NLP

**✅ Technical Explanation (Simple & Clear)**

**Natural Language Processing (NLP)** is a branch of Artificial Intelligence (AI) that helps computers understand, interpret, generate, and respond to **human language** — the way we speak or write.

NLP combines **computational linguistics**, **machine learning**, and **deep learning** to enable applications like:

* **Text classification** (e.g., spam vs. non-spam)
* **Chatbots / Virtual assistants** (e.g., Siri, Alexa, ChatGPT)
* **Language translation** (e.g., Google Translate)
* **Sentiment analysis** (e.g., analyzing movie reviews)
* **Speech recognition** (e.g., converting spoken Telugu to text)

**🎬 Tollywood Analogy**

**Imagine NLP as the "Brahmanandam" of AI** — it's the one who listens to everyone on-screen, understands the scene, responds with witty or emotional lines, and helps the story move forward!

Let’s make it even more fun:

**🎭 Scene Setup:**

**Hero:** Human (speaks Telugu or Hindi)  
**AI System:** A robot or assistant (like a robot in *Robo* or *Aditya 369*)  
**Goal:** Hero wants the robot to understand what he says and reply back properly.

Without NLP, the robot hears:

"ఇదేమి మాటల దండకం!?"

But **with NLP**, the robot can understand:

"He is talking emotionally because he’s angry."

So the robot (AI) uses:

* **Tokenization** to split the sentence word by word.
* **Sentiment Analysis** to realize the emotion (anger).
* **Named Entity Recognition** to pick up important parts like “అమ్మ” (mother), “వీధి” (place), etc.
* **Response Generation** to reply in a human-like way.

**NLP = The on-screen dialogue writer + translator + emotion detector + voice assistant – all in one AI brain.**

**💡 Example:**

🗣️ Human: “ఈ సినిమాలో హీరో ఎవరు?”  
🤖 NLP-Powered App: “ఈ సినిమాలో హీరో మహేష్ బాబు గారు.”

**✅ Top Python Libraries for Natural Language Processing (NLP):**

| **Library** | **Purpose / Features** |
| --- | --- |
| **NLTK** (Natural Language Toolkit) | Great for beginners, supports tokenization, stemming, lemmatization, POS tagging, etc. |
| **spaCy** | Industrial-strength NLP with fast tokenization, named entity recognition (NER), and syntactic analysis. |
| **TextBlob** | Simple API for common NLP tasks like sentiment analysis, translation, and noun phrase extraction. |
| **Transformers (Hugging Face)** | State-of-the-art NLP using pre-trained models like BERT, GPT, T5, etc. |
| **Gensim** | Topic modeling, document similarity, and word embeddings (e.g., Word2Vec, LDA). |
| **Flair** | Easy-to-use NLP library built on PyTorch, supports NER, classification, etc. |
| **Stanza** (by Stanford) | Supports multiple languages, dependency parsing, POS tagging, etc. |
| **OpenNLP** | Java-based toolkit for tasks like sentence detection, tokenization, NER. |
| **AllenNLP** | Research-focused library built on PyTorch, great for deep NLP experiments. |
| **fastText (Facebook)** | Lightweight and fast text classification, word embeddings. |

**⚙️ Other Useful Tools:**

* **Tesseract OCR** – for extracting text from images (used with NLP for scanned docs).
* **BeautifulSoup / Scrapy** – for web scraping before NLP processing.
* **SpeechRecognition** – converts speech to text before applying NLP.

**🔍 Example Use Case:**

python

CopyEdit

import spacy

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp("Mahesh Babu acted brilliantly in the movie Sarkaru Vaari Paata.")

for ent in doc.ents:

print(ent.text, ent.label\_)

**Output:**  
Mahesh Babu - PERSON  
Sarkaru Vaari Paata - WORK\_OF\_ART

**WORKING WITH PDF:**

| **Library** | **Purpose / Features** |
| --- | --- |
| **PyPDF2 / PyPDF4 / pypdf** | **Read, merge, split, and rotate PDF files. Extract text.** |
| **pdfplumber** | **Accurate text extraction with layout preservation (tables, columns).** |
| **PDFMiner.six** | **Detailed parsing of PDF structure. Best for custom layouts.** |
| **ReportLab** | **Create PDF documents from scratch (e.g., invoices, reports).** |
| **fitz (PyMuPDF)** | **Read text, extract images, render pages — faster and image-aware.** |
| **camelot / tabula-py** | **Extract tables from PDFs (especially scanned tables).** |

**🔧 Common Tasks & Sample Code**

**1. ✅ Reading PDF Text (PyPDF2 / pypdf)**

python

CopyEdit

from PyPDF2 import PdfReader

reader = PdfReader("sample.pdf")

for page in reader.pages:

print(page.extract\_text())

**2. 📄 Extracting Structured Text (pdfplumber)**

python

CopyEdit

import pdfplumber

with pdfplumber.open("sample.pdf") as pdf:

for page in pdf.pages:

text = page.extract\_text()

print(text)

📝 *Great for layout-sensitive files (like resumes or tables).*

**3. ✂️ Merging or Splitting PDFs (pypdf)**

python

CopyEdit

from PyPDF2 import PdfMerger

merger = PdfMerger()

merger.append("file1.pdf")

merger.append("file2.pdf")

merger.write("merged.pdf")

merger.close()

**4. 🖼️ Extracting Images (PyMuPDF / fitz)**

python

CopyEdit

import fitz # PyMuPDF

doc = fitz.open("sample.pdf")

for page\_num in range(len(doc)):

page = doc[page\_num]

images = page.get\_images(full=True)

print(f"Page {page\_num+1} has {len(images)} image(s).")

