What is NLP?

What is NLP?

NLP is a field of AI that helps computers **understand, interpret, and respond to human language**, just like we do. It allows machines to process text and speech, making sense of it in a meaningful way.

Think of how **Google Search**, **ChatGPT**, or **voice assistants like Siri** understand your words and respond – that's NLP in action!

Problem Statement:

Imagine you are working for a social media company. The company is concerned with the growing amount of fake news circulatng on it's platform. They have assigned you to investigate how fake news can be recognized and create a method of identifying it.

Fake_News_Data_Sample.csv

id	title	text	date	fake_or_factual
101	CELEBRITY OUTRAGE: Susan Sarandon's View Sparks Controversy	A heated debate erupts over recent comments...	Dec 30, 2015	Fake News
102	Elijah Cummings Criticizes Trump's Policies	In a recent interview, Elijah Cummings expressed concerns...	April 6, 2017	Fake News
103	Hillary Clinton Discusses Women's Role in Cabinet	Clinton highlights the importance of gender diversity...	April 26, 2016	Fake News
20	Russian bombing of U.S.-backed forces being discussed	WASHINGTON (Reuters) - U.S. Defense Secretary ...	September 18, 2017	Factual News
41	Britain says window to restore Northern Ireland narrows	BELFAST (Reuters) - Northern Ireland's political...	September 4, 2017	Factual News

Steps in the NLP Pipeline

1. **Data Collection:** We start by collecting news data — this can include headlines, articles, and sources.
2. **Data Cleaning:** Text data is often messy, containing special characters, numbers, and irrelevant information, so we clean it up.
3. **Exploratory Data Analysis (EDA):** We visualise word frequency, patterns, and common words in fake vs. real news.
4. **Feature Engineering:** We convert text into a format that a machine learning model can understand, like **TF-IDF** or **word embeddings.**
5. **Model Training:** We train machine learning models to classify the text as fake or real.
6. **Evaluation:** Finally, we test how well our model performs using accuracy, precision, and recall.

***Term Frequency (TF):** How **frequently** a term appears in a document.
***Inverse Document Frequency (IDF):** How **rare** a terms is in a doc.

Key Terminology in NLP (Explained Simply)

1. **Tokenization:**
    - Breaking text into smaller pieces like words or sentences.
    - Example: `"I love AI"` → `["I", "love", "AI"]`
2. **Stop Words:**
    - Common words like “is”, “the”, “in” that don't carry much meaning.
    - These are usually removed to focus on important words.
3. **Stemming and Lemmatization:**
    - Simplifying words to their root form.
    - Example: "running" → "run" (stemmed)

4. **Named Entity Recognition (NER):**
    - Finding and categorizing names in text like people, places, dates.
    - Example: "Elon Musk founded Tesla" → `[Person: Elon Musk], [Organization: Tesla]`
5. **Sentiment Analysis:**
    - Understanding if a text is positive, negative, or neutral.
    - Example: "I love this product!" → Positive sentiment.
6. **Parts of Speech (POS) Tagging:**
    - Identifying words as nouns, verbs, adjectives, etc.
    - Example: "The cat sat on the mat." → `[The: Article, cat: Noun, sat: Verb]`
7. **Bag of Words (BoW):**
    - Counting word occurrences without worrying about order.
    - Example: `"I love NLP. NLP is great."` → `{I: 1, love: 1, NLP: 2, is: 1, great: 1}`

8. **TF-IDF (Term Frequency-Inverse Document Frequency):**
    - A scoring method to find important words in a document.
9. **Word Embeddings:**
    - Converting words into numerical values to find meaning.
    - Example: `"king – man + woman = queen"`
10. **Text Classification:**

- Categorizing text into groups like spam/non-spam, news topics, etc.

1. Tokenization (Breaking Text into Units)

Definition:

Tokenization is the process of breaking down text into smaller units called **tokens**, which can be words, subwords, or characters, depending on the tokenization method used.

Purpose:

- Converts raw text into manageable pieces for further processing.
- Prepares the text for model inputs by breaking it into meaningful components.

