**Natural Language Processing (NLP)**

**5️⃣ Stemming**

🔹 **Definition**: Reducing words to their root form (removes suffixes).  
🔹 **Example**:

* **Before** → "running", "jumps", "easily"
* **After** → "run", "jump", "easili"

**6️⃣ Lemmatization**

🔹 **Definition**: Similar to stemming but returns actual dictionary words.  
🔹 **Example**:

* **Before** → "running", "better", "mice"
* **After** → "run", "good", "mouse"

**7️⃣ Part-of-Speech (POS) Tagging**

🔹 **Definition**: Assigning grammatical labels (noun, verb, adjective, etc.) to words.  
🔹 **Example**:

* "The cat sat on the mat" → [('The', DT), ('cat', NN), ('sat', VBD)]

**1️⃣1️⃣ Text Normalization**

🔹 **Definition**: Converting words into their standard form.  
🔹 **Example**:

* **Before** → "I luv NLP"
* **After** → "I love NLP"

## ****🔟 Handling Emojis and Special Characters****

🔹 **Definition**: Removing or converting emojis/emoticons into text.

## ****📌 Summary Table****

| **Technique** | **Purpose** |
| --- | --- |
| **Tokenization** | Splits text into words/sentences |
| **Lowercasing** | Reduces redundancy |
| **Stopword Removal** | Removes common words (which has no meaning) |
| **Punctuation Removal** | Cleans text |
| **Stemming** | Removes word suffixes |
| **Lemmatization** | Converts to base form |
| **POS Tagging** | Assigns grammatical labels |
| **Spelling Correction** | Fixes typos |
| **Handling OOV Words** | Replaces unknown words |
| **Emoji Handling** | Converts emojis to text |
| **Text Normalization** | Fixes slang/short forms |
| **Contraction Handling** | Expands short forms |

**1️⃣ Tokenization**

🔹 **Definition**: Splitting text into individual units like words or sentences.  
🔹 **Why?** Helps in understanding the structure of text and is the first step in NLP.

🔹 **Types of Tokenization**:

* **Word Tokenization** → "I love NLP" → ["I", "love", "NLP"]
* **Sentence Tokenization** → "Hello! How are you?" → ["Hello!", "How are you?"]

**2️⃣ Stopword Removal**

🔹 **Definition**: Removing common words that don’t add much meaning (e.g., "is", "the", "and").  
🔹 **Why?** Reduces noise and improves model efficiency.

🔹 **Example**:

* **Before** → "This is a great NLP tutorial"
* **After** → "great NLP tutorial"

## ****3️⃣ Basic NLP Techniques****

### **1️⃣ Bag of Words (BoW)**

🔹 **Definition**: A simple representation of text that converts sentences into a fixed-length vector based on word occurrences.  
🔹 **How It Works?**

* Create a vocabulary of all unique words.
* Represent each sentence as a vector with word frequencies.

🔹 **Example**:

| **Sentence** | **NLP** | **is** | **fun** | **deep** | **learning** |
| --- | --- | --- | --- | --- | --- |
| "NLP is fun" | 1 | 1 | 1 | 0 | 0 |
| "deep learning is fun" | 0 | 1 | 1 | 1 | 1 |

### **2️⃣ Term Frequency-Inverse Document Frequency (TF-IDF)**

🔹 **Definition**: A statistical measure that evaluates how important a word is in a document relative to a collection of documents.  
🔹 **Formula**:

TF(w)=count of word w in a documenttotal words in the documentTF(w) = \frac{\text{count of word } w \text{ in a document}}{\text{total words in the document}}TF(w)=total words in the documentcount of word w in a document​ IDF(w)=log⁡(Total documents1+Documents containing w)IDF(w) = \log \left(\frac{\text{Total documents}}{1 + \text{Documents containing } w}\right)IDF(w)=log(1+Documents containing wTotal documents​) TF−IDF(w)=TF(w)×IDF(w)TF-IDF(w) = TF(w) \times IDF(w)TF−IDF(w)=TF(w)×IDF(w)

🔹 **Example**:

| **Word** | **TF** | **IDF** | **TF-IDF** |
| --- | --- | --- | --- |
| NLP | 0.2 | 1.3 | 0.26 |
| fun | 0.1 | 1.7 | 0.17 |

**📌 List of Word Embedding Techniques**

1️⃣ **Count-Based Embeddings**

### **🔹 One-Hot Encoding**

* Represents each word as a **binary vector**.
* **Example:** Vocabulary = ["apple", "banana", "cherry"]
  + apple → [1, 0, 0]
  + banana → [0, 1, 0]
  + cherry → [0, 0, 1]
* **Issues**: No semantic meaning, large sparse matrices.

