Contents

[GENERATIVE AI 3](#_Toc205118523)

[NLP(NATURAL LANGUAGE PROCESSING) 3](#_Toc205118524)

[GENERATIVE AI VS DISCRIMINATIVE(PREDICTIVE) AI 5](#_Toc205118525)

[APPLICATION OF GEN-AI 7](#_Toc205118526)

[GEN-AI LIMITATIONS & CHALLENGES 7](#_Toc205118527)

[LARGE LANGUAGE MODELS(LLMs) 9](#_Toc205118528)

[LLM ARCHITECTURE 10](#_Toc205118529)

[CORE COMPONENTS OF LLMs 10](#_Toc205118530)

[EMBEDDINGS 13](#_Toc205118531)

[VECTORS 14](#_Toc205118532)

[WHY ARE VECTORS USEFUL? 14](#_Toc205118533)

[VECTOR DATABASE 14](#_Toc205118534)

[HOW LLMS WORK 15](#_Toc205118535)

[PROMPTS & TOKENS 17](#_Toc205118536)

[TOTAL TOKENS 18](#_Toc205118537)

[MODEL TYPES (LLM TYPES) 19](#_Toc205118538)

[CLASSIFICATION BASED ON – HOW THEY ARE TRAINED 19](#_Toc205118539)

[CLASSIFICATION BASED ON – HOW THEY ARE USED 21](#_Toc205118540)

[GENERAL PURPOSE AND DOMAIN-SPECIFIC LLMS 21](#_Toc205118541)

[FINE TUNING 23](#_Toc205118542)

[FINE TUNING TECHNIQUES 24](#_Toc205118543)

[DIFFERENT WAYS TO FINE TUNNING A MODEL 24](#_Toc205118544)

[RAG (RETRIEVER AUGUMENTED GENERATION) 26](#_Toc205118545)

[RAG WORKFLOW 26](#_Toc205118546)

[WHY RAG? 27](#_Toc205118547)

[BENEFITS 27](#_Toc205118548)

[RAG ARCHITECTURE 28](#_Toc205118549)

[EXAMPLES 28](#_Toc205118550)

[RAG FRAMEWORK 29](#_Toc205118551)

[TOOLS TO BUILD RAG SYSTEMS 30](#_Toc205118552)

[LANGCHAIN 30](#_Toc205118553)

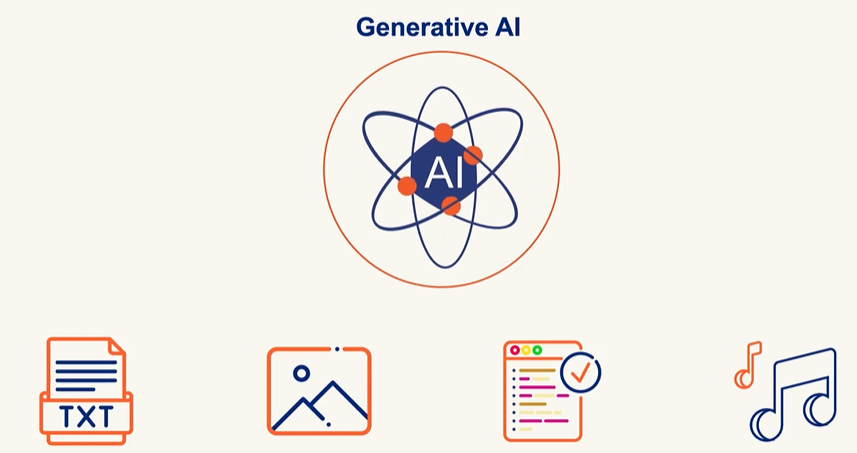
[AGENTIC AI 31](#_Toc205118554)

[EXAMPLES 31](#_Toc205118555)

[MULTI-AGENT AGENTIC AI? 32](#_Toc205118556)

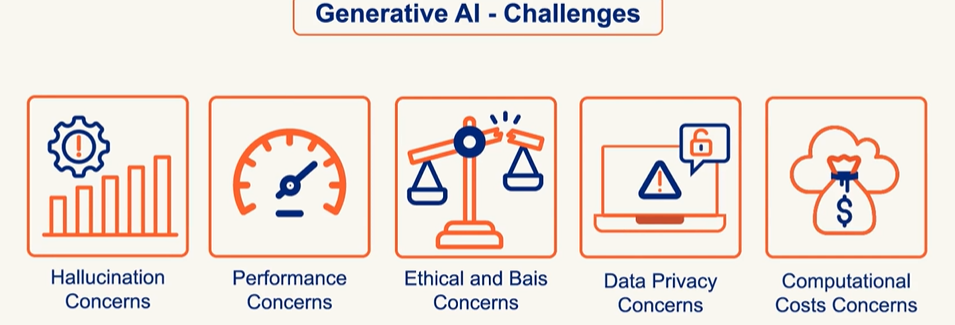
# GENERATIVE AI

What Is Generative Ai



*Generative AI creates new content like Text, Image, Audio and Code based in learned Pattern unlike Traditional AI which classifies and predicts*

Generative Ai Challenges



# NLP(NATURAL LANGUAGE PROCESSING)

A blue and white logo

AI-generated content may be incorrect.

* Enabling and teaching computers to understand, interpret, generate, and respond to human language
* In **Generative AI (GenAI)**, **Natural Language Processing (NLP)** plays a **central role**. GenAI systems like ChatGPT, Bard, Claude, and others are built on advanced NLP techniques to understand and generate human-like language

How NLP Works

Data can be classified as

1. Structured (Sheets/Tables): This type of data is easy to standardize and categorize
2. Unstructured (Audio, Video, Social Media Posts, Email): Difficult standardize and categorize

A diagram of data classification

AI-generated content may be incorrect.

Key Areas of NLP

1. **Text Processing**: Tokenization, stemming, lemmatization, stop-word removal.
2. **Syntax & Parsing**: Understanding grammatical structure.
3. **Semantics**: Understanding meaning and context.
4. **Sentiment Analysis**: Detecting emotions or opinions in text.
5. **Machine Translation**: Translating text between languages.
6. **Speech Recognition**: Converting spoken language into text.
7. **Text Generation**: Creating human-like text (like what I do!).
8. **Named Entity Recognition** (NER): Identifying names, places, dates, etc.
9. **Question Answering & Chatbots**: Building systems that can answer questions or hold conversations.

Challenges in NLP

Syntactic Ambiguity

* **Definition**: When a sentence can be parsed in more than one way due to its structure.
* **Example**: *“I saw the man with the telescope.” 🡪*Did you use a telescope to see the man, or did the man have a telescope?

Lexical Ambiguity

* **Definition**: When a word has multiple meanings.
* **Example**: *“Bank” 🡪* Could mean a financial institution or the side of a river.

Misspellings or Typos

* **Definition**: Errors in spelling that can confuse NLP systems.
* **Example**: *“Recieve”* instead of *“Receive” 🡪*May lead to incorrect parsing or missed keyword matches.

Coreferential Ambiguity

* **Definition**: When it's unclear what a pronoun or noun phrase refers to.
* **Example**: *“John told Tom that he won.”🡪*Who won—John or Tom?

Uncertainty and Idiomatic Ambiguity

* **Definition**: Phrases that are not meant to be taken literally or are vague.
* **Example**: *“Kick the bucket” 🡪* Literally means to kick a bucket, but idiomatically means to die.

Mixing of Languages

* **Definition**: Use of multiple languages in the same sentence or conversation.
* **Example**: *“Mujhe pizza pasand hai, especially with extra cheese.” 🡪* Hindi-English mix (code-switching), common in multilingual societies.

Issues with Social Media Slang Abbreviations

* **Definition**: Informal or abbreviated language used online that may not be in standard dictionaries.
* **Example**: *“LOL”, “BRB”, “SMH” 🡪*These can confuse models not trained on such data.

Inadequate Training Data

* **Definition**: When the model hasn’t seen enough examples of a certain language, dialect, or context.
* **Example**: A chatbot trained only on formal English may struggle with regional slang or dialects.

# GENERATIVE AI VS DISCRIMINATIVE(PREDICTIVE) AI

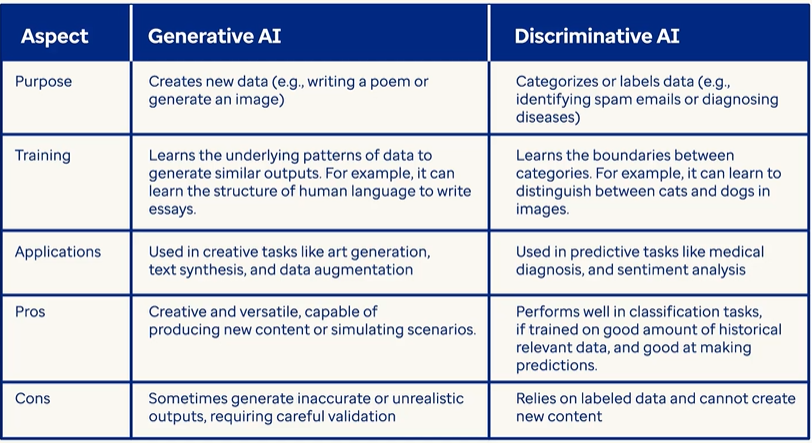
* **Generative AI models** learn the **joint probability** of inputs and outputs (P(x, y)). This means they understand how data is structured and can **generate new data** that looks similar to what they’ve seen.