Types of Tokenization:

1. **Word Tokenization:** Splits text into words (e.g., `"I love AI"` → `["I", "love", "AI"]`).
2. **Subword Tokenization (e.g., Byte-Pair Encoding, WordPiece):** Breaks words into smaller subunits to handle out-of-vocabulary words (e.g., `"unhappiness"` → `["un", "happiness"]`).
3. **Character Tokenization:** Breaks text into characters (e.g., `"hello"` → `["h", "e", "l", "l", "o"]`).

Example: Input text:

`"Transformers are amazing!"`

Tokenized output:

`["Transformers", "are", "amazing", "!"]`

2. Embedding (Converting Tokens to Numerical Representations)

Definition:

Embedding is the process of converting tokens (or token IDs) into dense numerical vector representations that capture their meaning in a continuous space.

Purpose:

- Represents words/subwords in a way that encodes semantic meaning and context.
- Helps the model understand relationships between words (e.g., similarity, analogy).

Common Types of Embeddings:

1. **Pre-trained Embeddings:** Word2Vec, GloVe (static word vectors).
2. **Contextual Embeddings:** BERT, GPT (dynamic, context-aware vectors).
3. **Learned Embeddings:** Models learn custom embeddings during training.

Example:

Tokenized text:

```
["Transformers", "are", "amazing", "!"]
```

Embedding output (for each token):

```
[ [0.21, 0.34, 0.56], [0.45, 0.78, 0.12], [0.89, 0.23, 0.44], [0.02, 0.19, 0.99] ]
```

How can we capture similarities between two words?

Bedroom: 3
Area: 1850sq ft
Bathrooms: 3

Bedroom: 3
Area: 1700sq ft
Bathrooms: 2

Bedroom: 3
Area: 1850sq ft
Bathrooms: 3

Bedroom: 3
Area: 1700sq ft
Bathrooms: 2

Bedroom: 10
Area: 7500sq ft
Bathrooms: 3

Dhoni

Cummins

Australia

just think about is there a way to retrieve features from these words ?

Dhoni

Person: 1
Healthy/fit: 0.9
Location: 0
Has two eyes: 1
Has government: 0

Cummins

Person: 1
Healthy/fit: 0.87
Location: 0
Has two eyes: 1
Has government: 0

Australia

Person: 0
Healthy/fit: 0.7
Location: 1
Has two eyes: 0
Has government: 1

Dhoni

Cummins

Australia

$$\begin{Bmatrix} 1 \\ 0.9 \\ 0 \\ 1 \\ 0 \end{Bmatrix}$$

$$\begin{Bmatrix} 1 \\ 0.87 \\ 0 \\ 1 \\ 0 \end{Bmatrix}$$

$$\begin{Bmatrix} 0 \\ 0.7 \\ 1 \\ 0 \\ 1 \end{Bmatrix}$$

Feature	Rose	Cactus	Desert
Has Leaves	1.0	0.8	0.0
Requires Water	0.9	0.4	0.1
Thrives in Sun	0.7	1.0	1.0
Has Thorns	0.2	1.0	0.0

"king - man + woman = queen"

Word	Dimension 1	Dimension 2	Dimension 3	Dimension 4	Dimension 5	...
King	0.9	0.8	0.3	0.5	0.7	...
Man	0.7	0.6	0.2	0.4	0.5	...
Woman	0.8	0.7	0.3	0.6	0.6	...
Queen	1.0	0.9	0.4	0.6	0.8	...

Vector Calculation:

The analogy works as follows in vector space:

$$\text{King} - \text{Man} + \text{Woman} \approx \text{Queen}$$

Let's apply this to a few dimensions:

$$(0.9, 0.8, 0.3, 0.5, 0.7) - (0.7, 0.6, 0.2, 0.4, 0.5) + (0.8, 0.7, 0.3, 0.6, 0.6)$$

$$= (0.9 - 0.7 + 0.8, 0.8 - 0.6 + 0.7, 0.3 - 0.2 + 0.3, 0.5 - 0.4 + 0.6, 0.7 - 0.5 + 0.6)$$

$$= (1.0, 0.9, 0.4, 0.6, 0.8)$$

As a result, the computed vector is very close to the actual vector for **Queen**, demonstrating how word embeddings encode gender relationships and semantic similarities.