SPACY

**✅ What is spaCy?**

**spaCy** is a powerful, fast, and production-ready **Natural Language Processing (NLP)** library in Python.

It helps you perform common NLP tasks like:

* Tokenization
* Part-of-speech (POS) tagging
* Named Entity Recognition (NER)
* Dependency Parsing
* Lemmatization
* Sentence segmentation

**🔧 spaCy Basics: Key Code & Concepts**

**1. Install and Load Language Model**

bash

CopyEdit

pip install spacy

python -m spacy download en\_core\_web\_sm

**2. Basic Usage**

python

CopyEdit

import spacy

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp("Mahesh Babu starred in Sarkaru Vaari Paata.")

for token in doc:

print(token.text, token.pos\_, token.lemma\_)

**Output:**

python-repl

CopyEdit

Mahesh PROPN Mahesh

Babu PROPN Babu

starred VERB star

...

**3. Named Entity Recognition (NER)**

python

CopyEdit

for ent in doc.ents:

print(ent.text, ent.label\_)

**Output:**

nginx

CopyEdit

Mahesh Babu - PERSON

Sarkaru Vaari Paata - WORK\_OF\_ART

**🎬 Tollywood Analogy for spaCy**

Let’s imagine spaCy is like a **top film director** — let’s call him **“Trivikram NLP”**

Here’s how spaCy works on a Tollywood script:

**🎭 Scene: Dialogues from a Movie Script**

**Input Sentence:**  
*"Mahesh Babu starred in Sarkaru Vaari Paata and fans loved it."*

**🎬 spaCy's Crew (Feature Functions):**

| **spaCy Role** | **Tollywood Role** | **What It Does** |
| --- | --- | --- |
| **Tokenizer** | Script Assistant | Breaks sentence into words (like breaking lines in script) |
| **POS Tagger** | Casting Director | Tags each word: is it a noun (hero), verb (action), etc.? |
| **NER (Named Entity Recog)** | Hero/Director Spotter | Finds important names like "Mahesh Babu" (actor), "Sarkaru..." (movie) |
| **Dependency Parser** | Scene Director | Understands who did what to whom (relations between words) |
| **Lemmatizer** | Dialogue Polisher | Converts verbs to base form: “loved” → “love” |
| **Pipeline** | Movie Editor (Workflow) | Executes all tasks in sequence for perfect analysis |

**📽️ spaCy = Trivikram NLP**

It reads the story, breaks it into shots, understands hero names, scenes, and even emotions — **all in one pass**, ready for dubbing or subtitles (NLP tasks).

**💡 Real Use Cases of spaCy:**

* Chatbots that recognize **entities** (e.g., movies, people, places)
* Resume parsers that extract skills and roles
* Text summarizers and translation engines
* reviews”)

**Tokenization**

**✅ What is Tokenization in NLP?**

**Tokenization** is the process of **breaking down text into smaller pieces**, called **tokens**.  
These tokens can be **words**, **subwords**, or **sentences** depending on the use case.

**🔍 Why is Tokenization important?**

Before a machine can **understand language**, it needs to break it into parts. Just like:

* Actors need their **lines** before they can perform.
* Editors cut the **scenes** before creating the final movie.

**🔧 spaCy Tokenization Example:**

python

CopyEdit

import spacy

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp("Mahesh Babu acted in Sarkaru Vaari Paata.")

for token in doc:

print(token.text)

**Output:**

nginx

CopyEdit

Mahesh

Babu

acted

in

Sarkaru

Vaari

Paata

.

🧠 *Each word (including punctuation) becomes a* ***token****.*

**🎬 Tollywood Analogy for Tokenization**

Imagine Tokenization is like the **clapboard boy (assistant director)** in a movie shoot.

**🎭 Scene:**

Script:  
*"Mahesh Babu acted in Sarkaru Vaari Paata."*

**Tokenization Job:**  
Break this script into **dialogue units (tokens)** so:

* The **hero** knows his line
* The **camera team** knows where to cut
* The **director** can assign expressions to each word!

**🎬 Movie Comparison Table:**

| **NLP Term** | **Tollywood Equivalent** | **Explanation** |
| --- | --- | --- |
| Token | Dialogue Line / Word | Each word or punctuation is a separate unit |
| Tokenization | Scene Breakdown / Script Cutting | Prepares raw script for shooting |
| Text | Movie Script | Unprocessed paragraph of text |

**🧾 Real-Life Analogy:**

You give spaCy the **dialogue sheet**, and it hands you **each word like a dialogue cue card**.

**📌 Common Types of Tokenization:**

* **Word Tokenization** – Splits text into words (used by spaCy, NLTK).
* **Sentence Tokenization** – Splits paragraphs into sentences.
* **Subword Tokenization** – Used in transformers (BERT), splits unknown words into pieces (e.g., play##ing).
* **Character Tokenization** – Splits into individual characters (used in spelling models).

**💻 TECHNICAL DEFINITION: What is Stemming?**

**Stemming** is the process of **reducing a word to its base or root form**.  
The root **may not be a real word**, but it’s useful for matching similar terms.

**🔍 Example:**

| **Original Word** | **Stem** |
| --- | --- |
| playing | play |
| played | play |
| player | player (depends on the algorithm) |

**💡 Why Use Stemming?**

To help **search engines, chatbots, and text models** understand that:

* "playing", "played", and "plays" → all relate to the same **concept**: play.

This reduces vocabulary size and improves generalization in NLP models.