* **Discriminative AI models** learn the **conditional probability** (P(y|x))—they focus on **distinguishing** between different categories or classes. Its goal is to **classify** or **predict** based on input data.
* Example:
  + **Spam filters** classify emails as spam or not spam.
  + **Face recognition systems** identify people from images.
  + **Sentiment analysis** detects if a review is positive or negative.

A logo with blue and orange squares

AI-generated content may be incorrect.



* **Gen AI is a subset of deep learning, and it uses artificial neural networks to process labeled and unlabeled data using supervised, unsupervised, and semi-supervised methods to generate new content like text, images, videos, or audio.**
* **Unlike conventional AI, Generative AI doesn't just classify or predict data; it generates brand-new content based on its training data.**
* *In simple terms, traditional “predictive” machine learning models attempt to learn the relationship between the data and what we want to predict while A generative AI model attempts to learn patterns so that it can generate new content.*

**CONVENTIONAL AI**

A blue and white logo

AI-generated content may be incorrect.

* Operates by learning from training data and making predictions, classifications, or performing language processing/computer vision.
* Example: Trained on apple images, it tells whether a supplied image is of an apple.

**GENERATIVE AI**

A blue and white logo

AI-generated content may be incorrect.

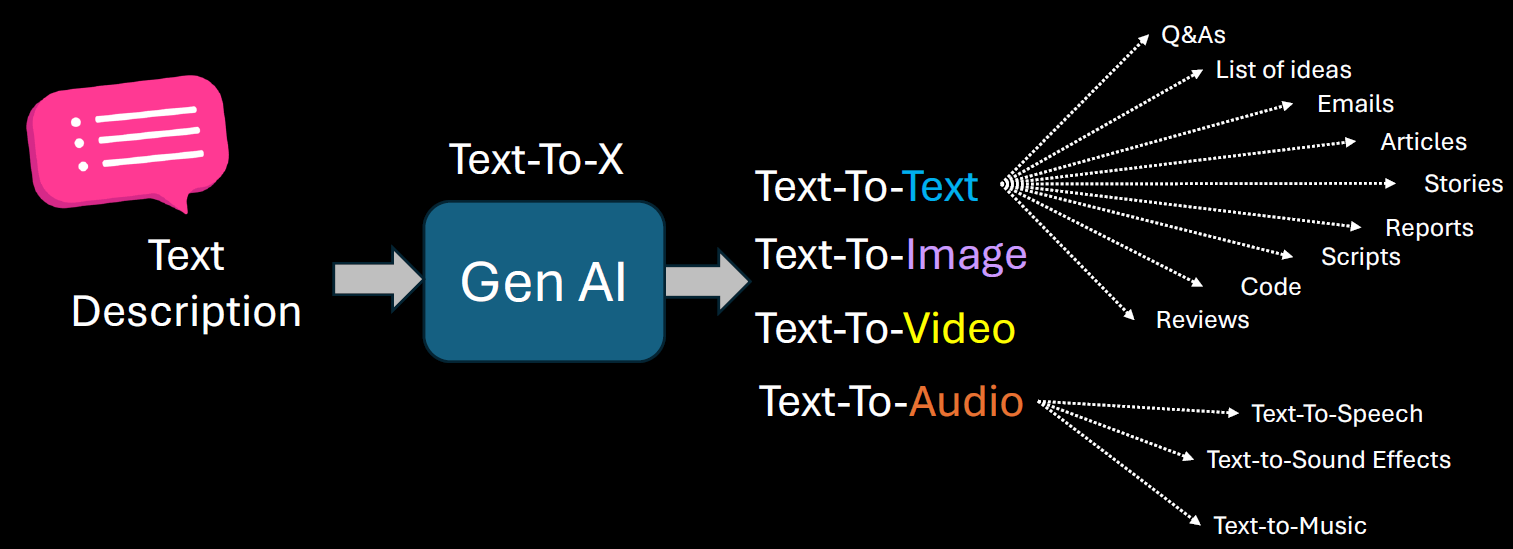
TYPES OF CONTENT GENERATED BY GEN-AI MODELS

A blue circle with text

AI-generated content may be incorrect.

* Learning from training data and creating *new* content.
* Example: Trained on apple images, it generates a new apple image not extracted from the training data.

# APPLICATION OF GEN-AI



GEN-AI USE CASES

1. BRAINSTORM ASSISTANT
2. SUMMARIZATION
3. CODE GENERATION
4. TEXT ENHANCEMENT
5. IMAGE GENERATION

## 

# GEN-AI LIMITATIONS & CHALLENGES

**PROMPT SENSITIVITY**

* Generative AI models are highly sensitive to how a prompt is phrased. Small changes in wording can lead to significantly different outputs.
* This can make it difficult to get consistent or desired results without careful prompt engineering.
* **Example:** Asking "Explain climate change simply" vs. "What causes climate change?" may yield different levels of detail or focus.

**KNOWLEDGE CUTOFF**

* Most generative AI models are trained on data available up to a certain point in time. They do not have real-time awareness or access to events or developments after their training cutoff.
* **Implication:** They cannot provide accurate information about recent events, new technologies, or updated regulations unless connected to live data sources.

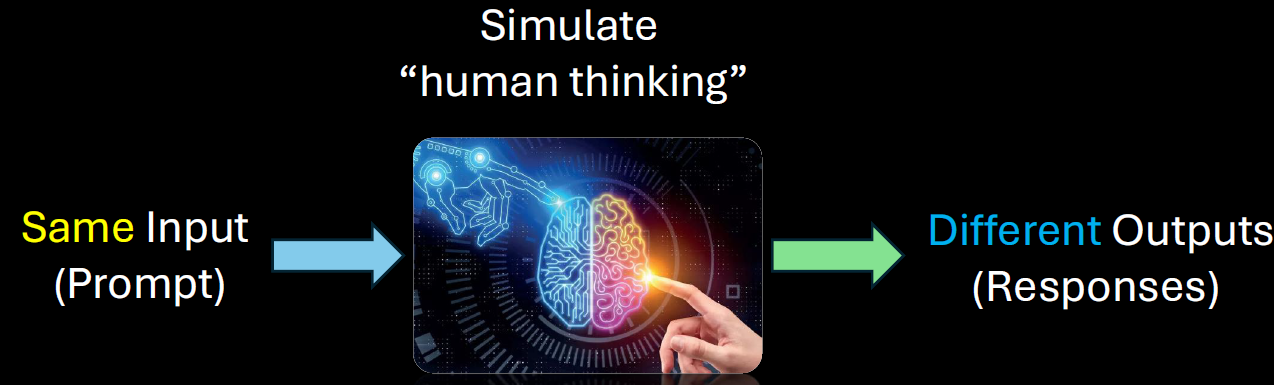
A diagram of a software program

AI-generated content may be incorrect.

The solution to this to

1. Retrain the model after regular interval of time
2. Connect the model with online tools

**IT IS NOT DETERMINISTIC**

****

Generative AI is **probabilistic**, not deterministic. This means the same prompt can produce different outputs each time it's run.

**Implication:** This variability can be useful for creativity but problematic for tasks requiring consistency and repeatability.

**STRUCTURED DATA**

* Generative AI struggles with tasks that require precise manipulation of structured data (like databases, spreadsheets, or complex logic).
* **Example:** It may misinterpret or incorrectly summarize tabular data or fail to follow strict formatting rules.

**HALLUCINATIONS**

* AI models can "hallucinate" — confidently generating false or misleading information that sounds plausible.
* **Example:** Citing non-existent research papers or inventing facts in a historical summary.

**LACK OF COMMON SENSE**

* Despite being trained on vast data, generative AI lacks **true understanding** or **common-sense reasoning**. It may fail at tasks that require intuitive knowledge or real-world logic.
* **Example:** It might suggest putting metal in a microwave or confusing cause and effect in a scenario.

**BIAS AND FAIRNESS**

* AI models can reflect and even amplify biases present in their training data. This can lead to unfair, offensive, or discriminatory outputs.
* **Example:** Gender or racial bias in job recommendations or stereotypical characterizations in generated content.

**DATA PRIVACY, SECURITY, AND MISUSE**

* Generative AI can inadvertently expose sensitive information if trained on private data. It can also be misused for harmful purposes like generating fake news, phishing emails, or deepfakes.
* **Concerns:**
  + **Privacy:** Leaking personal or proprietary data.
  + **Security:** Being used to craft convincing scams.
  + **Misuse:** Generating harmful or misleading content.

