Generative AI (GenAI)

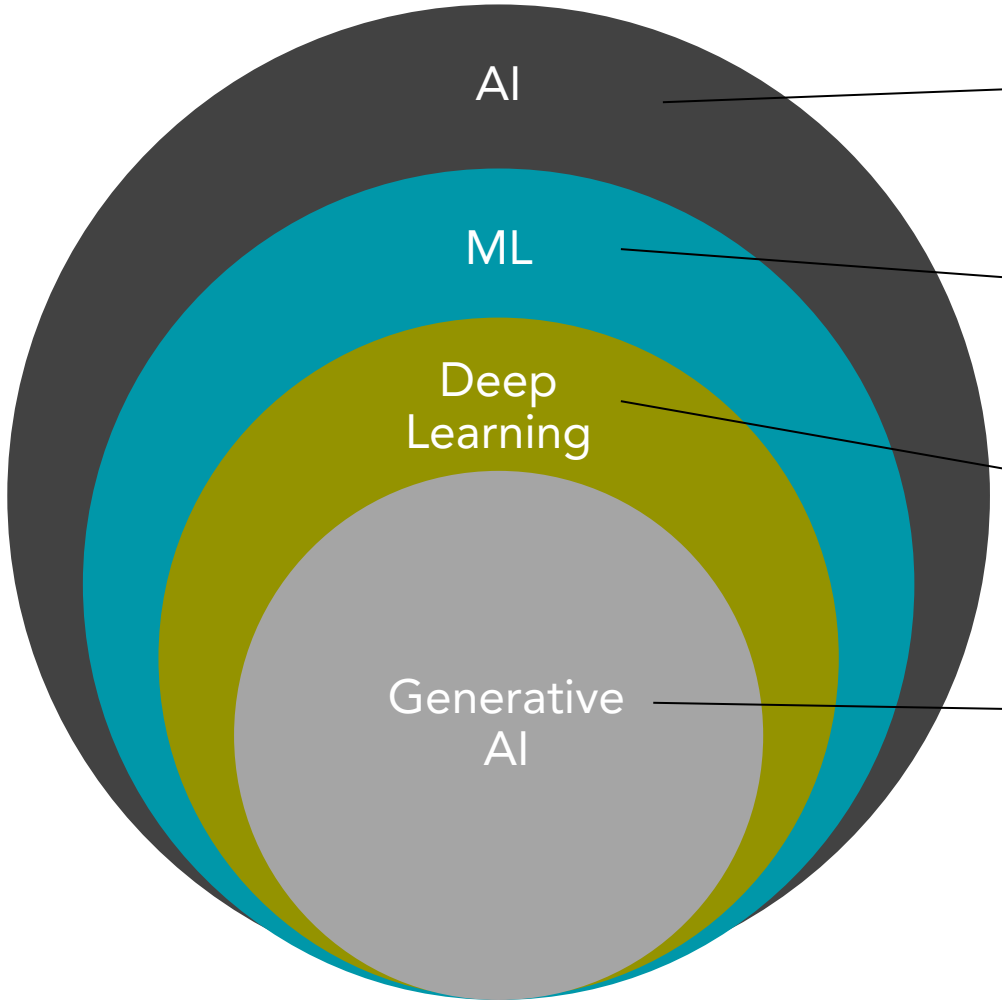
Shabab Akhter
External Lecturer
RUC

Contents

1. Background & fundamentals of GenAI
    a. What is GenAI & why is it so popular?
    b. Attention & Transformer models – how do they work
    c. Types of GenAI models – images, text, etc.
2. Transformer vs Natural Language Processing (NLP
    1. Explaining basics of NLP
    2. Basic vectorizing – TF-IDF
    3. Exercise: Modelling on text-based data using TF-IDF
4. Retrieval Augmented Generation (RAG) & Agentic workflows
    a. What is RAG & how does it work?
    b. What are agents & how do they work?
    c. Exercise: building a basic RAG application

Contents

Defining GenAI in relation to AI



Artificial Intelligence (AI) is a discipline - enables computer systems perform tasks normally requiring human intelligence

Machine Learning (ML) gives computers ability to learn without explicit programming

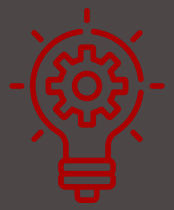
Deep Learning is a sub field of ML enabling processing of very complex patterns - using neural network models