“Tree – Bush + Flower ≈ Shrub”

Plant	Height	Water Requirement	Sunlight Preference	Leaf Size	Growth Rate
Tree	9.0	6.0	8.0	7.0	5.0
Bush	5.0	7.0	6.0	6.0	7.0
Flower	4.0	8.0	7.0	9.0	6.0
Calculation (Tree - Bush + Flower)	(9.0 - 5.0 + 4.0)	(6.0 - 7.0 + 8.0)	(8.0 - 6.0 + 7.0)	(7.0 - 6.0 + 9.0)	(5.0 - 7.0 + 6.0)
Result (Shrub)	8.0	7.0	9.0	10.0	4.0

Comparison of Tokenization vs. Embedding

Feature	Tokenization	Embedding
Purpose	Splits text into tokens	Converts tokens into numerical vectors
Output	List of words/subwords/characters	Dense numerical vectors
Example	`"I love AI"` → `["I", "love", "AI"]`	`"love"` → `[0.54, 0.76, 0.12, ...]`
Focus	Structure of text	Semantic meaning
Type	Preprocessing step	Model feature

Transformers (in AI)

Transformers are a type of deep learning model architecture that have revolutionized the fields of **Natural Language Processing (NLP), Computer Vision (CV),** and **Audio Processing.** They are particularly known for their ability to handle **long-range dependencies** in sequential data better than previous architectures like **RNNs** and **LSTMs**.

***Long-range dependencies** refer to the ability of a model to capture relationships between elements that are far apart in a sequence.*

Key Features of Transformers:

- **Self-Attention Mechanism:**
  - Allows the model to simultaneously focus on different parts of the input sequence.
  - Each word (or token) can attend to all other words in the sequence, capturing context effectively.
- **Parallel Processing:**
  - Unlike RNNs, which process data sequentially, transformers process entire sequences in parallel, making them faster and more efficient for large datasets.
- **Scalability:**
  - Can handle large datasets and models with billions of parameters, such as GPT-3 and BERT.
- **Pre-training and Fine-tuning:**
  - Transformers are often pre-trained on vast datasets and then fine-tuned on smaller, task-specific datasets.

Transformers Architecture

We are training the model to copy tokens. For example, once trained, “dog run” should translate to “dog run”.

Our vocab (size = 5):

Token	ID
<unused>	0
<start>	1
run	2
cat	3
dog	4

Our training data:

Original Sentence	Translated Sentence
dog run	dog run
cat dog	cat dog

Turned into token ids:

Original	Translated
1 4 2	1 4 2
1 3 4	1 3 4

For this whole example we’ll use a single training batch of size 2.

As shown on right, we feed these **inputs**:

```
[[1, 4, 2],
 [1, 3, 4]]
```

And we feed these **“outputs”**:

```
[[1, 4],
 [1, 3]]
```

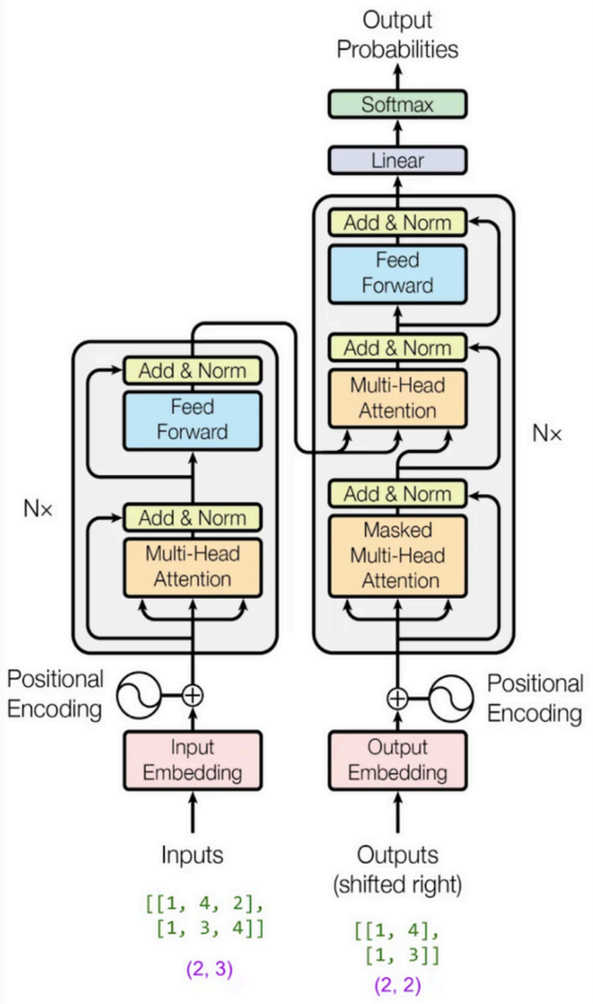
Once trained, the model will translate text. It will get to use the first target token (1 = <start>) to predict the second. It will get to use the first and second target tokens to predict the third. And so on. That’s why during training we use the target token in position n to predict the target token in position n+1 (getting to also look at information from all source tokens and all target tokens up to and including n). So our goal is to predict these **labels**:

```
[[4, 2],
 [3, 4]]
```