# LARGE LANGUAGE MODELS(LLMs)

* A **Large Language Model** is a type of deep learning model, typically based on the **Transformer architecture**, trained on vast corpora of text data.
* It uses **billions (or even trillions) of parameters** to learn statistical patterns in language, enabling it to perform a wide range of natural language processing (NLP) tasks such as text generation, summarization, translation, question answering, and more.
* LLMs are pre-trained on general data and can be fine-tuned for specific domains or tasks.
* Example: **ChatGPT**, **GPT-4**, **Claude**, **Gemini**, and **LLaMA**

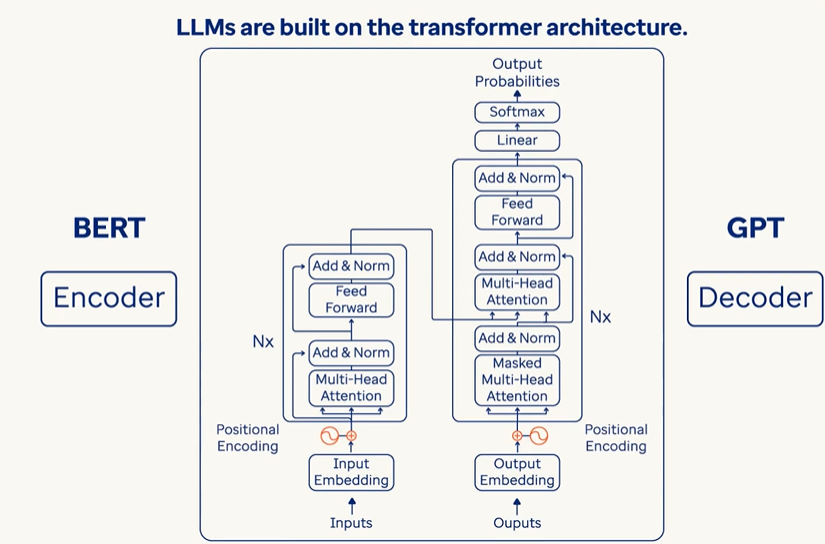


|  |
| --- |
| What are Parameters in LLM?  Simple Analogy: "Baking Cookies"  Imagine we are teaching a robot how to bake cookies. We give it a recipe, and it tries baking. Each time, it adjusts things like:   * How much sugar to use * How long to bake them * How much chocolate to add   These adjustable settings are like **parameters**. The more parameters it has, the more precisely it can tweak the recipe to make the perfect cookie.  In an LLM, instead of cookies, it’s learning **how to form sentences**. And instead of sugar and chocolate, it’s adjusting **billions of tiny knobs** that control how it understands and generates language.  In machine learning, a **parameter** is a value that the model learns during training. In LLMs:   * Each parameter is a **weight** in a neural network. * These weights determine how input words are transformed into output words. * The model adjusts these weights by analyzing **huge amounts of text** and minimizing errors in its predictions.   For example, GPT-3 has **175 billion parameters**, and GPT-4 has even more. The more parameters, the more nuanced and accurate the model can be—though it also requires more data and computing power. |

Examples of LLMs

|  |  |  |
| --- | --- | --- |
| **Model** | **Organization** | **Key Use** |
| GPT-4 | OpenAI | General purpose, ChatGPT |
| Claude | Anthropic | Helpful assistant, safety-focused |
| Gemini | Google | Multimodal AI |
| LLaMA | Meta | Open-source, research-focused |

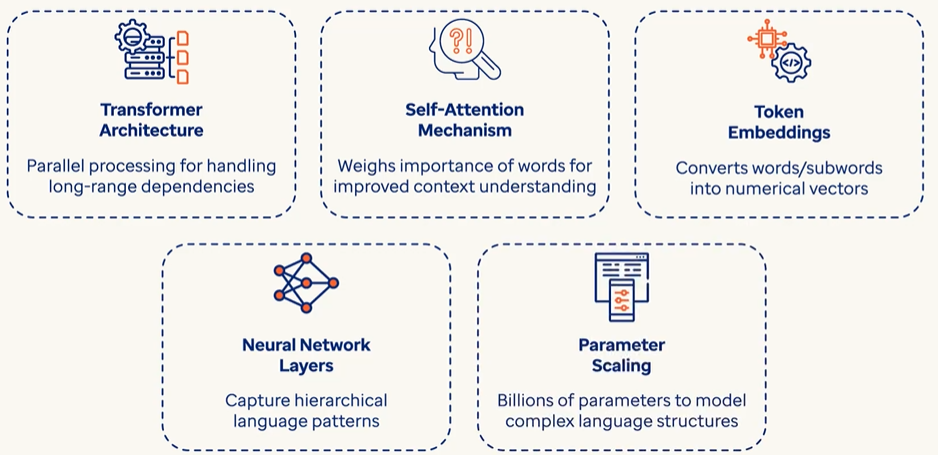
## LLM ARCHITECTURE

  
TRANSFORMER ARCHITECTURE

The **Transformer** is the core architecture behind most modern LLMs. It consists of two main parts:

1. ENCODER STACK
2. DECODER STACK

### CORE COMPONENTS OF LLMs



#### HOW ENCODERS FIT IN?

The **encoder** in a Transformer:

* Takes an input sequence (like a sentence).
* Processes it through **self-attention** to understand relationships between words.
* Outputs a **contextualized representation** of each token.

✅ Used in models like **BERT**, which are **encoder-only** and great for understanding tasks (e.g., classification, question answering).

##### EXAMPLE

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Let’s walk through a **simple example** to explain how the **encoder** in a Transformer works:  Input Sentence**: "The cat sat."**  Step 1: TOKENIZATION: The sentence is split into tokens: ["The", "cat", "sat"]  Step 2: Embedding + Positional Encoding : Each token is converted into a vector (embedding), and positional information is added so the model knows the order:  "The" → [0.1, 0.3, ...]  "cat" → [0.5, 0.2, ...]  "sat" → [0.4, 0.7, ...]  Step 3: Self-Attention : Now the encoder applies **self-attention**, which allows each word to "look at" the others and understand context.  For example:   * "cat" might attend to "The" to understand it's a subject. * "sat" might attend to "cat" to understand who is doing the action.   This step helps the model understand relationships like:   * Subject → "cat" * Verb → "sat" * Article → "The"   Step 4: Contextualized Representations : After self-attention and feed-forward layers, each token now has a **context-aware vector**:   |  |  | | --- | --- | | Token | Contextualized Vector (simplified) | | "The" | [0.12, 0.45, …] (knows it's an article for "cat") | | "cat" | [0.67, 0.88, …] (knows it's the subject of "sat") | | "sat" | [0.91, 0.34, …] (knows it's the action done by "cat") |   These vectors are the **encoder's output** — they’re like smart embeddings that understand the sentence structure and meaning. |

#### HOW DECODERS FIT IN?

The **decoder** in a Transformer:

* Takes the encoder's output (if present) and previously generated tokens.
* Uses **masked self-attention** to prevent peeking ahead.
* Uses **encoder-decoder attention** to focus on relevant input parts.

Outputs the next token in a sequence.

✅ Used in models like **GPT**, which are **decoder-only** and great for **text generation**.

##### EXAMPLE

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Let’s continue with the same example sentence: **"The cat sat."** Now imagine we want the Transformer to **generate this sentence** using the **decoder**.  Step 1: Input from Encoder  The **encoder** has already processed the sentence and produced **contextualized representations** for:   * "The" → [0.12, 0.45, ...] * "cat" → [0.67, 0.88, ...] * "sat" → [0.91, 0.34, ...]   These are passed to the **decoder**.  Step 2: Start Token   * The decoder begins with a special token like <START> to initiate generation.   Step 3: Masked Self-Attention  The decoder uses **masked self-attention** to look only at **previous tokens**, not future ones. This prevents it from "cheating" by seeing the full sentence ahead of time.  Example:   * At first step, it only sees <START> * At second step, it sees <START>, "The" * At third step, it sees <START>, "The", "cat" * And so on…   Step 4: Encoder-Decoder Attention  At each step, the decoder **attends to the encoder's output** to understand the context of the input sentence.  Example:   * When generating "cat", it looks at the encoder's representation of "The" and "cat" * When generating "sat", it attends to "The", "cat", and their relationships   Step 5: Output Token  The decoder predicts the next token in the sequence:   * <START> → "The" * "The" → "cat" * "cat" → "sat" * "sat" → <END>   Each prediction is based on:   * Previously generated tokens * Encoder's contextual output * Attention mechanisms   **Summary in Context of "The cat sat."**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Step | Decoder Input | Masked Self-Attention | Encoder-Decoder Attention | Output | | 1 | <START> | Only <START> | Looks at encoder output | "The" | | 2 | <START> The | <START>, "The" | Looks at encoder output | "cat" | | 3 | <START> The cat | All previous tokens | Looks at encoder output | "sat" | |

Encoder-Decoder Together

In full **Transformer models** (like **T5**, **BART**, or **original Transformer for translation**):

* The **encoder** reads the input (e.g., English sentence).
* The **decoder** generates the output (e.g., French translation), attending to the encoder's output.

