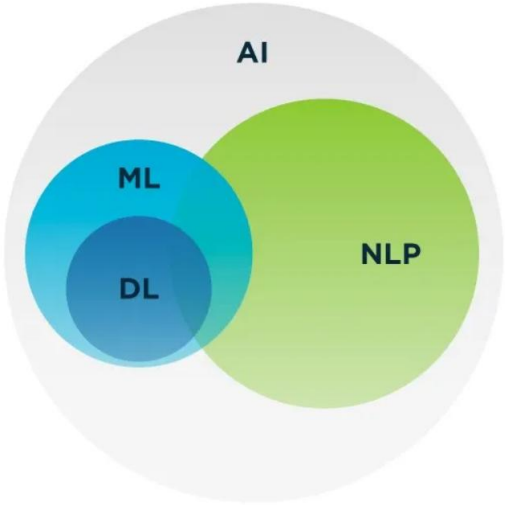
NATURAL LANGUAGE PROCESSING

- Natural Language Processing (NLP) is a field of artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful.
- It combines computational semantics, machine learning, and deep learning to process and analyze text or speech data.

"NLP evolved from rule-based systems in the 1950s (like ELIZA) to modern deep learning models such as GPT and BERT."



How NLP Works

NLP systems typically process text through several stages:

Text → Preprocessing → Feature Extraction → Modeling → Evaluation.

- **Preprocessing**: Cleaning text by removing noise (e.g., punctuation, stop words) and normalizing it (e.g., lowercasing, stemming, lemmatization).
- **Feature Extraction**: Converting text into numerical representations, like word embedding (e.g., Word2Vec, BERT).
- **Modeling**: Applying algorithms, often neural networks, to learn patterns in data for tasks like classification or translation.

- **Evaluation**: Measuring performance using metrics like accuracy, precision, recall, or F1-score.

NLP Components = Building Blocks	NLP Tasks = What You Want the NLP System To Do
These are the **fundamental processes / tools** that NLP uses to understand language. They are like the **machinery** needed to process text.	These are the **objectives / final goals** of NLP systems. They describe **what problem you want to solve** using NLP.
Tokenization POS Tagging NER Parsing Stemming Lemmatization Vectorization	Translation Sentiment Analysis Text Classification Summarization Chatbots Question Answering Information Extraction

Category	Name	What It Does	Example
NLP COMPONENTS	Tokenization	Splits text into words or sentences	"I love NLP." → ["I","love","NLP"]
	Stopword Removal	Removes common words	"I am a student" → ["student"]
	Stemming	Removes word endings	"studies" → "studi"
	Lemmatization	Converts to dictionary form	"better" → "good", "running" → "run"
	Lowercasing	Converts text to lowercase	"NLP Is Fun" → "nlp is fun"
	Punctuation Removal	Removes punctuation marks	"Hello!!!" → "Hello"
	POS Tagging	Assigns grammar labels	"Students write exams" → write(VB)
	Named Entity Recognition (NER)	Identifies people, places, dates, etc.	"Apple bought GitHub in 2018" → Apple(ORG)
	Parsing (Syntax Analysis)	Understands sentence structure	subject → verb → object

	Chunking	Groups words into phrases	"Natural Language" → Noun Phrase
	N-Grams	Creates word sequences	Bigram: "Natural Language"
	Vectorization (BoW, TF-IDF)	Converts text to numbers	"NLP" → [0.23, 0.11, ...]
	Word Embeddings	Semantic numerical representation	"king" ≈ "queen" in vector space
NLP TASKS	Text Classification	Assigns categories to text	Spam vs Not Spam
	Sentiment Analysis	Finds emotion	"I love this" → Positive
	Machine Translation	Converts between languages	"Hello" → "Namaste"
	Summarization	Shortens text meaningfully	News → 3-line summary
	Question Answering (QA)	Answers user questions	"Capital of France?" → Paris
	Chatbots / Conversational AI	Understands and responds to queries	Customer support bot
	Information Extraction	Extracts key facts	From text → Dates, Names, Amounts
	Topic Modeling	Discovers hidden topics	News → Politics / Sports / Tech
	Text Generation	Produces human-like text	ChatGPT writing a paragraph
	Speech-to-Text (ASR)	Converts voice to text	Voice typing in mobile
	Text-to-Speech (TTS)	Converts text to spoken voice	Google Assistant reading message
	Language Modeling	Predicts next word	"I am going to..." → "school"
	Named Entity Linking	Connects entities to a database	"Apple" → Apple Inc. (company)
	Document Classification	Categorizes documents	Legal/Medical/Finance

What Are N-grams in NLP?

