PropBank, FrameNet).
Domain Adaptation	May perform poorly outside training domains.

Applications

- **Question Answering:** Understands questions at meaning level.
- **Information Extraction:** Extracts who, what, where, and when from text.
- **Machine Translation:** Preserves deep syntactic and semantic relations.
- **Dialogue Systems:** Interprets user intent beyond surface structure.
- **Text Summarisation:** Identifies central ideas through semantic roles.

Grammatical Agreement Checking

Definition

Grammatical Agreement Checking is the process of ensuring that words within a sentence agree with each other according to the grammatical rules of number, person, gender, and tense.

It verifies whether the **syntactic dependencies** established during parsing are **morphologically and grammatically consistent**.

In essence, it ensures that all parts of a sentence "fit together" correctly.

Core Idea

In English and many other languages, certain words must **agree** in their grammatical features:

Agreement Type	Example	Explanation

Subject–Verb Agreement	She *writes* a letter. (✓) / She *write* a letter. (✗)	Verb agrees with the subject in number and person.
Pronoun–Antecedent Agreement	The *boy* lost *his* pen.	Pronoun must agree with its antecedent in gender and number.
Determiner–Noun Agreement	*This apple / These apples*	Determiner agrees with noun in number.
Tense Consistency	He *was eating* while she *cooked*.	Verb tenses must be logically aligned.

Grammatical agreement ensures both **syntactic correctness** and **semantic clarity**.

How Agreement Checking Works

Grammatical agreement is validated through **syntactic and morphological analysis** — often after parsing.

1. **POS Tagging:** Identify grammatical categories of words.
2. **Dependency Parsing:** Determine head–dependent relations.
3. **Feature Extraction:** Retrieve morphological features like number, gender, person, and tense.
4. **Rule Checking:** Verify consistency between dependent pairs (e.g., subject–verb, noun–determiner).

Agreement in Dependency Structure

In dependency parsing, the parser identifies relations such as nsubj, det, and amod.

Grammatical agreement checking validates these using feature compatibility.

Example:

Those apples are sweet.

Head	Dependent	Relation	Check
are	apples	nsubj	number(plural) → match ✓
apples	Those	det	number(plural) → match ✓

If mismatch occurs (e.g., *This apples*), agreement rules flag it as incorrect.

Agreement Features

Feature	Possible Values	Checked Between

Number	Singular / Plural	Subject–Verb, Noun–Determiner
Person	1st / 2nd / 3rd	Subject–Verb
Gender	Masculine / Feminine / Neutral	Pronoun–Noun
Case	Nominative / Objective / Possessive	Noun–Pronoun
Tense / Aspect	Past / Present / Perfect	Verb sequences

Rule Examples

Rule	Example (✓)	Example (✗)
Singular subject → singular verb	He *runs* fast.	He *run* fast.
Plural subject → plural verb	They *run* fast.	They *runs* fast.
Singular noun → singular determiner	*This apple* is red.	*These apple* is red.
Pronoun matches antecedent	*John lost his pen.*	*John lost her pen.*

Applications

Application	Purpose
Grammar Checkers	Detect subject–verb and pronoun–noun inconsistencies.
Machine Translation	Maintain agreement between source and target structures.
Text Correction Tools	Provide feedback on grammatical accuracy.
Essay Evaluation Systems	Score syntax quality automatically.
Speech-to-Text Correction	Validate morpho-syntactic consistency in transcripts.

Advantages

- Improves **sentence fluency and correctness**.
- Enhances **readability** and **clarity**.
- Supports **automated grammar and writing tools**.
- Strengthens downstream NLP tasks by reducing syntactic noise.

Limitations

Aspect	Challenge
Ambiguity	Some constructions (e.g., collective nouns) are flexible.
Language Variability	Rules differ across languages.
Error Propagation	Parsing or tagging errors can cause false alerts.
Context Dependence	Some agreements rely on discourse context, not syntax.