### **🔹 Bag of Words (BoW)**

* Counts the number of times a word appears in a document.
* **Example:**

nginx

CopyEdit

Sentence 1: "I love NLP"

Sentence 2: "NLP is great"

| **Word** | **Sentence 1** | **Sentence 2** |
| --- | --- | --- |
| I | 1 | 0 |
| love | 1 | 0 |
| NLP | 1 | 1 |
| is | 0 | 1 |
| great | 0 | 1 |

* **Issues**: Ignores word order and meaning.

### **🔹 TF-IDF (Term Frequency-Inverse Document Frequency)**

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

2️⃣ **Prediction-Based Embeddings (Shallow Neural Networks)**

### **🔹 Word2Vec (CBOW & Skip-Gram)**

* Uses a **shallow neural network** to learn word meanings.
* **CBOW (Continuous Bag of Words)**: Predicts a word using its surrounding words.
* **Skip-Gram**: Predicts surrounding words given a center word.
* **Issues**: Cannot handle out-of-vocabulary (OOV) words.

**Example:**

* CBOW: *"The cat is \_\_ on the mat."* → Predicts **"sitting"**.
* Skip-Gram: **"sitting"** → Predicts nearby words.

### **🔹 FastText (Word2Vec + Subword Information)**

* **Improvement over Word2Vec**: Breaks words into smaller parts (**subwords**).
* **Handles unseen words better** by learning from similar words.
* **Advantages**: Good for morphologically rich languages.

### **5️⃣ What are the advantages of using FastText over Word2Vec?**

**Answer:**

* **Handles out-of-vocabulary (OOV) words** by using **subword information**.
* **Improves performance on morphologically rich languages** like German or Turkish.
* **Better at handling misspellings** (e.g., "intelligent" and "intellignt" will have similar vectors).

### **🔹 GloVe (Global Vectors for Word Representation)**

* Uses **word co-occurrence matrix** instead of a neural network.
* Captures **global relationships** between words.
* **Issues**: Static embeddings, cannot handle new words.

3️⃣ **Contextualized Embeddings (Deep Learning Models - Transformers)**

### **🔹 ELMo (Embeddings from Language Models)**

* Uses **bidirectional LSTMs**.
* **Improvement over Word2Vec**: Generates **contextual** embeddings.
* **Issues**: Computationally expensive.

### **🔹 BERT (Bidirectional Encoder Representations from Transformers)**

* Uses **self-attention (Transformers)**.
* Generates **context-aware embeddings**.
* **Advantages**: Best for NLP tasks, state-of-the-art results.

### **🔹 GPT (Generative Pre-trained Transformer)**

* **Autoregressive model** (predicts words left to right).
* Used for **text generation tasks**.
* **Advantage**: Powerful for text generation.

## ****📍 Summary Table****

| **Embedding Technique** | **Type** | **Context-Aware?** | **Handles OOV?** | **Common Use** |
| --- | --- | --- | --- | --- |
| **One-Hot Encoding** | Count-Based | ❌ No | ❌ No | Simple NLP models |
| **TF-IDF** | Count-Based | ❌ No | ❌ No | Text classification |
| **Word2Vec** | Prediction-Based | ❌ No | ❌ No | Word similarity |
| **FastText** | Prediction-Based | ❌ No | ✅ Yes | Handling OOV words |
| **GloVe** | Count-Based | ❌ No | ❌ No | Semantic similarity |
| **BERT** | Transformer-Based | ✅ Yes | ✅ Yes | All NLP tasks |
| **GPT** | Transformer-Based | ✅ Yes | ✅ Yes | Text generation |

### **7️⃣ What is the difference between static and contextual word embeddings?**

| **Feature** | **Static (Word2Vec, GloVe)** | **Contextual (BERT, GPT)** |
| --- | --- | --- |
| Word Representation | Fixed for each word | Changes based on context |
| Handles Homonyms? | ❌ No | ✅ Yes |
| Example | "bank" (same vector for river & finance) | "bank" (different vectors for different contexts) |

### **9️⃣ What is subword embedding in FastText?**

**Answer:**

* Instead of treating words as a whole, FastText **breaks them into subword units** (n-grams).
* Helps with **handling rare words** and **misspellings**.
* **Example:**

arduino

CopyEdit

"playing" → ["pla", "lay", "ayi", "yin", "ing"]

Each subword has an embedding, and the word’s final representation is the sum of all subword embeddings.