Type of AI that creates new content (text, images, video etc.) based on learning from existing content

IT vendors massively embedding gen AI features in their product roadmaps after ChatGPT general availability

Conceptual overview of GenAI

Innovation



It enables the automation of creative tasks such as generating text, images, or music, leading to increased productivity and new possibilities

Revolution

GenAI is contributing to the ongoing digital transformation and shaping the future of industries like marketing, advertising and content creation

GenAI is a broad concept and touches many different domains

- **Text Generation** 📖
  - Chatbots & Virtual Assistants (e.g., ChatGPT, Bard)
  - Content Creation (e.g., copywriting, article drafting)
  - Code Generation (e.g., GitHub Copilot, OpenAI Codex)

- **Music & Audio Generation** 🎵
  - AI Composers (e.g., AIVA, Amper Music)
  - Voice Cloning & Text-to-Speech (e.g., ElevenLabs, Voicify)
  - Sound Design for Games & Media (e.g., Meta's AudioGen)

- **Video Generation & Editing** 🎥
  - AI-Generated Videos (e.g., Runway ML, Pika Labs)
  - Deepfakes & Face Animation (e.g., DeepFaceLab, Synthesia)
  - AI-powered Video Upscaling (e.g., Topaz AI)

- **Image Generation & Enhancement** 🖼️
  - AI Art & Design (e.g., Midjourney, DALL·E)
  - Photo Restoration & Editing (e.g., Adobe Firefly)
  - Medical Imaging Analysis (e.g., AI-assisted radiology)

- **Scientific & Engineering Applications** 🔬
  - Drug Discovery & Protein Folding (e.g., AlphaFold, Insilico Medicine)
  - Material Science Simulations
  - AI-assisted Code & Circuit Design

- **Gaming & Virtual Worlds** 🎮
  - AI-Generated Game Assets & Characters (e.g., Scenario.gg)
  - NPC Behavior Generation (e.g., Inworld AI)
  - AI Dungeon & Narrative-driven Games

Although GenAI can be applied to text, images & others, we will focus on text in this lecture

Under the hood, the main tech is a certain type of model – *Transformer Models*

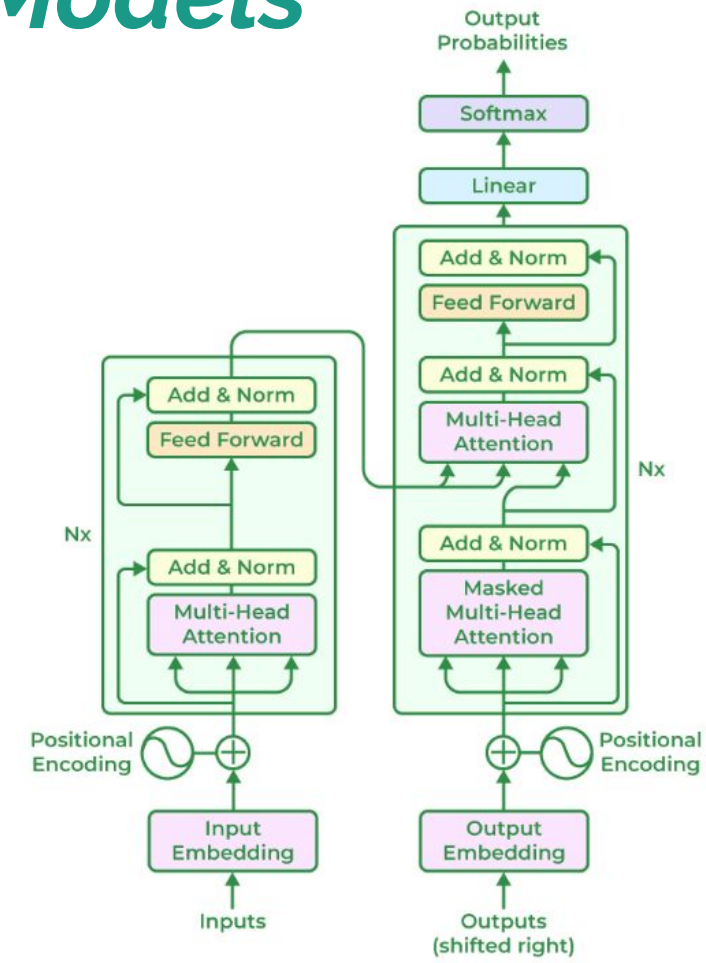
What are transformer models?