LLM Variants Based on Transformer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model Type | Uses Encoder | Uses Decoder | Example Models | Best For |
| Encoder-only | ✅ Yes | ❌ No | BERT, RoBERTa | Understanding tasks |
| Decoder-only | ❌ No | ✅ Yes | GPT-2, GPT-3, GPT-4 | Text generation, chat |
| Encoder-Decoder | ✅ Yes | ✅ Yes | T5, BART, Marian | Translation, summarization |

# EMBEDDINGS

* Embeddings are numerical representations of text—a way to convert words into numbers so machines can understand and process them.
* Machines do not inherently understand text; they operate using numbers. Embeddings allow machine learning models to interpret meaning, context, and relationships between words.

**KEY FUNCTIONALITY OF EMBEDDINGS**

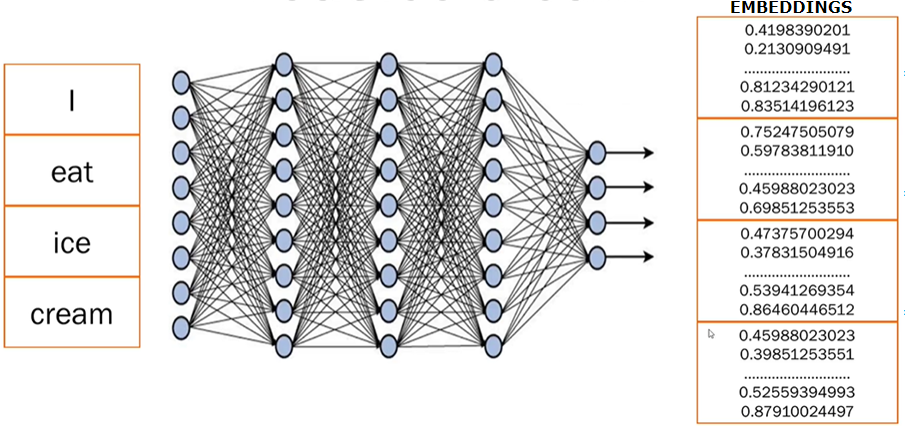
1. **CAPTURING MEANING**: Embeddings reflect the semantic meaning of words or sentences.
2. **CONTEXT UNDERSTANDING**: Embeddings account for the context of text (e.g., same word in different sentences/sentiments - “great” in sarcasm vs happiness).



1. **RELATIONSHIPS**:

* Words with strong associations (e.g., "ice" and "cream") have similar embeddings, reflecting their connection.
* Models understand the proximity(how close the words /sentences are) and sequence of words based on embeddings.

**HOW EMBEDDINGS ARE GENERATED**



1. **STEP 1:** **BREAKING TEXT INTO TOKENS**

* Sentences are broken into smaller pieces or tokens (e.g., splitting "I eat ice cream" into 4 tokens—"I", "eat", "ice", "cream").

2. **STEP 2: NEURAL NETWORK PROCESSING**:

* Trained **transformer models** analyze the text, generate embeddings, and capture meaning, context, and relations between tokens.

1. **STEP 3: NUMERICAL EMBEDDINGS:**

* Each token is converted into numerical data (random numbers).
* These numbers represent embeddings, storing all learned information about the word or sentence.
* Only the transformer model understands what these embeddings mean based on its training.

# VECTORS

A **vector** is a list of numbers that represents something—like a word, an image, or even a sentence—in a way that a computer can understand and work with.

|  |  |
| --- | --- |
| Simple Analogy | |
| Imagine we want to describe a fruit (say, an **apple**) using numbers:   * Sweetness: 8 * Crunchiness: 7 * Juiciness: 6   We could represent the apple as a vector: [8, 7, 6]  Now, a **banana** might be:[9, 3, 7]  These vectors help a computer compare fruits based on their features. |  |

In Language Models

When we talk about **words**, we turn them into vectors using **embeddings**. For example:

* “cat” → [0.12, -0.45, 0.88, ..., 0.03]
* “dog” → [0.10, -0.40, 0.85, ..., 0.05]

These vectors are **high-dimensional** (often 300 to 1,000+ numbers long) and capture the **meaning** of the word based on how it’s used in language.

## WHY ARE VECTORS USEFUL?

They allow computers to:

* **Compare** things (e.g., how similar two words or images are)
* **Search** by meaning (semantic search)
* **Cluster** similar items together
* **Feed data into machine learning models**

## VECTOR DATABASE

A diagram of a data processing process

AI-generated content may be incorrect.

* A **vector database** is a special kind of database designed to store and search **vectors (***which are just lists of numbers that represent things like text, images, or audio in a way that computers can understand*.)
* Primarily used for storing embeddings that represent complex data like images, text and audio in a form that machine can understand and process

Why Vectors?

* When you use **embeddings** (like we discussed earlier), we turn data (like the word *“cat”*) into a vector, such as:

**[0.12, -0.45, 0.88, ..., 0.03]**

* These vectors capture **meaning** and **context**. But once we have millions of them, we need a smart way to **store** and **search for** them efficiently. That’s where vector databases come in.

What Does a Vector Database Do?

It helps to:

* **Store** millions or billions of embeddings
* **Search** for the most similar vectors (e.g., “find texts similar to this one”)
* **Rank** results by similarity (using distance metrics like cosine similarity)

Real-World Example

Let’s say we run a **document search engine**:

1. We convert all the documents into vectors using an LLM.
2. We store those vectors in a vector database (like **Pinecone**, **Weaviate**, **FAISS**, or **Milvus**).
3. When a user asks a question, we:
   * Convert the question into a vector
   * Search the database for the **most similar document vectors**
   * Return the most relevant documents

# HOW LLMS WORK

A diagram of a network

AI-generated content may be incorrect.

1. Training Phase

* The LLM is trained using **self-supervised learning**: it learns by predicting **missing words** in a sentence.
* For example: "The cat sat on the \_\_\_." → Model tries to guess "mat".
* It sees **billions or trillions of words** from diverse sources, learning grammar, facts, reasoning patterns, and even some coding skills.

2. Tokenization

* Text is broken down into **tokens** (which may be words, subwords, or characters).
* For example, “ChatGPT is smart” → ["Chat", "G", "PT", "is", "smart"].

3. Transformer Architecture

* A special deep learning model that uses **attention mechanisms** to understand the context of each word in relation to others in the sentence.
* For example, it knows the word “bank” can mean money or river, depending on context.

4. Inference (When You Use It)

* When you type a prompt, the LLM:
  + Converts your input into tokens.
  + Uses its trained model to predict the **next best token**.
  + Repeats until it completes a meaningful output.
* This happens very fast — like autocomplete on steroids.

What Can LLMs Do?

Answering questions

* Write essays, emails, or stories
* Translate languages
* Generate code
* Summarize documents
* Act as chatbots
* Reason through logic puzzles (to some extent)

Why it feels like LLMs are answering questions

* It feels like an LLM is answering questions because it has learned to predict the next token in a way that mimics intelligent responses — by learning patterns from millions of real questions and answers.

|  |  |  |
| --- | --- | --- |
| When we ask:  > "What is the capital of France?" | The model sees a familiar pattern. It has seen many examples like this during training:  Q: What is the capital of France?  A: Paris  Q: What is the capital of Germany?  A: Berlin | So, it has learned that when someone types:  > "What is the capital of \_\_\_?"  The best next tokens are usually:  > " [Country name]" → "?" → " A: [Capital]"  So it predicts:  " Paris"   * That’s it. It doesn’t "know" what Paris is — it just has seen that people answer that question with “Paris” so many times that it becomes the most likely token sequence. |
| **Chain of Tokens Feels Like Thought**  Let’s say we ask:  > "Can you explain black holes?" | It starts predicting:  "Sure! A black hole is a region in space..."  Then it keeps going with:  "...where gravity is so strong that not even light can escape..."   * Each time, it's predicting the next most likely token, based on what it has already said and what it's seen in similar explanations before. | * This sequence of predictions sounds natural, structured, and smart — because it's imitating how humans write or speak. * It's Like a Super Autocomplete |

Analogy: The Super Parrot with a Giant Memory

Imagine a **super parrot** named **GPTy** who has:

1. **Read every book**, website, chat, and textbook ever written.
2. **Doesn’t understand the world**, but **remembers how humans talk** — sentence by sentence.

Now, we ask the parrot: “What is the capital of Japan?"

* GPTy searches its memory and remembers **100,000+ times** someone asked that same question, and people always replied: "Tokyo."
* So it just **repeats the most likely answer** it has seen:→ **"Tokyo"**

|  |  |
| --- | --- |
| **Simple Simulation (in Code)**  Let’s simulate a tiny LLM that only learned how to respond to one question:  def tiny\_model(prompt):  if prompt == "What is 2 + 2?":  return "4"  elif prompt.startswith("What is"):  return "I'm not sure, but maybe check a calculator!"  else:  return "Can you ask that again?" | print(tiny\_model("What is 2 + 2?")) # → 4  print(tiny\_model("What is 10 x 10?")) # → "I'm not sure..."  print(tiny\_model("Tell me a joke.")) # → "Can you ask that again?"  That’s basically what a real LLM does — except:   * It doesn't have **if-else** rules. * It has **math-based probabilities** to guess the next best word/token. * It was trained on **terabytes** of data, not 3 lines like our tiny version. |

## PROMPTS & TOKENS

A diagram of a cream line

AI-generated content may be incorrect.