### **1️⃣1️⃣ What is the difference between ELMo, BERT, and GPT?**

| **Feature** | **ELMo** | **BERT** | **GPT** |
| --- | --- | --- | --- |
| Architecture | BiLSTM | Transformer (Encoder) | Transformer (Decoder) |
| Training | Predicts next character | Masked Language Model (MLM) | Autoregressive (predicts next word) |
| Bidirectional? | ✅ Yes | ✅ Yes | ❌ No |

### **1️⃣3️⃣ What are positional encodings in Transformer models?**

**Answer:**  
Transformers **do not process words sequentially** (like RNNs), so they need **positional encodings** to understand the order of words.

### **1️⃣4️⃣ What is the difference between word-level and character-level embeddings?**

| **Feature** | **Word-Level** | **Character-Level** |
| --- | --- | --- |
| Example | "playing" | ["p", "l", "a", "y", "i", "n", "g"] |
| Good for? | General NLP | Handling misspellings & rare words |
| Model | Word2Vec, BERT | CNN, FastText |

## ****📌 LSTM vs. GRU (Key Differences & Comparison)****

Both **Long Short-Term Memory (LSTM)** and **Gated Recurrent Unit (GRU)** are types of **Recurrent Neural Networks (RNNs)** designed to handle **sequential data**. They solve the **vanishing gradient problem** of traditional RNNs using **gates** to control information flow.

### **🔄 1️⃣ Architecture Differences**

| **Feature** | **LSTM** | **GRU** |
| --- | --- | --- |
| **Number of Gates** | 3 (Forget, Input, Output) | 2 (Reset, Update) |
| **Hidden State Components** | Maintains a separate **cell state** and **hidden state** | Only one **hidden state** |
| **Computational Efficiency** | **Slower** (More parameters & memory usage) | **Faster** (Fewer parameters, lower memory) |
| **Performance on Small Datasets** | **Better** for longer sequences & complex dependencies | **Better** for short-term dependencies |
| **Training Time** | More complex, takes longer to train | Faster training & inference |

### **🧠 2️⃣ Why LSTMs Are Still Used in Time Series After Transformers?**

Even though **Transformers** dominate NLP tasks, **LSTMs still perform well in Time Series Analysis** due to:

🔹 **Smaller Dataset Suitability**: LSTMs work well with **limited labeled time series data**, whereas **Transformers require large datasets** to generalize.

🔹 **Less Computational Power Needed**: LSTMs are lightweight compared to Transformers, which require **quadratic memory complexity** due to **self-attention**.

🔹 **Continuous Data Processing**: Time series data often requires real-time processing. **Transformers process entire sequences at once**, making them less efficient for streaming data.

🔹 **Fewer Parameters & Faster Training**: LSTMs are computationally **cheaper to train** compared to **Transformer-based models like GPT, T5, BERT**, which need GPUs/TPUs.

🔹 **Better for Short-Length Dependencies**: While **Transformers excel in long-range dependencies**, LSTMs can **outperform them** for shorter time dependencies.

### **✔️ When should you use LSTM over GRU or vice versa in a real-world project?**

✅ **Use LSTM when:**

* You need to **capture long-term dependencies** in data.
* Your dataset is **complex with long sequences**, such as speech recognition, language modeling, or stock price prediction.
* You have **sufficient computational resources** and can afford a **slower training time**.

✅ **Use GRU when:**

* You need a **faster and more efficient** model with fewer parameters.
* Your dataset contains **short-term dependencies**, such as sensor data analysis or real-time event detection.
* You have **limited computational power**, as GRUs are computationally lighter than LSTMs.
* You are **prototyping models quickly** and need a simpler architecture.

**🔑 Rule of Thumb:** If **accuracy is more important** → go for **LSTM**. If **speed and efficiency matter** → use **GRU**.