**🔧 Common Stemming Algorithms:**

| **Algorithm** | **Notes** |
| --- | --- |
| Porter Stemmer | Most popular, rule-based |
| Snowball | Improved version of Porter |
| Lancaster | Very aggressive, faster |
|  |  |

**Stemming**

**🎬 TOLLYWOOD ANALOGY: Stemming with Heroes**

Imagine we’re analyzing **all Chiranjeevi movies**.

* You see names like:
  + "Chiru"
  + "Megastar"
  + "Chiranjeevi garu"
  + "Annayya"
  + "Hero Chiranjeevi"

Technically, they’re all different words.

**👉 But you (as a fan or NLP model) know they all mean one thing: chiranjeevi.**

So, **stemming** is like a **fan mentally reducing all nicknames and titles** to the **root hero**:

arduino

CopyEdit

"Annayya" → "Chiru" → "Chiranjeevi"

**🧠 Another Analogy: Movie Titles**

Let’s say you're indexing movies about **revenge**:

* "Revenge"
* "Revenged"
* "Revengeful"
* "Revenging"

With stemming:

* All these become → reveng

Even if reveng is not a real word, your NLP system knows it represents the **same concept**.

**🧪 Summary Table**

| **Term** | **Explanation** |
| --- | --- |
| Stemming | Cutting a word to its root base |
| Use Case | Search, chatbot understanding, classification |
| Tollywood Link | Fan reducing all hero names to “Chiranjeevi” |
| Real Output | "playing" → "play" |

**✅ Bonus: How to Use in Python**

python

CopyEdit

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem("playing")) # play

print(stemmer.stem("played")) # play

print(stemmer.stem("player")) # player

**💻 TECHNICAL DEFINITION: What is Lemmatization?**

**Lemmatization** is the process of reducing a word to its **base or dictionary form (called a lemma)**, using **vocabulary and grammar rules**.

Unlike stemming, lemmatization:

* **Considers the context** (like part of speech)
* Always returns a **real word**

**🔍 Examples:**

| **Word** | **Lemma (Base Form)** |
| --- | --- |
| playing | play |
| better | good |
| running | run |
| was | be |
| went | go |

✅ Lemmatization is more accurate and **context-aware**, but also **slower** than stemming.

**⚙️ Lemmatization in Python (NLTK)**

python

CopyEdit

import nltk

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

nltk.download('omw-1.4')

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize("running", pos='v')) # run

print(lemmatizer.lemmatize("better", pos='a')) # good

print(lemmatizer.lemmatize("was", pos='v')) # be

**🤖 Lemmatization vs. Stemming**

| **Feature** | **Stemming** | **Lemmatization** |
| --- | --- | --- |
| Rule-based cutting | Yes | No |
| Uses dictionary | ❌ No | ✅ Yes |
| Context-aware | ❌ No | ✅ Yes |
| Returns real word? | ❌ Not always | ✅ Always |

**🎬 TOLLYWOOD ANALOGY: Lemmatization**

Imagine you're making a **movie search engine** and a user searches:

"I saw the movie where **Bunny was dancing**."

Now:

* “Bunny” = Allu Arjun
* “was dancing” = dance (root form)

**➤ Lemmatization thinks like a Tollywood movie buff:**

“Ah! Bunny means Allu Arjun and dancing means dance. He dances in almost every film!” 🎥💃

So NLP lemmatization:

* Converts "was dancing" → "dance" (actual word)
* Understands "Bunny" means "Allu Arjun"

**✅ Analogy Table**

| **Phrase** | **Lemmatized Meaning** |
| --- | --- |
| Bunny dancing | Allu Arjun + dance |
| Chiru’s acting | Chiranjeevi + act |
| Sid sang a song | Sid Sriram + sing + song |
| Nani was running | Nani + run |

➡️ This helps **match context-rich queries** with correct results in NLP.

**✅ Use Cases for Lemmatization**

* **Chatbots** (to understand user intent better)
* **Sentiment Analysis** (mapping variations of words)
* **Search Engines** (matching queries like "run", "running", "ran")
* **Grammar correction**

**Summary**

| **Criteria** | **Stemming** | **Lemmatization** |
| --- | --- | --- |
| Output | root (not always a real word) | base form (always real word) |
| Speed | ✅ Fast | ❌ Slower |
| Accuracy | ❌ Less accurate | ✅ More accurate |
| Uses POS Tagging | ❌ No | ✅ Yes |

In **Natural Language Processing (NLP)**, **stop words** are **common words** that are often **filtered out** from text data before processing. These words usually **do not carry significant meaning** for tasks like text classification, sentiment analysis, or topic modeling.

**🔹 What Are Stop Words?**

Examples of English stop words:

csharp

CopyEdit

a, an, the, and, is, in, on, at, which, for, with, to, from, it, by, of, be, that, this

These words appear frequently but usually **do not help** in understanding the **core meaning** of a sentence.

**🔹 Why Remove Stop Words?**

* To **reduce noise** in text.
* To **decrease dimensionality** of the feature space.
* To **improve algorithm performance** by focusing on meaningful words.

Example:  
Original sentence:  
👉 *"This is a sample sentence for NLP processing."*  
After removing stop words:  
👉 *"sample sentence NLP processing"*

**🔹 Popular Stop Word Lists**

* **NLTK** (nltk.corpus.stopwords)
* **spaCy** (spacy.lang.en.stop\_words)
* **sklearn** (sklearn.feature\_extraction.text.ENGLISH\_STOP\_WORDS)

**🔹 Python Example (Using NLTK)**

python

CopyEdit

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

nltk.download('stopwords')

nltk.download('punkt')

text = "This is a sample sentence for NLP processing."

words = word\_tokenize(text.lower())

filtered = [word for word in words if word not in stopwords.words('english') and word.isalnum()]

print(filtered)

🔸 Output:  
['sample', 'sentence', 'nlp', 'processing']

**🔹 When *Not* to Remove Stop Words?**

* In **text generation**, **translation**, or **summarization**, stop words **carry grammar and meaning**.
* In **semantic analysis**, stop words may have **contextual importance**.