**WHAT IS A PROMPT?**

* A **prompt** is the **input** we give to a language model — it's the question, instruction, or text we type to get a response. We Think of it like a **conversation starter** or a **command**.It can be a single word, a sentence, or a long paragraph.

**WHAT IS A TOKEN?**

* A **token** is a **chunk of text** — usually a word or part of a word — that the model processes. It is numerical representation (converted by Tokenizer)of word or parts of word , phrases or a character
* Tokens can be as short as one character or as long as one word.
* For example:
  + "ChatGPT" → 1 token
  + "unbelievable" → might be split into ["un", "believ", "able"] → 3 tokens
  + "I am happy." → 4 tokens (["I", " am", " happy", "."])
* Most models (like GPT-4) use a tokenizer to split text into tokens. The number of tokens affects:
  + **Cost** (for API usage)
  + **Speed**
  + **Context limit** (e.g., GPT-4-turbo can handle up to 128k tokens)

|  |
| --- |
| * The set of all tokens used by the model is called the vocabulary of the model * The process of splitting text into tokens is called tokenization. |

**TOKENS IN CHAT GPT**

|  |  |
| --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect. | Open the URL: <https://platform.openai.com/tokenizer>  Enter the desired prompt  It will show how many has been created for a given prompt along with attention score (based on color code)  Note : Each model tokenize the prompt differently as they use different tokenizers |

### TOTAL TOKENS

* Tokens are numerical representation of characters, words or phrases

|  |
| --- |
| * Tokens are a fundamental metric for measuring usage in an AI system. * **Total tokens = Input tokens (**The number of tokens in the prompt or message you send**) + Output tokens(**The number of tokens in the model's response**)**   . **WHY IT MATTERS?**   * Language models have a **token limit** per interaction (e.g., 8,000 or 32,000 tokens depending on the model). * If the total number of tokens exceeds the limit, the model may truncate the input or fail to generate a complete response. |

#### CONTEXT WINDOW

A row of blue circles

AI-generated content may be incorrect.

* The context window refers to the maximum number of tokens (input + output) that the model can "see" or process at one time.
* The context window includes:
  + **Your input (prompt, messages, instructions)**
  + **The model’s output (response)**
  + **Any previous conversation history (if it's part of the current session)**
* Different models have different context window sizes. For example:
  + GPT-3.5: ~4,096 tokens
  + GPT-4 (standard): ~8,192 tokens
  + GPT-4 Turbo: up to 128,000 tokens

**WHY DO IT MATTERS?**

A black background with white lines and yellow text

AI-generated content may be incorrect.

* If the conversation exceeds the context window, older parts of the conversation may be truncated or forgotten.
* This affects the model’s ability to maintain long-term coherence or remember earlier details.
* For example, a LLM with context window of 10K, which is fed by an article of 15K token – it will truncate the token after 10K tokens of the Article. Hence the long documents may need to be chunked to fit within the context window of the LLM

# MODEL TYPES (LLM TYPES)

## CLASSIFICATION BASED ON – HOW THEY ARE TRAINED

A diagram of a software development process

AI-generated content may be incorrect.

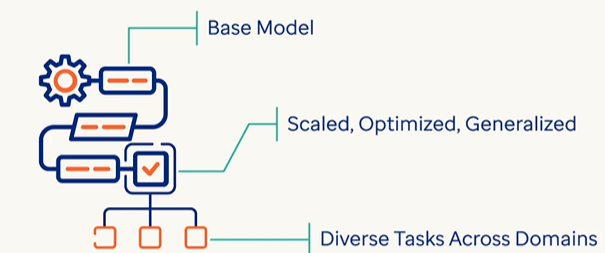
BASE MODEL

|  |  |
| --- | --- |
|  | * Base MODEL: The base model is outcome of pre-training phase on a large corpus of text using self-supervised learning (e.g., predicting the next word). * These models can process language but not optimized for further adaptation. |

|  |
| --- |
| * Self-supervised learning is a type of machine learning where a model learns from unlabeled data by creating its own labels from the data itself.   **HOW IT WORKS**  Imagine a sentence:  "The cat sat on the \_\_\_."  The model is trained to predict the missing word ("mat") using the rest of the sentence. No human labeled this data — the model learns by solving puzzles it creates from raw text.  **KEY FEATURES:**   * No manual labeling needed. * The model learns patterns, grammar, and meaning from large text corpora. * Used heavily in training base models like BERT, GPT, etc. |

* + Purpose: Learns general language patterns, grammar, facts, and reasoning.
  + Example: GPT-3 before any fine-tuning.
  + Limitation: Not optimized for specific tasks or instructions.

Foundational Model



* A broader term that includes base models and other large-scale models trained on diverse data.
* Built on base model – which are scaled , optimized for multitask adaptability
* It can perform diverse task across multiple domains without requiring task specific training.
  + Purpose: Acts as a foundation for building more specialized models.
  + Example: PaLM, GPT-4, LLaMA—used as starting points for downstream tasks.
  + Note: All base models are foundational, but not all foundational models are used as-is.

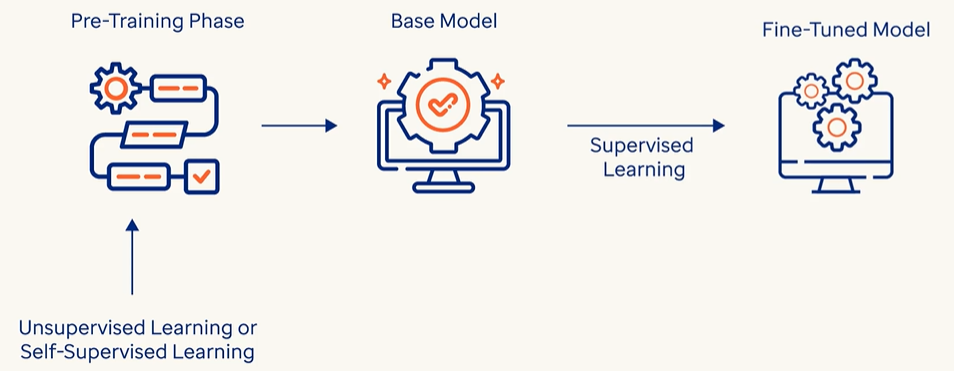
Instruction-Tuned MODEL



* A foundational or base model further trained to follow natural language instructions.
  + Purpose: Makes the model more helpful, safe, and aligned with human intent.
  + How: Trained on datasets like “prompt → response” pairs.
  + Example: InstructGPT, ChatGPT.

Fine-Tuned Model

* A model adapted to perform specific tasks or domains (e.g., legal, medical, customer support).
* The base model is further tuned using dataset using supervised learning.

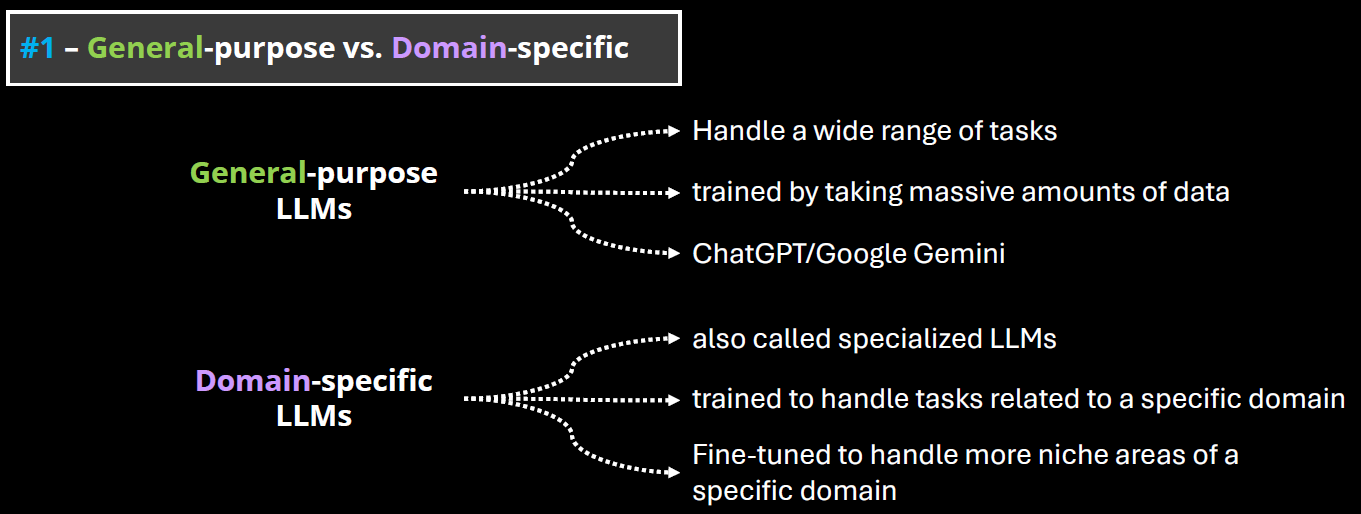


* Purpose: Improves performance on narrow use cases.
* How: Trained on labeled data or domain-specific examples.
* Example: A GPT model fine-tuned for legal document summarization.