### **✔️ What is the difference between RNN, LSTM, and GRU?**

| **Feature** | **RNN** | **LSTM** | **GRU** |
| --- | --- | --- | --- |
| **Vanishing Gradient** | Yes | Solves it with gates | Solves it with fewer gates |
| **Gates** | None | Forget, Input, Output | Reset, Update |
| **Computational Cost** | Low | High | Medium |
| **Performance on Long Sequences** | Poor | Best | Good |
| **Memory Usage** | Low | High | Medium |
| **Training Speed** | Fast | Slow | Faster than LSTM |
| **Best Use Case** | Short-term dependencies | Long-term dependencies | Short sequences, faster training |

### **✔️ Why do we use a tanh activation function inside LSTMs?**

🔹 **LSTMs use tanh because:**

1. **Keeps values in range [-1, 1]** → Helps stabilize the gradient.
2. **Balances activations** → Prevents exploding values in long sequences.

In LSTM diagram, the pink operations are point wise operations and yellow sigmoid and tanh are neural network layer (ANN) having respective activation functions. Dimension of hidden state, cell state always same and these are assigned by number of units in ANN.

**🟡 Tokenization Techniques in LLMs**

Tokenization splits text into smaller units (**tokens**) before processing.

## ****1️⃣ Byte-Pair Encoding (BPE)****

### **How It Works?**

* Starts with **single characters** as tokens.
* **Merges the most frequent adjacent pairs** iteratively until a fixed vocabulary size is reached.
* Efficient for **handling rare words** by breaking them into subwords.

### **Example 1:** Tokenizing "unhappiness"

1. Start with characters: ["u", "n", "h", "a", "p", "p", "i", "n", "e", "s", "s"]
2. Merge common pairs:
   * "h" + "a" → "ha"
   * "p" + "p" → "pp"
   * "i" + "n" → "in"
   * "s" + "s" → "ss"
3. Final tokens: ["un", "ha", "pp", "in", "e", "ss"]

### **Example 2:** Tokenizing "internationalization"

1. Start with characters: ["i", "n", "t", "e", "r", "n", "a", "t", "i", "o", "n", "a", "l", "i", "z", "a", "t", "i", "o", "n"]
2. Merge common pairs:
   * "i" + "n" → "in"
   * "t" + "e" → "te"
   * "r" + "n" → "rn"
   * "a" + "t" → "at"
3. Final tokens: ["in", "ter", "na", "tion", "ali", "zation"]

### **Where It's Used?**

✅ **GPT-3, GPT-4, OpenAI Tokenizer**

## ****2️⃣ WordPiece Tokenization****

### **How It Works?**

* Similar to BPE but **merges based on likelihood estimation** instead of frequency.
* Used in **BERT-based models** for efficient **subword tokenization**.

### **Example 1:** Tokenizing "unhappiness"

1. Break into known words: ["un", "happiness"]
2. If "happiness" is missing from the vocabulary:
   * Split further: ["un", "hap", "##pi", "##ness"]
3. Final tokens: ["un", "hap", "##pi", "##ness"]

### **Example 2:** Tokenizing "playing"

1. If "playing" is not in vocabulary:
   * Split into ["play", "##ing"]
2. If "play" is also missing:
   * Further split into ["pla", "##y", "##ing"]
3. Final tokens: ["pla", "##y", "##ing"]

### **Where It's Used?**

✅ **BERT, RoBERTa, DistilBERT**

## ****3️⃣ SentencePiece Tokenization****

### **How It Works?**

* Treats text as a **single sequence**, meaning it doesn’t rely on whitespace to split words.
* Used for **non-space-separated languages (Japanese, Chinese, etc.)**.

### **Example 1:** Tokenizing "This is an example"

1. No word boundaries assumed.
2. Possible output: ["▁This", "▁is", "▁an", "▁ex", "ample"]
3. Notice **the underscore** (▁) indicates a new word.

### **Example 2:** Tokenizing "Tokyo Olympics"

1. Possible output: ["▁Tok", "yo", "▁Olym", "pics"]
2. "Tokyo" is split into ["Tok", "yo"] because "yo" may appear frequently in other words.