**🔹 Customizing Stop Words**

You can modify stop word lists based on your use case:

python

CopyEdit

my\_stopwords = set(stopwords.words('english'))

my\_stopwords.remove('not') # Keep 'not' for sentiment analysis

Would you like an example using **spaCy**, **sklearn**, or in another language like **Hindi/Telugu**?

**You said:**

you missed tollywoord movies anology'

**ChatGPT said:**

You're absolutely right! Let's explain **Stop Words in NLP** using a **Tollywood (Telugu movie)** analogy to make it memorable and fun! 🎬🍿

**🎭 Stop Words in NLP — Tollywood Movie Analogy**

**🎞️ Imagine You're Watching a Telugu Movie:**

You’re watching a **Mahesh Babu** or **Allu Arjun** blockbuster. What do you focus on?

* Hero's **dialogues**
* Villain's **plot**
* Dramatic **twists**
* Emotional **monologues**
* Epic **fight scenes**
* Mass **songs**

But do you remember every:

* “**Orey!**”
* “**Em cheptunnav ra!**”
* “**Nuvvu vellipovadam better!**”
* “**Baagundi.**”

These are like **background expressions** or **fillers** — they are common in every movie and **don’t change the plot much**.

**🍿 Analogy Breakdown:**

| **Movie Element** | **NLP Equivalent** |
| --- | --- |
| Repetitive filler dialogues | Stop Words |
| Main storyline, twists, big dialogues | Important keywords (nouns, verbs) |
| Editing out fillers for trailer | Removing stop words |

**✅ Example:**

**Telugu Sentence:**

"Nenu cinema chudatam kosam theatre ki vellanu."

* **Stop Words (Telugu):** "nenu", "ki", "kosam"
* **Meaningful Words:** "cinema", "chudatam", "theatre", "vellanu"

**After removing stop words:**  
👉 "cinema chudatam theatre vellanu"  
Still makes sense, right? That’s how NLP treats text!

**💡 Moral of the Story:**

In **text processing**, just like in movie trailers, we remove repetitive parts (stop words) to **highlight the main plot**. The hero's punch dialogue is like a keyword; the background chatter is like a stop word.

**🔍 Vocabulary Matching in NLP**

**Vocabulary Matching** is the process of **comparing words or tokens in a text** against a **predefined set of vocabulary words** to:

* Check which words are known or unknown
* Filter out irrelevant or unseen words
* Use only important words (like in search engines, keyword extraction, etc.)

**📌 Real-Life Analogy (Tollywood Style 🍿)**

Imagine a **casting call** where only selected actors are invited. The casting team has a **list of preferred actors (vocabulary)**.

Now, when actors (tokens) show up, the team matches them to the list:

* If the actor is in the list → ✅ allowed (match)
* If not → ❌ rejected (no match)

In NLP:

* **Text tokens** = actors arriving
* **Vocabulary list** = selected actors
* **Matching** = check who gets in

**✅ Example in Python: Vocabulary Matching with NLTK and custom list**

python

CopyEdit

from nltk.tokenize import word\_tokenize

text = "This is a sample sentence for NLP processing."

vocabulary = {"nlp", "processing", "sample", "data", "model"} # predefined vocabulary

tokens = word\_tokenize(text.lower())

matched = [word for word in tokens if word in vocabulary]

unmatched = [word for word in tokens if word not in vocabulary]

print("Matched:", matched)

print("Unmatched:", unmatched)

**✅ Output:**

bash

CopyEdit

Matched: ['sample', 'nlp', 'processing']

Unmatched: ['this', 'is', 'a', 'sentence', 'for', '.']

**🔧 Vocabulary Matching in scikit-learn (during vectorization)**

python

CopyEdit

from sklearn.feature\_extraction.text import CountVectorizer

vocab = ["nlp", "processing", "sample", "data", "model"]

vectorizer = CountVectorizer(vocabulary=vocab)

corpus = ["This is a sample sentence for NLP processing."]

X = vectorizer.fit\_transform(corpus)

print("Feature Names:", vectorizer.get\_feature\_names\_out())

print("Vector Output:", X.toarray())

**✅ Output:**

lua

CopyEdit

Feature Names: ['nlp' 'processing' 'sample' 'data' 'model']

Vector Output: [[1 1 1 0 0]]

Explanation:

* 1 means word matched (present in the corpus and in vocab)
* 0 means word not matched

**🔍 spaCy-based Vocabulary Matching**

python

CopyEdit

import spacy

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp("This is a sample sentence for NLP processing.")

vocab = {"nlp", "processing", "sample", "data", "model"}

matched = [token.text for token in doc if token.text.lower() in vocab]

print("spaCy Matched:", matched)

**📦 Use Cases of Vocabulary Matching**

* Keyword detection
* Search engine queries
* Intent recognition
* Grammar/spelling checkers
* Named Entity Filtering
* Domain-specific term extraction (e.g., "medical", "legal")

**Patterns**

**🔍 What Are Patterns in NLP?**

In **Natural Language Processing (NLP)**, **patterns** refer to **rules or structures** (often defined using **regular expressions** or specific token structures) used to **identify, extract, or manipulate** specific types of text data.