And our loss will be based on how well the **output probabilities** (shown at top right) match.

(2, 2, 5)

```
[[[0.2, 0.2, 0.2, 0.3, 0.0], [0.3, 0.2, 0.2, 0.1, 0.2]],
 [[0.0, 0.1, 0.1, 0.6, 0.1], [0.1, 0.0, 0.0, 0.5, 0.3]]]
```



How Transformers Work

- **Input Text Processing:**
  - **Tokenization:** Text is broken into smaller pieces (tokens).
  - Convert tokens to **embeddings**.
  - **Positional encoding** is added to embeddings.
- **Self-Attention Mechanism:**
  - Each word/token interacts with all others, understanding relationships.
  - Uses **key-query-value** attention to determine the importance of words.
- **Feed-Forward Layers:**
  - Transform the attended representations for the **next layer**.
- **Final Output:**
  - The output can be used for **classification, translation**, or **text generation tasks**.

RAG

What is RAG (Retrieval-Augmented Generation)?

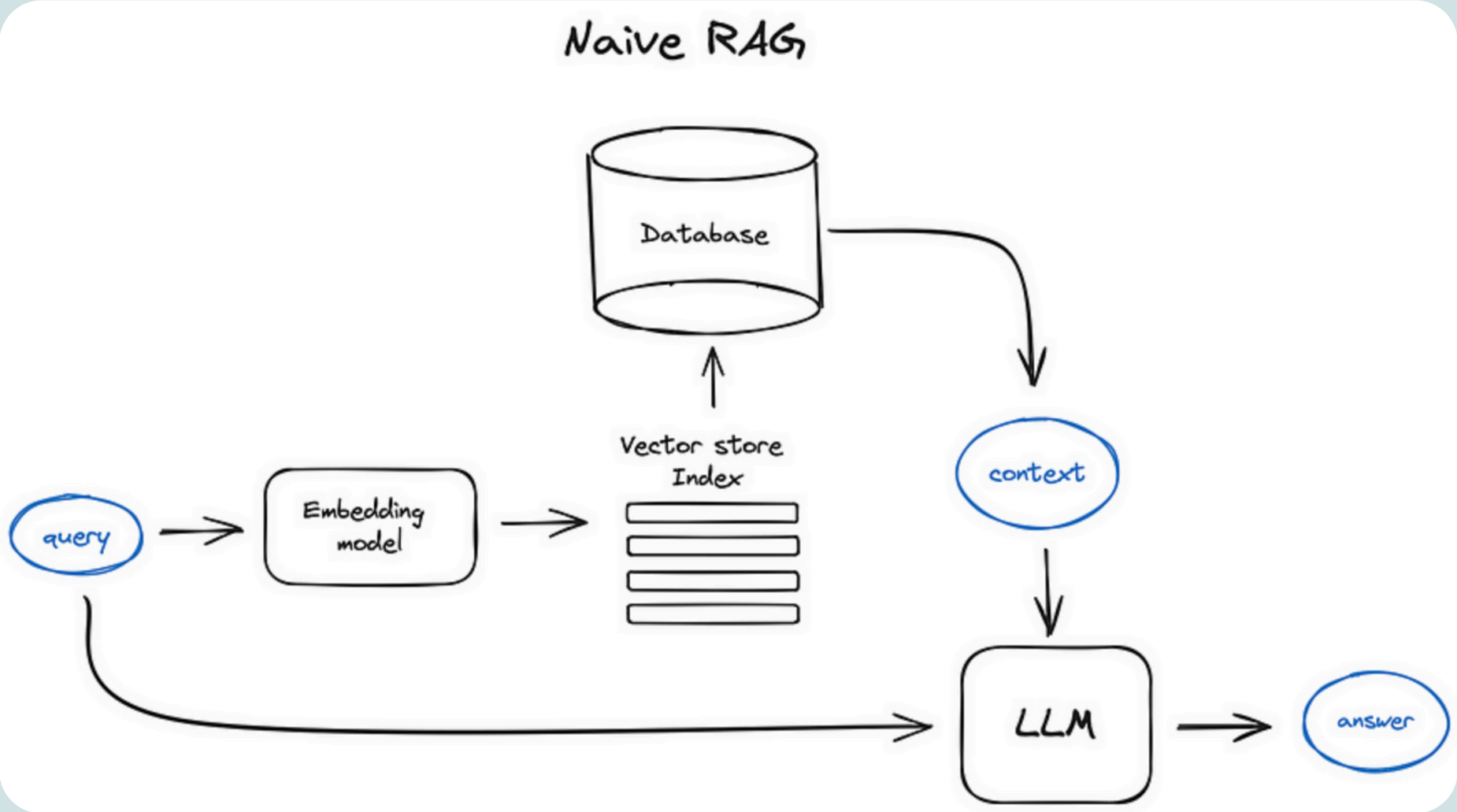
Retrieval-Augmented Generation (RAG) is an advanced AI technique that **combines the power of search (retrieval) and text generation (generation)** to provide more accurate and fact-based answers.