### **Where It's Used?**

✅ **T5, LLaMA, mT5, ALBERT**

## ****1️⃣ Fine-Tuning (Supervised Learning Approach)****

### 🔹 **Definition:**

Fine-tuning means **training an LLM on a specific dataset** to improve its performance on a particular task. The model adjusts its weights based on labeled examples.

### 🔹 **Key Features:**

* Requires **task-specific** labeled data.
* The model **learns from examples** rather than general instructions.

## ****2️⃣ Instruction Tuning (Improving Model’s Ability to Follow Instructions)****

### 🔹 **Definition:**

Instruction tuning **does not require labeled task-specific data** but instead **teaches the model how to follow general human instructions more effectively**.

### 🔹 **Key Features:**

* Instead of **just feeding data**, we teach the model **how to follow commands better**.
* Uses **varied instructions and responses** to generalize performance across different tasks.
* **Less expensive** than fine-tuning (does not require large-scale labeled data).
* Helps in **zero-shot and few-shot learning**—where the model can perform tasks it wasn’t explicitly trained for.

### **📝 Example: Instruction Tuning GPT for Better Generalization**

Let’s say you have GPT-3 and want it to **better understand human instructions across many tasks**.

🔹 **Steps:**

1. Train the model on **instruction-based examples** like:
   * **“Summarize this article in one sentence.”**
   * **“Translate this paragraph into French.”**
   * **“Explain this concept in simple words.”**
2. The model **learns how to follow human instructions better**, even on tasks it has never seen before.

✅ **Result:**  
A **GPT-3-Instruct model** that follows human commands more naturally.

### **🔹 What is PEFT?**

PEFT (Parameter-Efficient Fine-Tuning) is a method to **adapt a large pre-trained model** to a specific task **without modifying all its parameters**. Instead of updating the entire model, PEFT **only fine-tunes a small subset of parameters**, making it much **faster and memory-efficient** than full fine-tuning.

## ****2️⃣ PEFT Techniques with Examples****

### **📌 (A) LoRA (Low-Rank Adaptation)**

🔹 Instead of updating the entire weight matrix, LoRA **adds small trainable matrices** that adjust the model's behavior.  
🔹 This significantly reduces the number of parameters that need updating.

📝 **Example:**

* Suppose we have a **general GPT-3 model** and want to fine-tune it for **legal text processing**.
* Instead of retraining the whole model, LoRA **adds small additional layers** that adapt the model to legal terminology.
* This allows the model to work efficiently on legal tasks **without losing general knowledge**.

### **📌 (B) Adapters**

🔹 Adapters are **small neural networks inserted inside the model layers** that are trained while keeping the rest of the model frozen.  
🔹 They **specialize the model for different tasks without changing the base model**.

📝 **Example:**

* A company wants to use **BERT** for **sentiment analysis, medical text classification, and chatbot responses**.
* Instead of fine-tuning BERT three times, they **train separate adapters** for each task and switch between them when needed.
* The base BERT model remains the same, but the adapters specialize it for different tasks.

### **📌 (C) Prompt Tuning vs. Prefix Tuning**

🔹 These techniques modify **only the input prompts**, rather than changing the model’s internal parameters.

📝 **Example (Prompt Tuning):**

* A general **GPT model** is trained to generate creative content.
* Instead of fine-tuning, **special prompt embeddings** are trained to make GPT write only in a **legal** or **medical** style.

📝 **Example (Prefix Tuning):**

* Instead of changing model weights, **extra context tokens** are added to the input.
* For a **summarization model**, prefix tuning adds a trained prompt like:  
  “Summarize this legal contract in simple terms:”
* This directs the model to perform well on a task **without modifying its core knowledge**.

### **🔴 Challenges in LLMs**

7️⃣ **Computational Challenges**

* Why do **LLMs require massive GPU clusters**?
* **Memory bottlenecks and model parallelism** (ZeRO, FSDP, etc.).
* What is **model quantization (int8, int4, GPTQ)?**

8️⃣ **Hallucinations in LLMs**

* Why do LLMs **generate false information**?
* How to **mitigate hallucinations**? (RAG, Fact-checking, Fine-tuning).

9️⃣ **Bias & Ethics in LLMs**

* How do LLMs learn biases?
* Strategies for **reducing harmful outputs**.