**🔹 Types of Patterns in NLP**

| **Pattern Type** | **Used For** | **Example** |
| --- | --- | --- |
| **Regex (Regular Expression)** | Finding text patterns like dates, emails, etc. | \d{4}-\d{2}-\d{2} → matches 2025-07-02 |
| **POS Patterns** | Matching part-of-speech sequences | Adjective + Noun: e.g., "great movie" |
| **Entity Patterns** | Extracting named entities | "Dr. APJ Abdul Kalam" → [Title][Name] |
| **Linguistic Rules** | Rule-based pattern matching (e.g., spaCy Matcher) | Match "buy" followed by a product name |

**🔧 Example 1: Regex Pattern Matching**

python

CopyEdit

import re

text = "My email is example123@gmail.com and phone is 9876543210."

pattern = r"\b[\w.-]+@[\w.-]+\.\w+\b" # pattern to match email

emails = re.findall(pattern, text)

print(emails)

✅ Output:

css

CopyEdit

['example123@gmail.com']

**🔧 Example 2: Pattern Matching in spaCy using Matcher**

python

CopyEdit

import spacy

from spacy.matcher import Matcher

nlp = spacy.load("en\_core\_web\_sm")

matcher = Matcher(nlp.vocab)

pattern = [{"LOWER": "machine"}, {"LOWER": "learning"}] # pattern: "machine learning"

matcher.add("ML\_PATTERN", [pattern])

doc = nlp("I am learning machine learning with Python.")

matches = matcher(doc)

for match\_id, start, end in matches:

span = doc[start:end]

print("Matched:", span.text)

✅ Output:

makefile

CopyEdit

Matched: machine learning

**🧠 Common Use Cases of Patterns**

* Email/phone number/date extraction
* Hashtag or mention detection on social media
* Custom named entity recognition
* Sentence or phrase structure matching
* Filtering spam patterns or profanity

**💡 Real-Life Analogy (Tollywood Style 🎬)**

Imagine a **film casting director** looking for:

* Actors with **mustache + sunglasses** → pattern: [mustache, sunglasses]
* Or someone saying the line **"Nenu chepta..."** → pattern: specific phrase

These are like **rules or patterns** we define in NLP to catch **who fits a role** in a sentence!

Great! Let’s explore **NER (Named Entity Recognition)** in NLP — especially with **spaCy**, and as always, we’ll add some **Tollywood-style fun** at the end! 🎬

**🧠 What is NER (Named Entity Recognition)?**

**NER** is the process of **identifying and categorizing named entities** in text — such as:

| **Entity Type** | **Examples** |
| --- | --- |
| PERSON | "Mahesh Babu", "Pawan Kalyan" |
| ORG | "T-Series", "Netflix" |
| GPE | "Hyderabad", "India", "USA" |
| DATE | "July 2, 2025", "tomorrow" |
| TIME | "2 PM", "midnight" |
| MONEY | "₹1 crore", "$100" |
| LOC | "Bay of Bengal", "KBR Park" |
| PRODUCT | "iPhone", "Audi", "Samsung TV" |
| EVENT | "Oscar Awards", "IPL Final" |

**✅ spaCy NER Code Example**

python

CopyEdit

import spacy

nlp = spacy.load("en\_core\_web\_sm")

text = "Mahesh Babu signed a ₹100 crore deal with Netflix in Hyderabad on July 1st, 2025."

doc = nlp(text)

for ent in doc.ents:

print(f"{ent.text:<25} → {ent.label\_:<10} → {spacy.explain(ent.label\_)}")

**✅ Output:**

pgsql

CopyEdit

Mahesh Babu → PERSON → People, including fictional

₹100 crore → MONEY → Monetary values, including unit

Netflix → ORG → Companies, agencies, institutions

Hyderabad → GPE → Countries, cities, states

July 1st, 2025 → DATE → Absolute or relative dates or periods

**🎯 How NER Works Behind the Scenes**

spaCy uses:

* **Statistical models** (trained on labeled data)
* Context of the word
* Token’s shape, prefix, suffix
* POS tags and dependencies

You can **custom-train** it for domain-specific entities (e.g., “Blockbuster Rating”, “Superstar”, “Villain”).

**🛠️ Use Cases of NER**

* 💼 Resume parsing (extract names, skills, orgs)
* 💡 Chatbots (detect location, product, date)
* 🔎 Search engines (query understanding)
* 📊 Financial reports (detect currency, date, orgs)
* 📰 News classification (tag people/places/events)

**🎬 Tollywood Analogy: NER is like a Filmfare Jury**

NER acts like a **jury** reading a Tollywood article and tagging:

**"Allu Arjun's Pushpa 2 released in Hyderabad in May 2025 and grossed ₹300 crore at the box office."**

| **Text** | **NER Label** |
| --- | --- |
| Allu Arjun | PERSON |
| Pushpa 2 | WORK\_OF\_ART (or PRODUCT) |
| Hyderabad | GPE |
| May 2025 | DATE |
| ₹300 crore | MONEY |

👉 Like giving awards to best **Actor (PERSON)**, **Film (WORK\_OF\_ART)**, **Location (GPE)**.