Transformer models are the foundation of modern Generative AI, enabling state-of-the-art performance in text, images, code, and more.

Originally introduced in the paper **"Attention is All You Need" (Vaswani et al., 2017)**, transformers have revolutionized AI by enabling parallel processing and long-range dependencies.

Core Components of a Transformer:

1. **Self-Attention Mechanism** – Determines which parts of the input are most relevant to each other.
2. **Positional Encoding** – Maintains word order information.
3. **Feedforward Layers** – Processes contextualized embeddings.
4. **Multi-Head Attention** – Enables the model to focus on different aspects of the input.



How do transformer models work?

Transformer models, like GPT4, are essentially generating words based on a given question/input. It literally generates one word at a time, i.e., given a question, it predicts what is the most likely first word in the answer, then the second word, then third and so on, until the answer is complete.

Let's take an example and go step by step

Step 1: User Input (Tokenization & Embedding)

🔹 **What Happens?**

- A user asks a **question**:
  "Who wrote Harry Potter?"
- The input is **tokenized** into smaller subwords:

```
css                                                    Copy  Edit

["Who", "wrote", "Harry", "Potter", "?"]
```

- These tokens are mapped to **word embeddings** (numerical vectors), which capture their semantic meaning.

Token	Embedding Vector (Simplified)
Who	[0.3, 0.7, -0.2, ...]
wrote	[0.5, -0.1, 0.8, ...]
Harry	[0.9, 0.2, -0.5, ...]
Potter	[0.7, -0.3, 0.6, ...]
?	[0.1, 0.4, -0.7, ...]

👉 **Why?**

- These embeddings allow the model to understand word relationships and context.
- Words with similar meanings (e.g., "write" and "author") have similar vectors.

How do transformer models work?

Transformer models, like GPT4, are essentially generating words based on a given question/input. It literally generates one word at a time, i.e., given a question, it predicts what is the most likely first word in the answer, then the second word, then third and so on, until the answer is complete.

Let's take an example and go step by step

Step 2: Adding Positional Encoding

🔹 What Happens?

- Since Transformers **don't process words sequentially** like RNNs, **positional encodings** are added to preserve word order.

👉 **Why?**

- Without this, "Who wrote Harry Potter?" could be interpreted the same as "Harry Potter wrote Who?", which makes no sense.

Token	Word Embedding (4D)	Positional Encoding (4D)
Who	[0.3, 0.7, -0.2, 0.5]	[0.0, 1.0, 0.0, 1.0]
wrote	[0.5, -0.1, 0.8, -0.3]	[0.84, 0.54, 0.91, 0.42]
Harry	[0.9, 0.2, -0.5, 0.1]	[0.91, 0.41, 0.99, 0.14]
Potter	[0.7, -0.3, 0.6, 0.4]	[0.14, 0.99, 0.27, 0.96]
?	[0.1, 0.4, -0.7, -0.2]	[0.99, 0.14, 1.0, 0.01]

Step 3: Self-Attention (Capturing Context)

🔹 What Happens?

- **Every word attends to every other word** in the sentence.
- The model calculates **attention scores** to determine which words are most relevant.

📌 Example:

For **"Who wrote Harry Potter?"**, attention values might look like:

Token	Attention Focus (Top Words)
Who	wrote, Harry Potter
wrote	Who, Harry Potter
Harry	Potter, wrote
Potter	Harry, wrote
?	Who, wrote

👉 **Why?**

- The model learns that **"Who"** and **"wrote"** are strongly related to the expected answer.
- "Harry Potter" is recognized as an **object** of the verb "wrote".

How do transformer models work?

Transformer models, like GPT4, are essentially generating words based on a given question/input. It literally generates one word at a time, i.e., given a question, it predicts what is the most likely first word in the answer, then the second word, then third and so on, until the answer is complete.

Let's take an example and go step by step

Step 4: Multi-Head Attention (Understanding Context in Multiple Ways)

🔹 What Happens?

- Multiple **attention heads** analyze different linguistic aspects of the input.
- Each head looks at **different relationships**, such as:
  - **Head 1:** Subject-Verb relationship (Who ↔ wrote)
  - **Head 2:** Object identification (Harry ↔ Potter)
  - **Head 3:** Interrogative emphasis ("Who" gets more weight)

👉 Why?