Definition:

An **N-gram** is a sequence of **N words** occurring together in a sentence.

Why use N-grams?

- Helps the model understand **context**
- Useful in text classification, autocomplete, keyword extraction, language modeling

N-gram Type	Meaning	Example (sentence: “Natural Language Processing”)
1-gram (Unigram)	Single word	["Natural", "Language", "Processing"]
2-gram (Bigram)	Pair of words	["Natural Language", "Language Processing"]
3-gram (Trigram)	Three words	["Natural Language Processing"]
4-gram	Four words	Looks at bigger context

When to use N-grams?

- **Unigrams:** Good for basic text analysis
- **Bigrams:** Captures simple context ("machine learning")
- **Trigrams and above:** Captures phrase meaning ("New York City")

Python Code: Generate N-grams Using NLTK

```
import nltk
from nltk.util import ngrams
from nltk.tokenize import word_tokenize

sentence = "Natural Language Processing is interesting"
tokens = word_tokenize(sentence)

# Unigrams
unigrams = list(ngrams(tokens, 1))
print("Unigrams:", unigrams)

# Bigrams
bigrams = list(ngrams(tokens, 2))
print("Bigrams:", bigrams)

# Trigrams
trigrams = list(ngrams(tokens, 3))
print("Trigrams:", trigrams)
```

Python Code: Using SpaCy

```
import spacy
from spacy.util import filter_spans

nlp = spacy.load("en_core_web_sm")

doc = nlp("Natural Language Processing is interesting")
tokens = [token.text for token in doc]

# Function to generate n-grams
def generate_ngrams(tokens, n):
    return [" ".join(tokens[i:i+n]) for i in range(len(tokens)-n+1)]

print("Bigrams:", generate_ngrams(tokens, 2))
print("Trigrams:", generate_ngrams(tokens, 3))
```

Python Code: N-grams With TF-IDF (scikit-learn)

```
from sklearn.feature_extraction.text import TfidfVectorizer

texts = [
    "I love natural language processing",
    "language processing is fun"
]

vectorizer = TfidfVectorizer(ngram_range=(1,2))  # Unigrams + Bigrams
X = vectorizer.fit_transform(texts)

print("Features (Vocabulary):")
print(vectorizer.get_feature_names_out())

print("\nTF-IDF Matrix:")
print(X.toarray())
```

Word Cloud for Bigrams

```
from wordcloud import WordCloud

bigram_text = " ".join(bigram_list)

wc = WordCloud(width=600, height=400).generate(bigram_text)

plt.imshow(wc)
plt.axis("off")
plt.title("Bigram Word Cloud")
plt.show()
```

Applications of NLP

NLP is used in almost every digital platform today.

a) Chatbots & Virtual Assistants

Google Assistant, Siri, Alexa, Customer service bots.

b) Machine Translation

Google Translate, Bing Translate.

c) Spam Detection

Email filtering in Gmail, Outlook.

d) Sentiment & Opinion Analysis

Product reviews, social media analysis, brand monitoring.

e) Search Engines

Understanding search intent.

("apple benefits" → fruit or company?)

f) Document Processing

Resume scanning, invoice extraction, automated summarization.

g) Predictive Text & Auto-correction

Mobile keyboards, Gmail Smart Compose.

h) Speech Recognition

Voice-to-text systems.

i) Content Moderation

Detecting harmful or abusive language.

Challenges in NLP

Human language is complex, and machines struggle with many aspects.

a) Ambiguity

Words or sentences may have multiple meanings.