**🔍 Bonus: Filter Only Specific Entity Types**

python

CopyEdit

people = [ent.text for ent in doc.ents if ent.label\_ == "PERSON"]

dates = [ent.text for ent in doc.ents if ent.label\_ == "DATE"]

**🌍 Multilingual NER**

If you want to run NER on **Telugu or Hindi**, use:

bash

CopyEdit

python -m spacy download xx\_ent\_wiki\_sm

Then use:

python

CopyEdit

nlp = spacy.load("xx\_ent\_wiki\_sm")

But accuracy may be limited — for better performance, use:

* [IndicNLP](https://github.com/AI4Bharat/indicnlp_corpus)
* [iNLTK](https://github.com/goru001/inltk)
* Indic-Transformers

**🎯 What is Bag of Words (BoW)?**

It's a technique to convert **text** into **numerical vectors**, based on **word frequency**.  
It **ignores grammar** and word order, and focuses only on the **presence/count** of words.

**🧠 Simple Example (Plain NLP):**

Let’s say we have 2 movie reviews:

* Review 1: "Allu Arjun is stylish"
* Review 2: "Ram Charan is powerful"

**Step 1: Build vocabulary (unique words)**

css

CopyEdit

["Allu", "Arjun", "is", "stylish", "Ram", "Charan", "powerful"]

**Step 2: Represent each sentence as a vector (frequency of each word)**

| **Word** | **Review 1** | **Review 2** |
| --- | --- | --- |
| Allu | 1 | 0 |
| Arjun | 1 | 0 |
| is | 1 | 1 |
| stylish | 1 | 0 |
| Ram | 0 | 1 |
| Charan | 0 | 1 |
| powerful | 0 | 1 |

This is your **BoW representation**. Machines now understand the text numerically!

**🎬 Tollywood Analogy – Understanding BoW with Tollywood Movies**

Imagine you're a **fan club manager** of multiple Tollywood stars. You get **fan letters** daily describing their favorite actors.

Each letter talks about:

* **Hero names** (e.g., "Mahesh", "NTR")
* **Traits** (e.g., "mass", "class", "dancer", "action king")

You don’t read the full letter. You just:

1. **List all unique keywords** (mass, class, dancer, etc.)
2. **Count how many times each word is used per letter**

Now every fan letter becomes a **numeric report**:

* Letter 1: "Pawan is mass and stylish"  
  → {"Pawan":1, "mass":1, "stylish":1}
* Letter 2: "NTR is mass and dancer"  
  → {"NTR":1, "mass":1, "dancer":1}

👉 You’ve converted **Tollywood fan letters** into **Bag of Words Vectors**!

**🔍 Summary (Using Tollywood Style)**

| **Term** | **Explanation** |
| --- | --- |
| Vocabulary | All unique Tollywood keywords (mass, stylish, dancer, etc.) |
| Document | One fan letter or movie dialogue |
| BoW Vector | Count of each keyword in the letter |
| Meaning | Focus is not on story, but on **what words appear and how often** |

**💡 How to explain in interview?**

**"BoW is like summarizing Tollywood fan mails by counting how many times fans praise heroes like ‘mass’, ‘class’, or ‘dancer’. We ignore grammar or sentence flow and only care about keyword frequency — this helps ML models learn patterns from text!"**

**Bag of Words (BoW)**

**🎯 What is Bag of Words (BoW)?**

It's a technique to convert **text** into **numerical vectors**, based on **word frequency**.  
It **ignores grammar** and word order, and focuses only on the **presence/count** of words.

**🧠 Simple Example (Plain NLP):**

Let’s say we have 2 movie reviews:

* Review 1: "Allu Arjun is stylish"
* Review 2: "Ram Charan is powerful"

**Step 1: Build vocabulary (unique words)**

css

CopyEdit

["Allu", "Arjun", "is", "stylish", "Ram", "Charan", "powerful"]

**Step 2: Represent each sentence as a vector (frequency of each word)**

| **Word** | **Review 1** | **Review 2** |
| --- | --- | --- |
| Allu | 1 | 0 |
| Arjun | 1 | 0 |
| is | 1 | 1 |
| stylish | 1 | 0 |
| Ram | 0 | 1 |
| Charan | 0 | 1 |
| powerful | 0 | 1 |

This is your **BoW representation**. Machines now understand the text numerically!

**🎬 Tollywood Analogy – Understanding BoW with Tollywood Movies**

Imagine you're a **fan club manager** of multiple Tollywood stars. You get **fan letters** daily describing their favorite actors.

Each letter talks about:

* **Hero names** (e.g., "Mahesh", "NTR")
* **Traits** (e.g., "mass", "class", "dancer", "action king")

You don’t read the full letter. You just:

1. **List all unique keywords** (mass, class, dancer, etc.)
2. **Count how many times each word is used per letter**

Now every fan letter becomes a **numeric report**:

* Letter 1: "Pawan is mass and stylish"  
  → {"Pawan":1, "mass":1, "stylish":1}
* Letter 2: "NTR is mass and dancer"  
  → {"NTR":1, "mass":1, "dancer":1}

👉 You’ve converted **Tollywood fan letters** into **Bag of Words Vectors**!

**🔍 Summary (Using Tollywood Style)**

| **Term** | **Explanation** |
| --- | --- |
| Vocabulary | All unique Tollywood keywords (mass, stylish, dancer, etc.) |
| Document | One fan letter or movie dialogue |
| BoW Vector | Count of each keyword in the letter |
| Meaning | Focus is not on story, but on **what words appear and how often** |

**💡 How to explain in interview?**

**"BoW is like summarizing Tollywood fan mails by counting how many times fans praise heroes like ‘mass’, ‘class’, or ‘dancer’. We ignore grammar or sentence flow and only care about keyword frequency — this helps ML models learn patterns from text!"**

**🔡 1. CountVectorizer**

**📌 What It Does:**

* Converts text into a matrix based on **word counts** (how many times each word appears in a sentence/document).
* Simple and fast.