- This enables the model to **disambiguate meanings** and handle complex sentence structures.

Step 5: Feedforward Layer (Context Refinement)

🔹 What Happens?

- The self-attention output is passed through a **neural network layer** to refine understanding.
- The model builds a deeper representation of the input.

👉 Why?

- This step ensures that words aren't just **statistically related** but are **semantically and syntactically** aligned.

How do transformer models work?

Transformer models, like GPT4, are essentially generating words based on a given question/input. It literally generates one word at a time, i.e., given a question, it predicts what is the most likely first word in the answer, then the second word, then third and so on, until the answer is complete.

Let's take an example and go step by step

Step 6: Generating the Answer

🔹 What Happens?

- The Transformer **predicts the next token** in the response based on the input.
- The softmax layer outputs **probabilities for possible words**.

📌 Example:

For **"Who wrote Harry Potter?"**, the model predicts:

Token	Probability
J.K.	0.85
Rowling	0.95
author	0.20
British	0.10

- The highest probability token **"J.K."** is chosen.
- The process repeats to predict **"Rowling"**, forming the answer **"J.K. Rowling"**.

Step 7: Output Formatting & Post-Processing

🔹 What Happens?

- The generated answer **"J.K. Rowling"** is formatted properly.
- Additional **language refinements** might be applied (e.g., punctuation).
- If the model is used in a chatbot, it might wrap the response in a complete sentence: **"J.K. Rowling wrote Harry Potter."**

How are transformer models trained?

Before a Transformer like GPT or BERT can generate or predict text, it must be trained on vast amounts of data

Training involves:

- Feeding massive datasets (e.g., books, websites, articles).
- Adjusting model weights using backpropagation and gradient descent.
- Minimizing the error (loss function) over many iterations.

📌 **Key Concepts:**

1. **Dataset Preparation** – Text is collected, tokenized, and converted into numerical representations.
2. **Forward Pass** – The model predicts the next word (or fills in missing words).
3. **Loss Calculation** – The model's output is compared to the correct answer, and an error (loss) is computed.
4. **Backpropagation & Weight Updates** – The model adjusts its internal parameters (weights) to improve future predictions.
5. **Repeat for Many Iterations** – The model learns patterns in language by training over millions/billions of text samples.

How are transformer models trained?

🛠️ Step 1: Preprocessing the Training Data

🔹 **What Happens?**

- Text data is **collected** from diverse sources (books, Wikipedia, web).
- The data is **cleaned** (removing unnecessary symbols, normalizing case).
- Text is **tokenized** into subwords using **Byte Pair Encoding (BPE)** or **WordPiece**.
- Each token is converted into a **numerical embedding**.

📌 **Example:** Sentence → **"The cat sat on the mat."**

Tokenized → `["The", "cat", "sat", "on", "the", "mat", "."]`

Converted to IDs → `[101, 2001, 3003, 2045, 101, 4098, 102]`

👉 **Why?**

- Machine learning models can only process numbers, so text must be converted into numerical form.

🛠️ Step 2: Forward Pass - Making Predictions

🔹 **What Happens?**

- The token embeddings are passed through the **Transformer network**.
- Self-attention mechanisms allow the model to understand word relationships.
- The model **predicts a word/token** based on its learned representations.

📌 **Example (GPT Training – Next-Word Prediction):** Training sentence: **"The cat sat on the" → [Target: "mat"]**

Model Prediction:

Token	Probability
mat	0.85
chair	0.10
roof	0.05

- The model **incorrectly predicts "chair"** instead of "mat" → leads to **loss calculation**.

How are transformer models trained?

🛠️ Step 3: Loss Calculation

🔹 **What Happens?**

- The predicted output is compared with the **actual target word**.
- A loss function (e.g., **Cross-Entropy Loss**) calculates how far the prediction is from the correct answer.
- **Higher loss** means worse predictions.

📌 **Example:**

If the correct word is **"mat"** but the model predicts **"chair"**, the loss is computed as:

$$Loss = -\sum(y_i \log(\hat{y}_i))$$

where:

- y_i is the actual word ("mat").
- $\hat{y}_i$ is the predicted probability of "mat" (e.g., 0.10 instead of 0.85).

👉 **Why?**

- The goal is to **reduce this loss** so that the model becomes more accurate over time.