Example: "I saw the man with the telescope."

b) Sarcasm & Emotion

Machines find it difficult to detect sarcasm or hidden emotions.

Example: "Oh great, my phone broke again."

c) Evolving Language

New slang, emojis, internet language evolve rapidly.

Example: "lit", "vibes", "LOL", "🔥"

d) Multilingual & Dialects

Different accents, spellings, and grammatical structures.

Example: colour vs color, Hinglish, Kannada-English mix.

e) Noise in Real Data

Tweets, chats, and social media text contain:

- Spelling mistakes
- Short forms
- Emojis
- Hashtags

f) Bias in Training Data

If training data is biased, the NLP model becomes biased.

Example: Gender or racial stereotypes appearing in predictions.

g) Context Understanding

Long sentences, references, and implied meaning are hard for machines.

Example: "John went home. He was tired." (He = John)

NLP Workflow

The **NLP Workflow** is the step-by-step process used to convert raw text into meaningful insights using Natural Language Processing.

It consists of **7 main stages**:

```
Raw Text
   ↓
Data Preprocessing
   ↓
Feature Extraction
   ↓
Model Training
   ↓
Model Evaluation
   ↓
Deployment & Monitoring
```

📄 1. Problem Definition

Identify what you want the NLP system to do.

Examples:

- Classify customer reviews as positive/negative
- Build a chatbot
- Summarize text
- Translate text

❓ *Why important?*

The entire workflow depends on the final goal.

📄 2. Data Collection

Gather text data from various sources.

Sources:

- Websites (web scraping)
- Social media posts
- Product reviews
- Documents (PDFs, articles)
- Open datasets (IMDB, CoNLL, News datasets)

 Example: Download reviews dataset for sentiment analysis.

---

3. Data Preprocessing (Cleaning)

Prepare raw text for analysis.

Includes:

- Lowercasing
- Removing stopwords ("the", "is")
- Removing punctuation
- Tokenization
- Stemming / Lemmatization
- Removing numbers, emojis, URLs

 Example:

"THE product WAS amazing!!! 😍" → "product amazing"

---

4. Feature Extraction / Representation

Convert text into numerical form so ML models can understand it.

Techniques:

- **Bag of Words (BoW)**
- **TF-IDF**
- **Word Embeddings (Word2Vec, GloVe, FastText)**
- **Contextual Embeddings (BERT)**

 Example:

Text → TF-IDF vector → fed to ML model.

📄 5. Model Selection and Training

Choose and train an NLP model on the prepared data.

Model types:

Traditional ML:

- Naive Bayes
- Logistic Regression
- SVM

Deep Learning:

- RNN, LSTM
- Transformers (BERT, GPT)

🔍 *Example:*

Train a classifier to label reviews as positive or negative.

📄 6. Model Evaluation

Check how well the model performs.

Metrics:

- Accuracy
- Precision, Recall
- F1-score
- BLEU (for translation)
- ROUGE (for summarization)
- Perplexity (for language models)

🔍 *Example:*

Classification accuracy = 89%

📄 7. Deployment & Monitoring

Deploy the NLP model for real-world use.

Methods:

- Web API (FastAPI, Flask)

- Cloud deployment (AWS, GCP, Azure)
- Mobile app integration
- Chatbot frameworks

Monitoring includes:

- Checking errors
- Updating model with new data
- Ensuring fairness & reducing bias

 Example:

Deploying a sentiment classifier on a customer feedback portal.