## CLASSIFICATION BASED ON – HOW THEY ARE USED

### GENERAL PURPOSE AND DOMAIN-SPECIFIC LLMS

The distinction between General Purpose LLMs and Domain-Specific LLMs lies in their training data, capabilities, and intended use cases.:



#### GENERAL PURPOSE LLMS

* These are large language models trained on a broad and diverse dataset that spans many domains (e.g., science, literature, law, medicine, pop culture, etc.).
* Examples: GPT-4, Claude, Google Gemini, LLaMA.
* Strengths:
  + Versatility: Can handle a wide range of tasks (e.g., summarization, translation, coding, creative writing).
  + Adaptability: Can generalize well across different topics and user needs.
  + Scalability: Useful in applications where domain-specific knowledge is not required.
* Limitations: May lack deep expertise in specialized fields.

#### DOMAIN-SPECIFIC LLMS

* These are LLMs trained or fine-tuned on specialized datasets from a particular field (e.g., legal, medical, financial, scientific).
* Examples: Med-PaLM (medical), FinGPT (finance), BioGPT (biomedical), Legal-BERT (legal texts)
* Strengths:
  + High accuracy in domain-specific tasks.
  + Better understanding of terminology, context, and nuances in the field.
  + Often used in regulated industries where precision is critical.
* Limitations:
  + Limited generalization outside their domain.
  + May require frequent updates to stay current with domain knowledge.
  + Less flexible for multi-domain tasks.

#### OPEN AND CLOSED SOURCE LLMs

**The distinction between Open-Source and Closed-Source LLMs revolves around accessibility, transparency, control, and community involvement.**

**A diagram of a source

AI-generated content may be incorrect.**

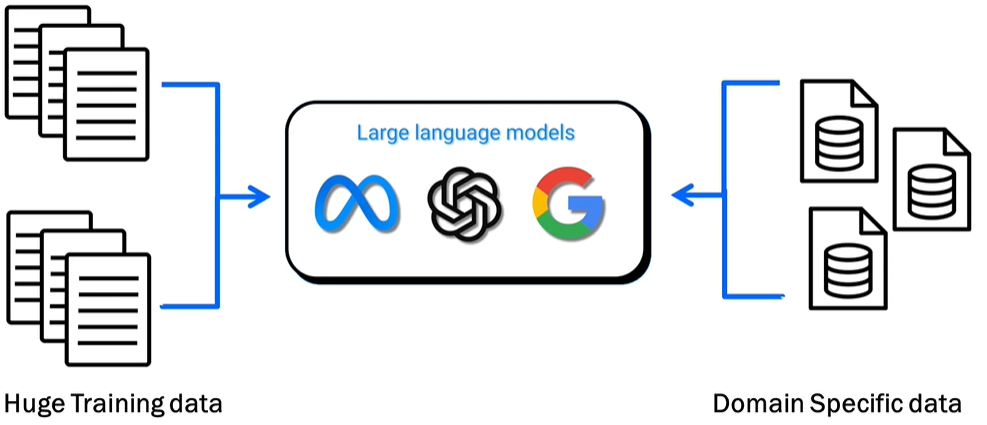
##### OPEN-SOURCE LLMS

* These models have their code, weights, and training data (or methodology) publicly available. Anyone can inspect, modify, or fine-tune them.
* Examples: Meta’s LLaMA (e.g., LLaMA 2, LLaMA 3), Mistral, Falcon,OpenChat, BLOOM (by BigScience)

CLOSED-SOURCE LLMS

* These models are proprietary. The model weights, training data, and architecture details are not publicly available.
* Examples: OpenAI’s GPT-4, Anthropic’s Claude, Google’s Gemini, , Cohere Command R+,Amazon Titan

## FINE TUNING



WHAT IS FINE TUNING

* Adjusting a pre-trained large language model (LLM) to provide better results on specific tasks or domain-specific datasets (e.g., healthcare, finance, etc.).

WHAT IS THE PURPOSE OF FINE TUNING

* Enhance the model's ability to generate focused and precise results on specialized datasets.
* Example: Fine-tuning an LLM on medical data improves its accuracy in answering medical queries compared to a general-purpose/pre-trained model.

### FINE TUNING TECHNIQUES

**1. Full Fine-Tuning**

**What it is**: Updating all model parameters using labeled data.

**Pros**: High flexibility and performance gains.

**Cons**: Computationally expensive, risk of overfitting, requires large datasets.

**Use case**: Domain-specific models (e.g., legal, medical).

**🧩 2. Adapter-Based Fine-Tuning**

**What it is**: Introduces small trainable modules (adapters) between layers of the frozen base model.

**Pros**: Efficient, modular, avoids catastrophic forgetting.

**Popular variants**: AdapterFusion, Compacter.

**Use case**: Multi-task learning, low-resource environments.

**🧠 3. LoRA (Low-Rank Adaptation)**

**What it is**: Injects low-rank matrices into the attention layers to reduce the number of trainable parameters.

**Pros**: Lightweight, fast training, minimal memory footprint.

**Use case**: Personalization, rapid prototyping.

**🧵 4. Prefix Tuning / Prompt Tuning**

**What it is**: Learns a fixed set of tokens (prefix or prompt) that steer the model’s behavior.

**Pros**: Extremely parameter-efficient, fast.

**Cons**: Limited flexibility compared to full fine-tuning.

**Use case**: Task-specific tuning, few-shot learning.

**🧪 5. Instruction Tuning**

**What it is**: Fine-tuning on datasets where tasks are framed as instructions.

**Pros**: Improves generalization across tasks, aligns model behavior with human intent.

**Use case**: Chatbots, general-purpose assistants.

**🧬 6. Reinforcement Learning from Human Feedback (RLHF)**

**What it is**: Uses human preferences to guide model outputs via reinforcement learning.

**Pros**: Aligns model with human values, improves helpfulness and safety.

**Cons**: Complex pipeline, expensive to scale.

**Use case**: Alignment-focused models like ChatGPT.

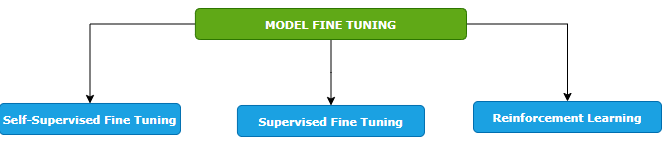
**🧮 7. Quantization-Aware Fine-Tuning**

**What it is**: Fine-tuning while converting model weights to lower precision (e.g., INT8).

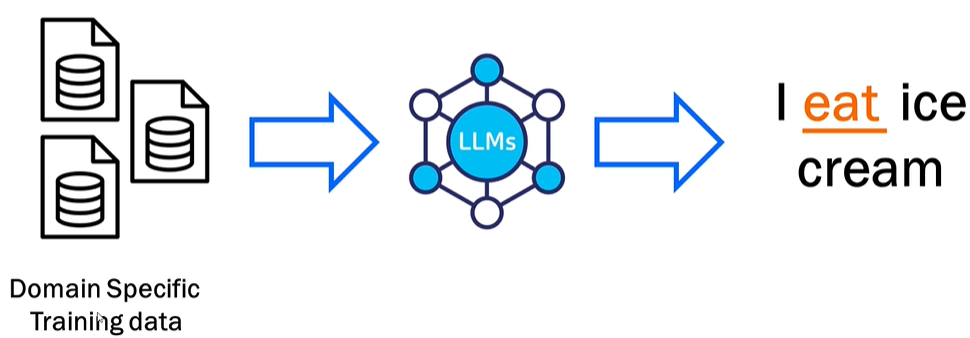
**Pros**: Reduces model size and inference cost.

**Use case**: Edge deployment, mobile devices.

### DIFFERENT WAYS TO FINE TUNNING A MODEL



#### SELF-SUPERVISED FINE TUNING



* Training the model on a domain-specific dataset. Means – we give the foundation model a big pile of training data that is specific to our domain, and the model will learn from it. In this way, the model learns to predict missing pieces of data.
* Example - when we say I ice cream, the model predicts that the missing word is eat.
* This is like how the foundation model is trained but the key difference here is that we are fine tuning the model by providing the domain specific data set. Example - Like if we want to train it on health care data set, we will pass it - the drug structure, scientific studies, all the documents that are related to drug and the model will learn from it and it would be able to generate content based on that.

#### SUPERVISED FINE TUNING

A screenshot of a computer

AI-generated content may be incorrect.

* The model is trained using a labeled dataset where both inputs and outputs are provided.
* Example: Input: "How do I find a broken bone?" → Output: "X-ray."
* Helps the model learn more precise responses based on labeled data.