🛠️ Step 4: Backpropagation & Weight Updates

🔹 **What Happens?**

- The model **adjusts its internal weights** to improve future predictions.
- Uses **Gradient Descent** + **Backpropagation** to update the Transformer's parameters.
- The updates propagate through **millions/billions of parameters**.

🔎 **How it works:**

1. **Compute Gradient:** Determine how much each weight contributed to the error.
2. **Update Weights:** Adjust weights in the opposite direction of the error.
3. **Repeat:** Over many iterations, the model learns better representations.

📌 **Example:**

- Initially, the model thinks **"chair"** is a good next word.
- After many training updates, the model learns **"mat"** is more probable.

How are transformer models trained?

🛠️ Step 5: Repeat for Millions of Examples

🔹 **What Happens?**

- The process is repeated over **massive datasets** for **millions of sentences**.
- Over time, the model learns: ✅ Grammar & syntax
  - ✅ Common sense reasoning
  - ✅ Long-range dependencies
  - ✅ World knowledge (if trained on large corpora)

📌 **Example – Before vs. After Training:**

Input	Model Prediction (Early Training)	Model Prediction (After Training)
"Paris is the capital of"	"Spain" (wrong ❌)	"France" (correct ✅)
"The sky is"	"green" (wrong ❌)	"blue" (correct ✅)

👉 **Why?**

- **Early training** is random.
- **After many updates**, the model generalizes and predicts accurately.

- ✅ Transformers **learn from data** using self-attention & backpropagation.
- ✅ Training involves **predicting words, calculating loss, updating weights** over millions of samples.
- ✅ After training, models are **fine-tuned** for specific tasks.
- ✅ Pretrained models like **GPT and BERT** power real-world AI applications.

Contents

Embedding is an important part of the process, and this has been around for a while

Before the introduction of Transformer models (2017), Natural Language Processing (NLP) relied on simpler techniques to represent text numerically. These techniques had limitations in understanding context and long-range dependencies.

1️⃣ TF-IDF (Term Frequency - Inverse Document Frequency)

🛠️ How It Works:

- **Counts how often words appear** in a document (**Term Frequency**) and balances that with how rare they are across all documents (**Inverse Document Frequency**).
- Used for **document retrieval & search engines** (e.g., Google Search before deep learning).

📌 Example:

Consider two documents:

- **Doc 1:** "AI is amazing and powerful."
- **Doc 2:** "AI is transforming the world."

Word	TF (Doc 1)	TF (Doc 2)	IDF Score	TF-IDF (Doc 1)	TF-IDF (Doc 2)
AI	1	1	0.5	0.5	0.5
amazing	1	0	1.5	1.5	0
world	0	1	1.5	0	1.5

2️⃣ Word2Vec (Word Embeddings)

🛠️ How It Works:

- Introduced in 2013, **Word2Vec** learns **dense vector representations** of words.
- Words with **similar meanings have similar vectors** in a high-dimensional space.
- Uses two methods: **CBOW (Continuous Bag of Words)** & **Skip-gram**.

📌 Example:

- Word vectors in a **300-dimensional space**:
  - **King → [0.2, -0.5, 0.7, ...]**
  - **Queen → [0.3, -0.6, 0.8, ...]**
- **Mathematical analogy**:
  King - Man + Woman ≈ Queen 🏰

Challenges with these techniques

- **TF-IDF & Word2Vec** capture individual word importance but fail in **understanding relationships, context, or long-term dependencies**.
- Transformers introduced **context-aware embeddings**, **self-attention**, and **long-range understanding**, revolutionizing NLP.

🚀 **Example:**
Unlike Word2Vec, Transformers **understand context**:

"Apple is a fruit." 🍎 → Recognized as a food item.
"Apple released a new iPhone." 📱 → Recognized as a tech company.

🔑 Key Takeaways

✅ **TF-IDF:** Simple frequency-based word importance (good for search engines).
✅ **Word2Vec:** Dense, context-free word embeddings (good for similarity tasks).
✅ **Transformers:** Contextual embeddings, self-attention, and **understanding word meaning dynamically**.