Python Code

Data Collection (Example using a small dataset)

```
# Sample dataset
texts = [
    "I love this product",
    "This is the worst thing ever",
    "Absolutely amazing experience",
    "I hate this item"
]

labels = [1, 0, 1, 0]   # 1 = Positive, 0 = Negative
```

Data Preprocessing (Cleaning + Tokenization + Stopwords + Lemmatization)

```
import spacy
nlp = spacy.load("en_core_web_sm")

def preprocess(text):
    doc = nlp(text.lower())
    clean_tokens = []
    for token in doc:
        if token.is_stop or token.is_punct:
            continue
        clean_tokens.append(token.lemma_)  # Lemmatization
    return " ".join(clean_tokens)

# Test preprocessing
for t in texts:
    print("Original:", t)
    print("Cleaned :", preprocess(t))
    print()
```

Feature Extraction (TF-IDF Vectorization)

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform([preprocess(t) for t in texts])

print("Vocabulary:", vectorizer.get_feature_names_out())
print("TF-IDF Matrix:\n", X.toarray())
```

Model Selection & Training (Logistic Regression)

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X, labels)

print("Model trained successfully!")
```

Model Evaluation (Accuracy, F1-score)

```
from sklearn.metrics import accuracy_score, f1_score

y_pred = model.predict(X)

print("Accuracy:", accuracy_score(labels, y_pred))
print("F1-Score:", f1_score(labels, y_pred))
```

Deployment Simulation (Predict on New Text)

```
# New text to classify
new_text = "I really love this"
processed = preprocess(new_text)
vectorized = vectorizer.transform([processed])

prediction = model.predict(vectorized)
print("Prediction (1 = Positive, 0 = Negative):", prediction[0])
```

End-to-End Pipeline (ALL Steps Together)

```
from sklearn.pipeline import Pipeline

nlp_pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(preprocessor=preprocess)),
    ("clf", LogisticRegression())
])

# Train
nlp_pipeline.fit(texts, labels)

# Predict
print(nlp_pipeline.predict(["The product is awesome"]))
print(nlp_pipeline.predict(["I dislike this"]))
```

Data Collection using Web Scraping (BeautifulSoup)

Install dependencies:

```
pip install requests beautifulsoup4 spacy scikit-learn
python -m spacy download en_core_web_sm
```

Step 1: Web Scraping Example (Data Collection)

We will scrape text (example: quotes from a sample website).

```
import requests
from bs4 import BeautifulSoup

url = "https://quotes.toscrape.com/"   # sample site for scraping

response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

# Collect all text quotes
texts = [quote.text for quote in soup.find_all("span", class_="text")]

# Assign dummy labels for demonstration
labels = [1 if "life" in t.lower() else 0 for t in texts]   # Just a sample rule

print("Scraped Texts:")
for t in texts:
    print("-", t)
```

Step 2: Preprocessing (Cleaning + Lemmatization)

```
import spacy
nlp = spacy.load("en_core_web_sm")

def preprocess(text):
    doc = nlp(text.lower())
    clean_tokens = []
    for token in doc:
        if token.is_stop or token.is_punct or token.like_num:
            continue
        clean_tokens.append(token.lemma_)
    return " ".join(clean_tokens)

cleaned_texts = [preprocess(t) for t in texts]
print("Cleaned:", cleaned_texts[:5])
```

Step 3: Feature Extraction (TF-IDF)

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_texts)

print("Vocabulary Size:", len(vectorizer.get_feature_names_out()))
```

Step 4: Model Training (Logistic Regression)

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X, labels)

print("Model Trained Successfully!")
```

Step 5: Model Evaluation

```
from sklearn.metrics import accuracy_score, f1_score

y_pred = model.predict(X)

print("Accuracy :", accuracy_score(labels, y_pred))
print("F1-Score:", f1_score(labels, y_pred))
```

Step 6: Deployment Simulation (Prediction)

```
sample = "Life is beautiful but challenging"
sample_clean = preprocess(sample)
sample_vec = vectorizer.transform([sample_clean])

prediction = model.predict(sample_vec)
print("Prediction:", prediction[0])
```

Step 7: End-to-End NLP Pipeline (with Web Scraping)

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(preprocessor=preprocess)),
    ("clf", LogisticRegression())
])

pipeline.fit(texts, labels)

print(pipeline.predict(["Life teaches great lessons"]))
print(pipeline.predict(["This is boring"]))
```

Dataset: Amazon Alexa Reviews

```
import pandas as pd
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import Pipeline
from collections import Counter
import matplotlib.pyplot as plt