**🧠 Formula:**

Just counts the **number of times** a word appears.

Example: “Pawan is power star” → {pawan: 1, is: 1, power: 1, star: 1}

**🎬 Tollywood Analogy:**

Imagine fan letters where the number of times a name appears = fan obsession.

* One fan writes "Allu Arjun is stylish stylish stylish"  
  → So the count for "stylish" is 3.  
  This shows *frequency*, not *importance*.

**✅ Use When:**

* You want **raw frequency** info.
* You’re working with short texts or spam detection.

**🧮 2. TfidfVectorizer (Term Frequency – Inverse Document Frequency)**

**📌 What It Does:**

* Like CountVectorizer but **gives less importance to very common words** and more to **rare but meaningful words**.

**🧠 Formula:**

TF-IDF = TF (term frequency) × IDF (inverse document frequency)

* TF: How often a word occurs in the document.
* IDF: How **rare** the word is across all documents.

Example:  
“is”, “the” → used everywhere → IDF is low → TF-IDF score becomes low  
“Baahubali” → appears in just one document → IDF is high → TF-IDF is high

**🎬 Tollywood Analogy:**

Let’s say:

* Every fan letter says “is”, “mass” → common.
* One letter says “Baahubali” → very specific.

TF-IDF understands that *“Baahubali” is more unique*, so gives it more weight.

**✅ Use When:**

* You want to highlight **important & rare words**.
* You're building a **text classifier** or **search engine**.

**🔁 Summary Table**

| **Feature** | **CountVectorizer** | **TfidfVectorizer** |
| --- | --- | --- |
| Value Used | Word count | Weighted word score (based on frequency + rarity) |
| Ignores Common Words? | ❌ No | ✅ Yes (through IDF) |
| Output Type | Integer counts | Float weights |
| Analogy | Count how many times “mass” appears | Give importance to rare terms like “Baahubali” |
| Use When | Raw frequency is enough | Meaning matters more than repetition |

**Embeddings**

**What Are Embeddings?**

**Embeddings** are dense vector representations of words or phrases that **capture their meanings, context, relationships**, and even analogies.

Unlike Bag of Words or TF-IDF, embeddings:

* Understand similarity: *“king” is close to “queen”*
* Understand context: *“Apple” (fruit) vs. “Apple” (company)*

**🔑 Why Use Advanced Embeddings?**

| **Feature** | **Traditional (BoW/TF-IDF)** | **Embeddings (Word2Vec, BERT)** |
| --- | --- | --- |
| Context-aware? | ❌ No | ✅ Yes |
| Dimensionality | 🔺 High (sparse) | 🔻 Low (dense) |
| Semantic similarity | ❌ Not captured | ✅ Captured |
| Word order? | ❌ Ignored | ✅ Partially or fully handled |

**🌟 Popular Types of Embeddings**

| **Technique** | **Description** | **Analogy** |
| --- | --- | --- |
| **Word2Vec** | Learns word meanings from surrounding words (context window) | Like a friend guessing your mood based on the friends around you |
| **GloVe** | Learns from **global** co-occurrence statistics | Like reading an entire movie script to know how often "hero" and "action" appear together |
| **FastText** | Like Word2Vec, but uses subword info (handles unknown words better) | “stylish” → includes “style”, “styl”, “ish” |
| **BERT** (Transformer-based) | Contextual embeddings — same word has different vectors in different sentences | “Bank” in "river bank" ≠ "money bank" |
| **ELMo** | Similar to BERT but based on LSTMs | Bidirectional understanding of context |

**📌 Technical Overview**

**🔹 1. Word2Vec (Google)**

Two architectures:

* **CBOW** (predicts current word from context)
* **Skip-gram** (predicts context from current word)

python

CopyEdit

from gensim.models import Word2Vec

sentences = [["allu", "arjun", "is", "stylish"],

["mahesh", "babu", "is", "superstar"]]

model = Word2Vec(sentences, vector\_size=100, window=3, min\_count=1, sg=1)

print(model.wv['stylish']) # 100-d vector

**🔹 2. GloVe (Stanford)**

Uses a matrix of **global word co-occurrence counts**.

Pretrained GloVe vectors:

bash

CopyEdit

wget http://nlp.stanford.edu/data/glove.6B.zip

Use with gensim or directly load into models.

**🔹 3. FastText (Facebook)**

Handles misspelled and rare words better.

python

CopyEdit

from gensim.models import FastText

model = FastText(sentences, vector\_size=100, window=3, min\_count=1)

print(model.wv['stylish']) # Even 'stylishly' will work!

**🔹 4. BERT Embeddings (HuggingFace Transformers)**

python

CopyEdit

from transformers import BertTokenizer, BertModel

import torch

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

model = BertModel.from\_pretrained('bert-base-uncased')

text = "Allu Arjun is stylish star."

inputs = tokenizer(text, return\_tensors='pt')

outputs = model(\*\*inputs)

# Get [CLS] token vector (sentence-level)

embedding = outputs.last\_hidden\_state[:, 0, :]

**🎬 Tollywood Analogy – Understanding Word Embeddings**

Imagine each **actor** in a movie has a **personality vector**.

| **Actor** | **Traits** |
| --- | --- |
| Mahesh Babu | Classy, calm, action |
| Allu Arjun | Stylish, dance, mass |
| Jr. NTR | Mass, emotional, fierce |
| Ram Charan | Powerful, versatile |

A model learns that:

* “Stylish” is near “Allu Arjun”
* “Powerful” is near “Ram Charan”
* “Superstar” is similar to “Mahesh Babu”

👉 This is exactly what **embeddings** do — convert words into vector space where **similar meanings are closer**.