#### REINFORCEMENT LEARNING

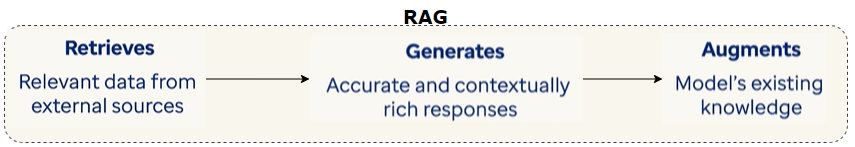
|  |  |
| --- | --- |
| A blue arrow with black text  AI-generated content may be incorrect. | A blue arrow pointing to a black arrow  AI-generated content may be incorrect. |
| **LOW SCORE FOR BAD RESULT** | **HIGH SCORE FOR GOOD RESULTS** |

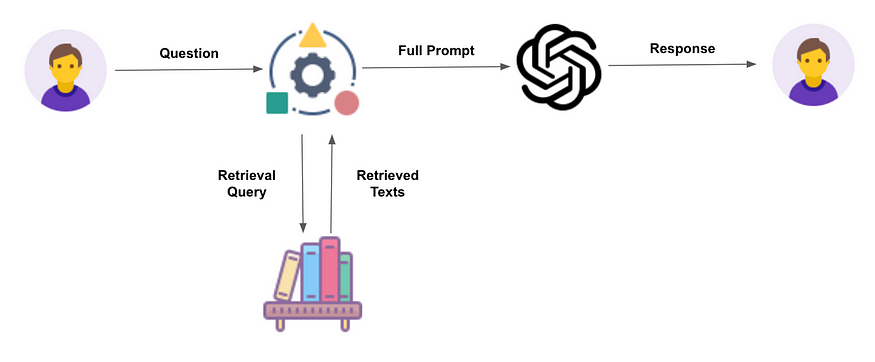
* **Feedback-based learning**.
* The model generates outputs, and scores are assigned based on quality (high score for good results, low score for bad results).
* The model learns overtime from the feedback to improve predictions.

#### KEY FEATURES OF FINE TUNING

1. **STARTS FROM A PRE-TRAINED MODEL**: Fine tuning builds on top of a foundation model already trained on large datasets—it does not start from scratch.
2. **REQUIRES DOMAIN-SPECIFIC DATA**: We must provide good-quality, specific data for training tailored to your use case (e.g., drug data for healthcare).
3. **No Universal SOLUTION**: Each task/use case is unique, requiring case-specific implementation and variations.
4. **ITERATIVE PROCESS**: Fine tuning is repetitive and requires multiple cycles of iteration and adjustments for optimal results.

# RAG (RETRIEVER AUGUMENTED GENERATION)





* **Retriever**: Fetches relevant information from a knowledge base.
* **Augmented**: Adds value by combining retrieved data with generative capabilities.
* **Generation**: Produces a response using a language model (e.g., ChatGPT, Gemini).

## RAG WORKFLOW

|  |
| --- |
| Step 1: Ask a Question –   * When we ask a question. The question goes into a knowledge base (like a smart library). * The knowledge source can be Databases, Articles or Websites * This knowledge base is often a vector database, which stores information in a format that helps find similar meanings.   Step 2: It Finds Relevant Text   * The system retrieves the most relevant documents or pieces of information that match the question. In the vector Database. This is called the retrieval step.   Step 3: It Builds a Full Input   * The retrieved information is combined with the original question. * This combination becomes a full prompt or input for the next step.   Step 4: AI Generates the Answer   * **Full prompt is sent to a Language Model (like ChatGPT or Gemini).** * **Note**: Retrieved data + original query = **prompt** for the **Language Model (LM)**. * The model reads both questions and the retrieved info and then generates a smart answer. * The response is based on internal data (from the knowledge base) and the intelligence of the AI model. |

## WHY RAG?

**RAG is important due to the following limitations of LLMs**

1. KNOWLEDGE CUT-OFF DATE
   1. The LLM model will have information up to their training cut-off date and lack of information beyond that point.
   2. Hence LLMs **can’t access real-time updates** or dynamic data, and they may miss recent changes in the organization or product.
2. LACK OF ACCESS OF ENTERPRISE DATA
   1. They lack access of enterprise specific data unless they are fine-tuned for customized for that enterprise

|  |
| --- |
| * Let’s say in banks or an enterprise company, the data will be sitting privately within companies private infrastructure or within companies’ network. * In this use-case - We don't want ChatGPT or any other LLMs to have information about private data. So that is where vector databases come into picture where the questions retrievals will happen from the company’s internal source. * Then – we are just using the capabilities of LLM to prepare a nice prompt to give a proper reply. |

Enterprise Use Cases

* **Customer support**: Pulls product details from internal DBs.
* **Educational tools**: Provides precise, sourced answers.
* **Banking/Enterprise**: Keeps sensitive data internal while leveraging LMs.

## BENEFITS

Access to Private/Internal Data

* RAG allows LLMs to use your organization’s internal documents (e.g., product manuals, policies, reports) that are not part of public training data.
* This makes responses more relevant and accurate for enterprise use.

Reduces Hallucinations

* LLMs sometimes make up answers when they don’t have enough information.
* RAG reduces this by grounding responses in real, retrieved documents.

Improves Contextual Accuracy

* RAG retrieves context-specific information before generating a response.
* This ensures the answer is tailored to the user’s query and environment.

Keeps Data Secure

* Sensitive data stays within private infrastructure.
* The LLM only sees the retrieved content, not the entire database—helping with data privacy and compliance.

Dynamic and Up-to-Date Responses

* Instead of relying on static training data, RAG can pull real-time or recently updated documents.
* This makes it ideal for fast-changing domains like tech support or policy updates.

## RAG ARCHITECTURE

A diagram of a computer system

AI-generated content may be incorrect.Step 1: Prepare Your Data

* Collect documents, images, videos, etc.
* Send them to an **Embedder**, which converts them into a format (vectors) that computers can search easily.

Step 2: Store the Data

* The Embedder sends these vectors to a **Vector Storage and Retrieval Engine** (like a smart library).

Step 3: User Asks a Question

* A person types a question into a **chat interface**.

Step 4: Process the Question

* The question goes to a **User Query module**, then to the **Embedder** to be converted into a vector (just like the data was).

Step 5: Search for Matching Info

* The query vector is sent to the **Vector Storage and Retrieval Engine**.
* It searches for the most relevant information based on meaning (semantic search).

Step 6: Generate the Answer

* The retrieved information is sent to a **Large Language Model** (like ChatGPT).
* The model uses both the question and the retrieved info to create a smart, accurate response.

Step 7: Show the Answer

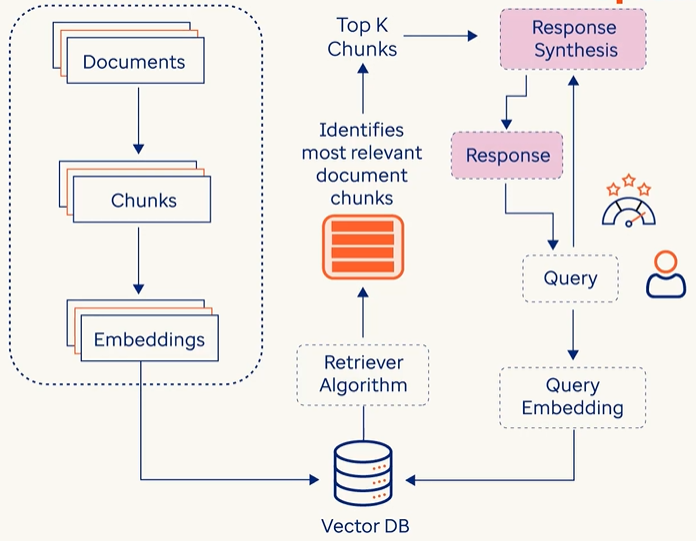
* The response is sent back to the **chat interface**, where the user sees the final answer.