#Load dataset
df = pd.read_csv("amazon_alexa.tsv", sep="\t")
```

```
df.head()

#Check Data Info
print(df.shape)
print(df.feedback.value_counts())

#Text Preprocessing (spaCy)

nlp = spacy.load("en_core_web_sm")

def preprocess(text):
    doc = nlp(text.lower())
    clean_words = []
    for token in doc:
        if token.is_stop or token.is_punct or token.like_num:
            continue
        clean_words.append(token.lemma_)
    return " ".join(clean_words)

df["cleaned"] = df["verified_reviews"].apply(preprocess)
df.head()

#Feature Extraction (TF-IDF)

tfidf = TfidfVectorizer()
X = tfidf.fit_transform(df.cleaned)
y = df.feedback

#Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

#Train a Sentiment Classifier (Logistic Regression)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

#Evaluate the Model
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Predict Sentiment for New Alexa Reviews
def predict_sentiment(review):
    cleaned = preprocess(review)
    vector = tfidf.transform([cleaned])
    pred = model.predict(vector)[0]
    return "Positive ☺" if pred == 1 else "Negative ☹"

print(predict_sentiment("Alexa is awesome, I use it everyday!"))
print(predict_sentiment("It stopped working after a week"))

#Build a Full Pipeline (Preprocessing → TF-IDF → Classifier)
```

```
pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(preprocessor=preprocess)),
    ("clf", LogisticRegression(max_iter=1000))
])

pipeline.fit(df.verified_reviews, df.feedback)

print(pipeline.predict(["Alexa is very useful"]))
print(pipeline.predict(["Terrible product, waste of money"]))

#Visualize Word Frequencies (Positive vs Negative)
positive_words = " ".join(df[df.feedback==1]["cleaned"]).split()
negative_words = " ".join(df[df.feedback==0]["cleaned"]).split()

pos_counts = Counter(positive_words).most_common(20)
neg_counts = Counter(negative_words).most_common(20)

pos_df = pd.DataFrame(pos_counts, columns=["word","count"])
neg_df = pd.DataFrame(neg_counts, columns=["word","count"])

plt.bar(pos_df.word, pos_df.count)
plt.xticks(rotation=45)
plt.title("Top Positive Words")
plt.show()

plt.bar(neg_df.word, neg_df.count)
plt.xticks(rotation=45)
plt.title("Top Negative Words")
plt.show()
```

Using nltk

```
import pandas as pd
import nltk
nltk.download('punkt')
nltk.download('stopwords')

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import Pipeline


stopwords_en = set(stopwords.words('english'))
ps = PorterStemmer()

df = pd.read_csv("amazon_alexa.tsv", sep="\t")
```

```
df.head()

def preprocess(text):
    # Lowercase
    text = text.lower()

    # Tokenize
    tokens = word_tokenize(text)

    clean_words = []

    for token in tokens:
        # Remove stopwords, punctuation, numbers
        if token in stopwords_en or token in string.punctuation or token.isnumeric():
            continue

        # Stemming (convert to root form)
        clean_words.append(ps.stem(token))

    return " ".join(clean_words)
df["cleaned"] = df["verified_reviews"].apply(preprocess)
df.head()

#TF-IDF Vectorization
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(df.cleaned)

y = df.feedback

#Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

#Train Sentiment Classifier
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

#Model Evaluation
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Predict Sentiment for New Alexa Reviews
def predict_sentiment(review):
    cleaned = preprocess(review)
    vector = tfidf.transform([cleaned])
    pred = model.predict(vector)[0]
    return "Positive ☺" if pred == 1 else "Negative ☹"

print(predict_sentiment("Alexa is awesome"))
print(predict_sentiment("Stopped working after one week"))

#Full Pipeline Using NLTK (One-Step)
```

```
pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(preprocessor=preprocess)),
    ("clf", LogisticRegression(max_iter=1000))
])

pipeline.fit(df.verified_reviews, df.feedback)

print(pipeline.predict(["Alexa is amazing, totally love it"]))
print(pipeline.predict(["Worst purchase ever"]))
```