## ****Optimizing LLM Inference Techniques****

When deploying Large Language Models (LLMs) like GPT, LLaMA, or Falcon, reducing inference **latency** and **cost** is critical. Here are some key techniques:

### **1️⃣ Quantization**

* Reduces model size by lowering precision (e.g., **FP32 → INT8** or **INT4**).
* **Speeds up inference** with minimal loss in accuracy.
* Example: GPT-4 running on edge devices using **4-bit quantization (QLoRA)**.

### **2️⃣ Pruning**

* Removes **less important weights** from the neural network.
* Reduces **computational load** while maintaining accuracy.
* Example: Removing redundant connections in **BERT for mobile applications**.

### **3️⃣ Knowledge Distillation**

* A smaller **"student model"** learns from a larger **"teacher model"**.
* Used for **deploying LLMs in low-power environments**.
* Example: **DistilBERT (66% of BERT’s size) runs 60% faster** with minimal accuracy loss.

### **4️⃣ Caching & KV Cache Optimization**

* Stores previously computed **attention values** to speed up responses.
* Essential for **chatbots with long conversations**.
* Example: **GPT models cache key-value (KV) pairs to avoid recomputing past tokens**.

### **5️⃣ Speculative Decoding**

* Generates **multiple tokens in parallel** rather than sequentially.
* **Reduces latency** by 2x–4x in transformer models.
* Example: Used in **OpenAI's GPT acceleration pipeline**.

### **6️⃣ Flash Attention & Efficient Transformers**

* Uses **FlashAttention** and **xFormers** to reduce memory bandwidth issues.
* Optimizes **self-attention computation** for large sequences.
* Example: **LLaMA 2 and GPT-4 use FlashAttention to scale better**.

### **7️⃣ Mixture of Experts (MoE)**

* Activates only **a subset of model parameters per query**, saving computation.
* Used in **GPT-4 and Switch Transformer for scaling**.
* Example: Google’s **GLaM (Gated MoE) runs on fewer active parameters per request**.

### **🔹 What is RAG?**

RAG combines a **retrieval system** (fetching external knowledge) with a **generation model** (LLM) to enhance responses.

## ****1️⃣ Vanilla RAG (Standard RAG)****

This is the **basic RAG pipeline** where:

* A query is embedded and used to retrieve **top-k relevant documents** from a **vector database**.
* The LLM takes the retrieved documents **as context** and generates a response.

📌 **Example:** A chatbot that retrieves **FAQs from a company knowledge base** to answer customer queries.

🔹 **Pros:** Simple and efficient  
🔹 **Cons:** May retrieve irrelevant or redundant information

## ****2️⃣ Multi-Hop RAG****

Instead of retrieving information **in one step**, Multi-Hop RAG:

* Uses an **initial retrieval** to fetch **partial answers**.
* Uses those answers to make **another query** to get more relevant details.

📌 **Example:**  
A **medical AI assistant** that first retrieves **symptoms for a disease**, then looks for **related treatments**.

🔹 **Pros:** Better at handling **complex, multi-step questions**  
🔹 **Cons:** More computationally expensive

## ****3️⃣ Hierarchical RAG (Chunk-Based RAG)****

Used for **long documents** like books, research papers, and contracts.

* The text is **broken into chunks** (e.g., paragraphs, sections).
* RAG first **retrieves a relevant chunk**.
* The **LLM summarizes or answers questions** from that chunk.

📌 **Example:**  
A **legal AI assistant** answering questions about a contract by retrieving specific clauses.

🔹 **Pros:** Efficient for long documents  
🔹 **Cons:** May miss global context

## ****4️⃣ RAG with Re-Ranking (Ranked RAG)****

Instead of **using the first k retrieved documents**, this approach:

* **Re-ranks** retrieved documents based on relevance using **BM25, Cohere Rerank, or BGE models**.
* Passes only **the best-ranked documents** to the LLM.

📌 **Example:**  
A **research assistant** retrieving papers but **prioritizing the most cited ones**.

🔹 **Pros:** **Higher accuracy, avoids irrelevant results**  
🔹 **Cons:** Adds **extra processing time**

## ****5️⃣ Adaptive RAG (Dynamic Retrieval Augmentation)****

* Adjusts the **number of retrieved documents** dynamically based on **query complexity**.
* If the query is **simple**, it retrieves fewer documents.
* If the query is **complex**, it retrieves more documents.