Exercise

1. Let's take a text dataset and embed it using TF-IDF and then make a model
2. Your turn:
    1. Using given dataset, convert text to vectors
    2. Train model on vectorized dataset
    3. Try different vectorizing techniques and evaluate performance

3. We have already seen how TF-IDF works, lets look at how a transformer model can do the embedding
4. Your turn:
    1. Try modelling on the same data as previous exercise but instead of TF-IDF, use a transformer model to generate embeddings
    2. Train model and evaluate performance

Contents

GPT models are great, but they are highly general & poor at specific questions

GPT models are trained on a lot of generic data. This means that it cannot perform well on specific tasks.
For example, what if I wanted the GPT model to tell me about internal processes at my company – can GPT do this?

There are a few ways to achieve this:

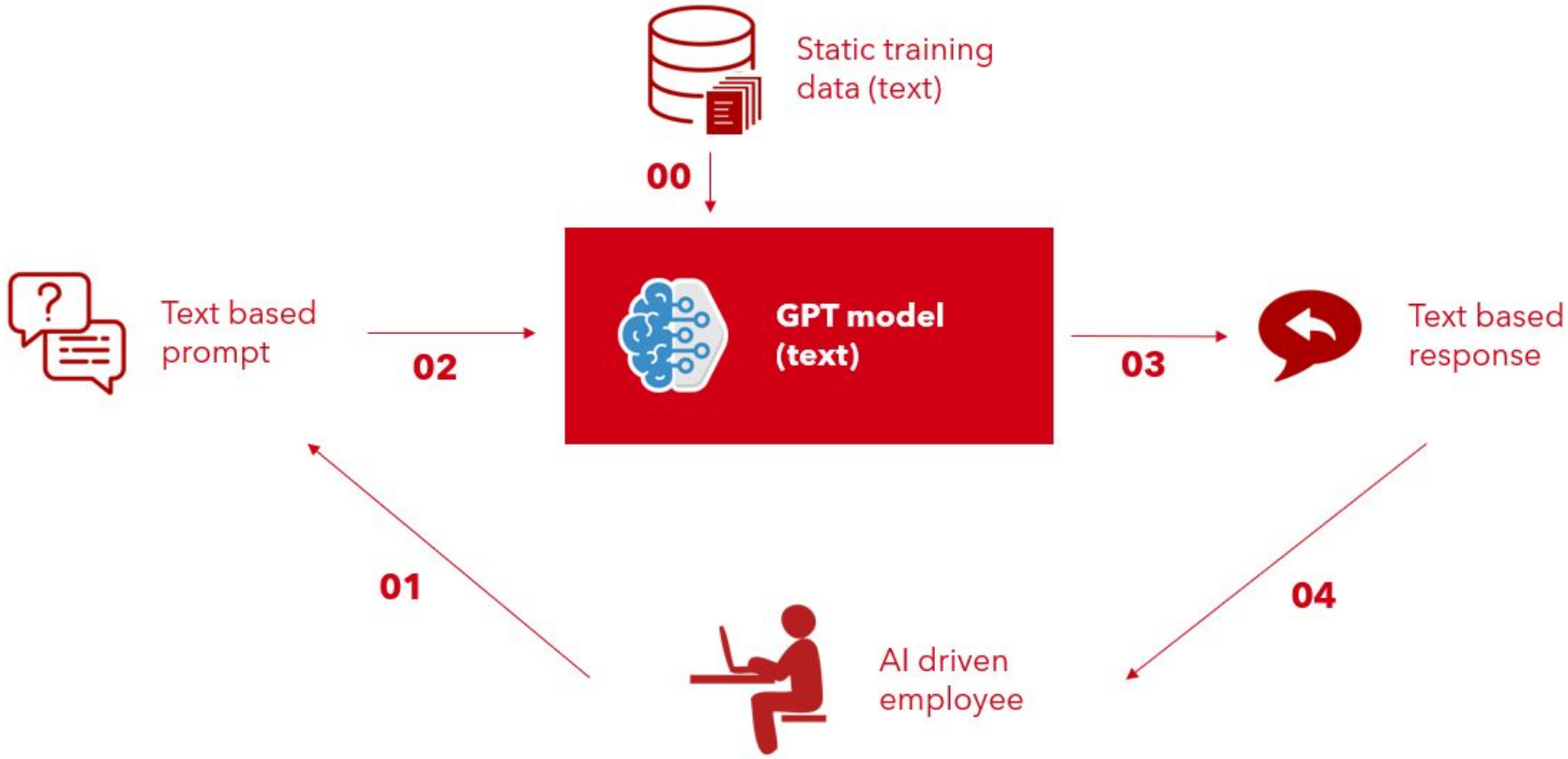
🔍 What is RAG?