## EXAMPLES

|  |
| --- |
| Simple Example   1. **User asks a question** → e.g., *"What are the benefits of using Kubernetes?"* 2. **Retriever fetches relevant documents** → From a knowledge base, database, or web search. 3. **Generator uses those documents to answer** → The LLM generates a response using the retrieved context. |
| Healthcare Chatbot for Prescriptions  **Use Case**: A patient asks:  *"Can I take ibuprofen with Metformin?"*  **RAG Flow**:   * **Retriever** finds a medical FAQ: *"Ibuprofen may increase the risk of kidney issues when taken with Metformin."* * **Generator** responds:   *"It’s advised to consult a doctor before combining ibuprofen with Metformin, as it may affect kidney function."*  ✅ **Grounded in real medical data**, not just model memory. |
| Enterprise Knowledge Assistant  **Use Case**: An employee asks:  *"What is our company’s travel reimbursement policy?"*  **RAG Flow**:   * **Retriever** pulls from internal HR documents. * **Generator** summarizes the policy:   *"Employees can claim travel expenses up to ₹10,000 per trip with prior approval from their manager."*  ✅ **Uses internal documents**, not public data. |
| Academic Research Assistant  **Use Case**: A student asks:  *"What are recent advancements in quantum computing?"*  **RAG Flow**:   * **Retriever** pulls from arXiv papers or IEEE articles. * **Generator** summarizes:   *"Recent work includes quantum error correction using surface codes and improvements in superconducting qubit coherence times."*  ✅ **Up-to-date and citation-ready**. |
| E-commerce Support Bot  **Use Case**: A customer asks:  *"What’s the return policy for electronics?"*  **RAG Flow**:   * **Retriever** finds the policy page. * **Generator** replies:   *"Electronics can be returned within 15 days if unopened and in original packaging."*  ✅ **Accurate and policy-compliant**. |

## RAG FRAMEWORK





|  |  |
| --- | --- |
| DATA INGESTION | * In this phase, the system gathers large volume of data from different data sources such as Databases, documents, API or online content * The main goal of data ingestion is to prepare external knowledge in a format that makes the system easier for retrieve and synthesis later * This knowledge comes for structure data like DB tables, unstructured data like text files or even programmatically generated by API calls |
| INDEXING & STORING | * At this stage, each document is split into multiple chunks from which corresponding embeddings (A numerical vector representation of textual chunks, which captures meanings and context) are created * Once the embeddings are created, each chunk along with its embeddings is stored and indexed in vector database |
| RETRIEVAL | * The primary objective of this step is to find relevant documents that has answers to user’s query submitted in natural language * The Query is the converted to embeddings to the vector DB * The Retriever Algorithm, then compare the user’s query and the embedding already stored in vector database * The Algo identified the most relevant document chunks which is like user query embeddings and finally it retrieves the Top K relevant chunks for Vector DB, which are the pieces of information which can answer user’s query |
| RESPONSE SYNTHESIS | * In this phase the Top K chunks and user’s query is passed Response Synthesis Module * This module has LLM which provides the response in natural language |
| QUERY OR CHAT ENGINE |  |
| EVALUATION |  |

## TOOLS TO BUILD RAG SYSTEMS

## LANGCHAIN

* **LangChain** is an open-source framework designed to help developers build powerful applications using **Large Language Models (LLMs)** like ChatGPT, Gemini, Claude, etc., by connecting them with **external data sources, tools, and workflows**.
* It is designed for integrating LMS or language models into applications.
* **LangChain Can be used to implement RAG**

|  |
| --- |
| ANALOGY   * Just like LEGO bricks can be assembled using instructions or imagination:   + LangChain provides **pre-built modules** for common tasks.   + Developers can **customize and combine** these modules to build complex AI workflows. * It’s **modular and flexible**, making it easy to mix and match components. |

**Why LangChain Is Useful**

* **Simplifies AI integration** into apps—no need to build from scratch.
* **Developer-friendly**: Reduces complexity of building AI-powered tools.
* Supports **multi-LLM setups** (e.g., combining ChatGPT and LLaMA).
* Enables **custom workflows** like chatbots, agents, and automation.

|  |
| --- |
| Example: Weather App with LangChain   * A user asks: “What’s the weather in Chandigarh today?” * LangChain:   1. Understands the query needs **real-time weather data**.   2. Calls an **external API or database** to fetch the latest info.   3. Passes the data to an LLM to **generate a natural response**.   4. Delivers: “It’s sunny and 40°C in Chandigarh today.” |

# AGENTIC AI

**Agentic AI** refers to AI systems that can **autonomously make decisions**, **take actions**, and **pursue goals** with minimal human intervention. These systems behave like **agents** — they don’t just respond to prompts, they **act independently** based on objectives, rules, and environmental feedback.

## EXAMPLES

|  |
| --- |
| AI Health Assistant (Personal Agent)  Scenario: A patient says: "Help me manage my diabetes."  *Agentic AI Actions:*   1. Retrieves your medical history and prescriptions. 2. Sets reminders for medication and meals. 3. Orders refills from the pharmacy. 4. Alerts your doctor if blood sugar readings are abnormal. 5. Adjusts your diet plan based on recent activity.   ✅ Autonomous, goal-driven, multi-step behavior. |
| AI Executive Assistant (e.g., AutoGPT)  Scenario:You say: "Plan a business trip to Mumbai next week."  *Agentic AI Actions:*   1. Searches for flights and books the best one. 2. Reserves a hotel near your meeting location. 3. Schedules meetings with clients. 4. Adds everything to your calendar. 5. Sends confirmation emails.   ✅ It acts like a human assistant, not just a chatbot. |
| 3. DevOps Agent  Scenario: "Monitor my app and scale it if traffic spikes."  *Agentic AI Actions:*   * Monitors server metrics. * Detects a traffic spike. * Automatically scales up resources. * Sends a Slack alert to the team. * Rolls back if errors increase.   ✅ Autonomous infrastructure management. |
| E-commerce Store Manager Bot  Scenario:"Manage my online store."  *Agentic AI Actions:*   1. Updates product listings. 2. Adjusts prices based on competitor data. 3. Responds to customer queries. 4. Flags suspicious orders. 5. Launches a weekend sale campaign.   ✅ Acts with initiative and adapts to changing conditions. |

## MULTI-AGENT AGENTIC AI?

* **Multi-Agent Agentic AI** is an advanced design pattern where **multiple autonomous AI agents** collaborate (or sometimes compete) to solve **complex, multi-step problems**. Each agent is **specialized**, goal-driven, and capable of acting independently — but they also **communicate and coordinate** with each other to achieve a shared objective.

|  |  |
| --- | --- |
| Concept | Description |
| Agentic AI | AI that can act autonomously to achieve goals |
| Multi-Agent System (MAS) | A system where multiple agents interact to solve problems |
| Specialization | Each agent is skilled in a specific domain or task |
| Coordination | Agents share information and plan together |
| Autonomy | Agents can make decisions and take actions without human input |

Example 1: Software Development Team (AI Agents)

Imagine building a web app using only AI agents:

|  |  |
| --- | --- |
| Agent | Role |
| 🧠 Planner Agent | Breaks down the project into tasks |
| 💻 Coder Agent | Writes backend and frontend code |
| 🎨 Designer Agent | Creates UI/UX mockups |
| 🧪 Tester Agent | Writes and runs test cases |
| 📦 DevOps Agent | Deploys the app to the cloud |

They **collaborate like a real team**, using tools like GitHub, CI/CD pipelines, and Slack.

Example 2: Healthcare Automation

|  |  |
| --- | --- |
| Agent | Task |
| 🩺 Diagnosis Agent | Analyzes symptoms and suggests possible conditions |
| 💊 Prescription Agent | Recommends medications based on diagnosis |
| 📅 Scheduling Agent | Books appointments and follow-ups |
| 📈 Monitoring Agent | Tracks patient vitals and alerts doctors |

Together, they provide **end-to-end patient care** with minimal human intervention.

Example 3: Travel Planning Assistant

User: *"Plan a 5-day trip to Japan under ₹1,00,000."*

|  |  |
| --- | --- |
| Agent | Task |
| ✈️ Flight Agent | Finds affordable flights |
| 🏨 Hotel Agent | Books budget-friendly hotels |
| 📍 Itinerary Agent | Plans daily activities |
| 💳 Budget Agent | Ensures total cost stays within limit |

They **negotiate trade-offs** (e.g., cheaper hotel = better flight) and deliver a complete plan.

Artificial Neural Networks (ANNs)

A diagram of a diagram

AI-generated content may be incorrect.

An ANN is made up of layers of **nodes (neurons)**:

1. **Input Layer** – Takes in the raw data/features (e.g., pixels of an image, words in a sentence).
2. **Hidden Layers** – Perform computations and extract features. There can be one or many of these.
3. **Output Layer** – Produces the final result (e.g., classification, prediction, generated content).

Input Layer – Where It All Begins

* This is the **first layer** of the network.
* It receives **raw data** like numbers, images, or text.
* The data is broken into **features** (e.g., for predicting apartment price: size, number of rooms, location).
* Each feature becomes an **input node** in the network.

Hidden Layers – The Brain of the Network

* These are the **intermediate layers** between input and output.
* They **process the input data** and extract deeper patterns or sub-features.
* More hidden layers = more ability to learn **complex relationships** in the data.

Output Layer – The Final Decision

* This layer gives the **final result** of the network.
* Depending on the task, it could:
  + Classify (e.g., spam or not spam),
  + Predict (e.g., apartment price),
  + Generate (e.g., new text or images).

Connections and Weights – How Learning Happens

* Every node is connected to others in the next layer.
* Each connection has a **weight**—a number that shows how important that connection is.
* During training, the network **adjusts these weights** to improve accuracy.
* For example, if “apartment size” is very important, its connection will have a **higher weight**.

Training the Network – Learning from Data

* The network learns by comparing its output to the correct answer and adjusting weights.This process is called **backpropagation**.
* Over time, the network learns which features are most important and how to combine them.