📌 **Example:**  
An **AI tutor** that provides **short answers for simple questions** and **detailed responses for harder ones**.

🔹 **Pros:** Saves **compute resources**  
🔹 **Cons:** Needs **fine-tuning to adjust retrieval thresholds**

## ****6️⃣ Agentic RAG (RAG + Agents)****

* Uses **LLM agents** to **analyze retrieved documents** before answering.
* Agents can **filter, summarize, or combine** different documents dynamically.

📌 **Example:**  
A **market research AI** that **retrieves financial reports, filters trends, and summarizes them**.

🔹 **Pros:** Improves **response reliability**  
🔹 **Cons:** **Slower inference time**

## ****7️⃣ Hybrid RAG (Combining Multiple Retrieval Methods)****

Instead of using **only vector search**, Hybrid RAG:

* Combines **vector search (embeddings)** and **keyword-based search (BM25, Elasticsearch)**.
* This improves retrieval when **exact keyword matching is important**.

📌 **Example:**  
A **search engine for medical literature** retrieving papers using **both keyword and semantic search**.

🔹 **Pros:** Best of **both retrieval worlds**  
🔹 **Cons:** Requires **more storage & compute**

## ****2️⃣ Popular Keyword-Based Search Techniques****

### **📌 TF-IDF (Term Frequency - Inverse Document Frequency)**

* **Measures word importance** in a document relative to a corpus.
* **Formula:** TF−IDF=TF×IDFTF-IDF = TF \times IDFTF−IDF=TF×IDF
  + **TF (Term Frequency):** Counts how often a word appears in a document.
  + **IDF (Inverse Document Frequency):** Gives **less weight** to common words (e.g., "the", "is").

🔹 **Example:**  
Query: **"Machine Learning Trends"**  
Document 1: **"Machine learning is evolving"** → High TF-IDF score ✅  
Document 2: **"AI and deep learning"** → Low TF-IDF score ❌

### **📌 BM25 (Best Matching 25) - Improved TF-IDF**

* **Enhances TF-IDF** by considering **document length** and **word saturation**.
* Gives **higher scores to shorter, more relevant documents**.

🔹 **Example:**  
Query: **"NLP transformers"**

* **Short document** with **NLP** and **transformers** in **title** → Higher BM25 score ✅
* **Long document** with many unrelated words → Lower BM25 score ❌

🔹 **Used In:**  
✔ **Elasticsearch**  
✔ **Search engines (Google, Bing, etc.)**

✨ **What is greedy search?**

* At each decoding step, the model selects the word with the **highest probability** as the next word.
* It’s fast but shortsighted — it makes the best choice **at each step** without considering the **long-term coherence** of the sentence.

👉 Example:  
Input: *Je suis étudiant.*  
Greedy translation: *I am student.* (Missing “a”)  
Better translation: *I am a student.*

⚠️ **Why greedy search isn’t ideal for machine translation:**

1. **Lack of global context:**
   * Greedy search only looks at the most probable word at each step, without considering how the sentence as a whole will flow.
   * A word choice that seems best at the current step may lead to awkward or incorrect sentences later.
2. **Misses better sequences:**
   * The model may miss a more accurate or fluent translation that requires picking a less probable word initially to produce a better final sequence.
   * Translation often requires long-range dependencies — what works best in the beginning may not be optimal for the sentence’s ending.
3. **Early mistakes compound:**
   * Once the model picks a wrong word, there’s no going back. Each subsequent word depends on the previous ones, so one mistake can ruin the whole sentence.

✅ **What’s better than greedy search?**

1. **Beam search:**
   * Keeps multiple possible translations (beams) at each step, not just the most probable one.
   * Helps find a more globally optimal translation.
   * Trade-off: More computationally expensive than greedy search.
2. **Top-k sampling:**
   * Chooses from the top k most likely next words (instead of just the top 1), introducing diversity.
   * Often used in more creative tasks (like chatbot dialogue).
3. **Nucleus sampling (top-p):**
   * Dynamically selects from the smallest set of words that account for most of the probability mass (top p%).
   * Helps avoid repetitive or generic translations.

**Transformer Detail:**

[**https://chatgpt.com/c/67b2eb0d-81a8-800e-a784-a421c0b1ee41**](https://chatgpt.com/c/67b2eb0d-81a8-800e-a784-a421c0b1ee41)