Instead of relying only on what the model was trained on, RAG **retrieves relevant information** from external sources (like databases or documents) before generating a response. This makes the output more reliable, updated, and context-aware.

Key Components in RAG Workflow:

1. **Embedding Model:** Converts text to numerical form (e.g., OpenAI, Sentence Transformers).
2. **Vector Database:** Stores knowledge in numerical form for fast retrieval (e.g., FAISS, Pinecone).
3. **LLM (Language Model):** Generates meaningful responses based on retrieved content (e.g., GPT, BERT).

RAG Process Flow Diagram:



How Does RAG Work?

Step 1: User Query Input

The user types a query, for example:

👉 `"What are the latest AI trends in 2024?"`

Step 2: Convert Query into Embeddings (Numbers)

Since computers don't understand text directly, the query is first converted into a special numeric format called **embeddings**, which captures the meaning of the text.

- A **pre-trained embedding model** (like OpenAI's `text-embedding-ada`) is used.
- Example:
  - Input: `"AI trends"`
  - Output: `[0.1, 0.5, -0.3, 0.8, ...]` (a vector representation)

Step 3: Search in Vector Database (Retrieval Step)

Once the query is converted into embeddings, the system searches for similar embeddings stored in a **vector database** (like FAISS, Pinecone, or Milvus).

- The vector database contains pre-processed and stored information (e.g., AI-related articles).
- The search finds the closest matching documents based on the query embeddings.
- Example:
    - Query embedding → Search → Finds related documents like “Top AI Trends 2024”

Step 4: Retrieve Relevant Documents

The system retrieves the most relevant text snippets/documents from the database based on the similarity score.

- Example retrieved content:
  `"In 2024, AI trends focus on generative AI, ethical AI, and multimodal models."`

Step 5: Pass Data to a Language Model (Generation Step)

Now, the retrieved information (documents/snippets) is sent to a **large language model (LLM)** (like GPT, BERT, etc.), which reads it and generates a response.

- Example process:
  - Input: "According to recent data, the top AI trends are..."
  - Output: `"The key AI trends in 2024 include generative AI and ethical AI."`

Step 6: Response to the User

Finally, the system generates a well-structured, fact-based response and presents it to the user.

Example Output:

"The latest AI trends in 2024 focus on generative AI, ethical AI considerations, and multimodal AI models."

Q&A