**🛠 Real Use Cases**

* Search engines (similar query understanding)
* Chatbots (understanding user intent)
* Sentiment analysis (context-aware)
* Recommendation systems (based on product descriptions)
* Document clustering, topic modeling

**🧠 Interview Line Example**

"I used Word2Vec and FastText to generate dense embeddings for Telugu movie reviews. These helped capture nuanced word relationships like ‘stylish’ and ‘Allu Arjun’ being similar. Later, I explored contextual embeddings with BERT to improve chatbot understanding.

**PARSING**

**🔶 What is Parsing in NLP?**

**Parsing** in Natural Language Processing (NLP) is the process of analyzing a sentence's **grammar structure**. It helps the computer understand how words relate to each other—like which is the subject, verb, object, etc.

There are two main types:

1. **Syntactic Parsing**: Understanding the grammatical structure.
2. **Dependency Parsing**: Understanding how words depend on each other (subject-verb-object relations).

**🎬 Tollywood Analogy:**

**Let’s say a sentence is like a Tollywood movie scene.**

Imagine you're watching a **scene from a movie** where:

* **Hero** (like Mahesh Babu) punches a **villain** (like Prakash Raj) and saves the **heroine** (like Samantha).

That’s your sentence:

**"Mahesh punched Prakash to save Samantha."**

**Now, let’s parse the scene like a scriptwriter:**

**🔹 Syntactic Parsing:**

Breaks the sentence down into grammar-based parts:

css

CopyEdit

[NP Mahesh] [VP punched [NP Prakash] [PP to [VP save [NP Samantha]]]]

* NP = Noun Phrase
* VP = Verb Phrase
* PP = Prepositional Phrase

🎬 **Analogy**: This is like dividing the **movie scene** into "Hero Entry", "Fight", and "Rescue" segments.

**🔹 Dependency Parsing:**

Builds **who does what**:

yaml

CopyEdit

punched ← subject: Mahesh

punched ← object: Prakash

save ← object: Samantha

save ← purpose of punched

🎬 **Analogy**:

* **Hero (Mahesh)** → does the **action (punch)**
* **Villain (Prakash)** → receives the punch
* **Purpose** → to **save the heroine (Samantha)**

This is like the **director’s storyboard** showing actions and dependencies in a scene.

**🧠 Why is Parsing Important?**

✅ For **chatbots**, to understand complex questions  
✅ For **machine translation**, to preserve meaning  
✅ For **text summarization**, to identify key actions  
✅ For **question answering**, to get "who did what to whom?"

**🛠 Bonus: Python Code Example**

python

CopyEdit

import spacy

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp("Mahesh punched Prakash to save Samantha.")

for token in doc:

print(f"{token.text:<10} → {token.dep\_:<10} → {token.head.text}")

Output:

css

CopyEdit

Mahesh → nsubj → punched

punched → ROOT → punched

Prakash → dobj → punched

to → aux → save

save → xcomp → punched

Samantha → dobj → save

**🔹 What is LSI in NLP?**

**LSI** stands for **Latent Semantic Indexing** (also called **Latent Semantic Analysis**, LSA).

**📌 It is a dimensionality reduction technique used in text analysis to:**

* Discover **hidden (latent) relationships** between words and documents.
* Capture **semantic meaning** beyond exact word matching.

It’s especially useful in:

* Information retrieval
* Document clustering
* Topic modeling
* Search engines

**🧠 How does it work?**

**Step-by-step:**

1. **Create a Term-Document Matrix**
   * Rows = words (terms)
   * Columns = documents
   * Values = frequency (or TF-IDF score)
2. **Apply SVD (Singular Value Decomposition)**
   * Break the matrix into 3 smaller matrices:  
     A = U × Σ × Vᵗ
   * This reduces noise and extracts **latent topics**.
3. **Reduce dimensions**
   * Keep only the top k concepts (topics).
   * Now similar words/documents appear closer in the reduced space even if they don’t share exact words.

**🎬 Tollywood Analogy for LSI**

**Imagine you're watching a Tollywood mashup video on YouTube.**

It shows scenes from:

* **Movie A**: Mahesh Babu saving a village.
* **Movie B**: Allu Arjun fighting a corrupt officer.
* **Movie C**: Jr. NTR standing against injustice.

Though they’re different movies with different dialogues, you **understand the hidden theme**:  
🔺 **"Hero fights for justice."**

That’s what LSI does!

Even if the **exact words** in the scripts are different, LSI finds that all these documents talk about the **same latent topic**: *justice*.

**📊 Python Example (using Scikit-learn):**

python

CopyEdit

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import TruncatedSVD

docs = [

"Mahesh saved the village from goons",

"Allu Arjun fought the corrupt police",

"NTR brought justice to poor people",

]

# Step 1: TF-IDF Matrix

vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(docs)

# Step 2: Apply LSI using SVD

lsa = TruncatedSVD(n\_components=2) # reduce to 2 topics

X\_lsa = lsa.fit\_transform(X)

print(X\_lsa)

This gives you 2D representations of each document capturing the **hidden topic space**.

**✅ Summary**

| **Feature** | **Description** |
| --- | --- |
| Full Form | Latent Semantic Indexing |
| Purpose | Find hidden meanings/topics in text |
| Uses | Search, Recommendation, Topic Clustering |
| Math Behind | Singular Value Decomposition (SVD) |
| Benefit | Understand similarity beyond exact words |