- RAG is an **AI framework** that **combines text generation (GPT)** with **retrieval from external knowledge sources.**
- Unlike standard GPT models, which rely **only on pre-trained knowledge**, RAG can **fetch relevant documents from databases, APIs, or the internet** in real-time.
- This helps GPT models **stay up to date, reduce hallucinations, and provide factually accurate responses.**

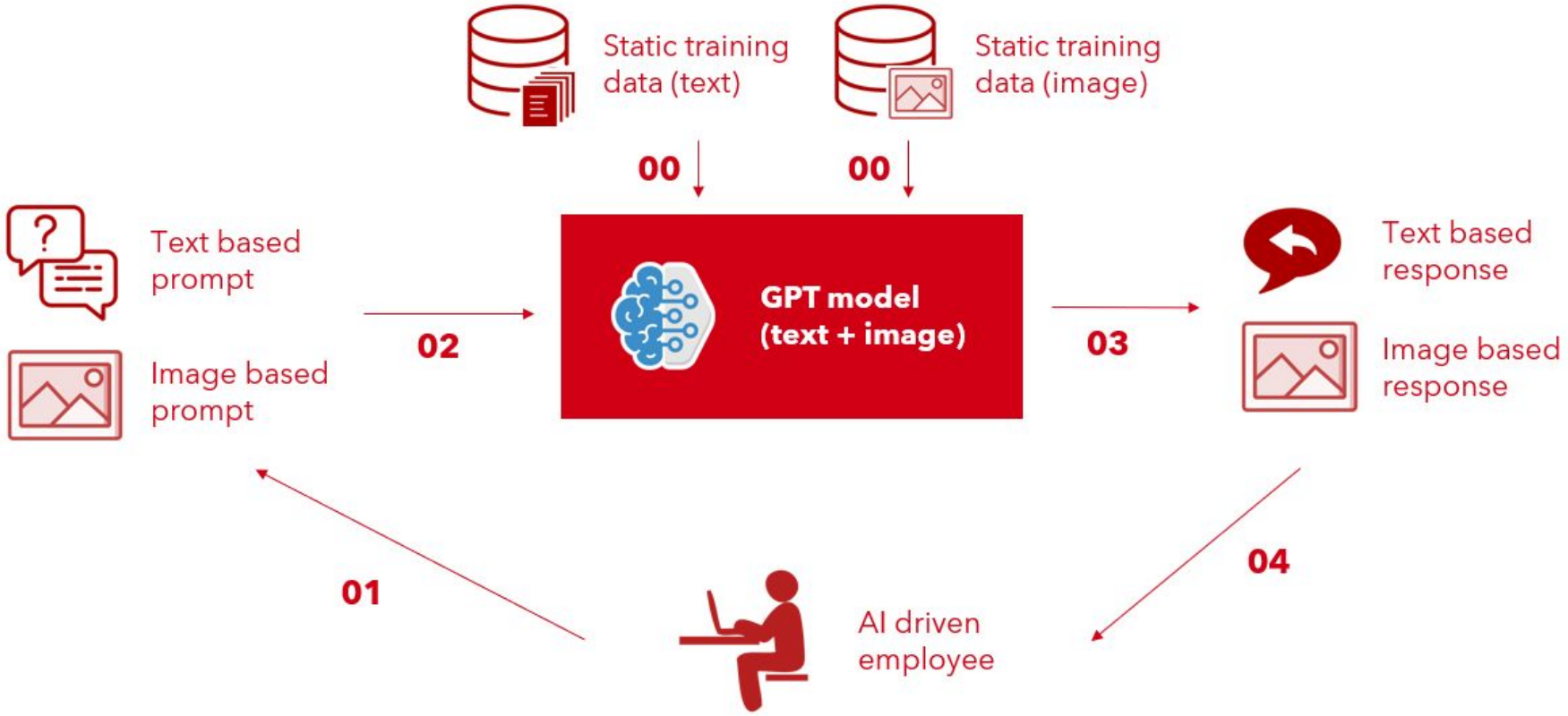
🔍 What is Fine-Tuning?

Fine-tuning is the process of **adapting a pre-trained Transformer model (like GPT-4) to a specific task or domain** by training it further on a smaller, specialized dataset. This allows the model to **refine its knowledge** beyond its general training.

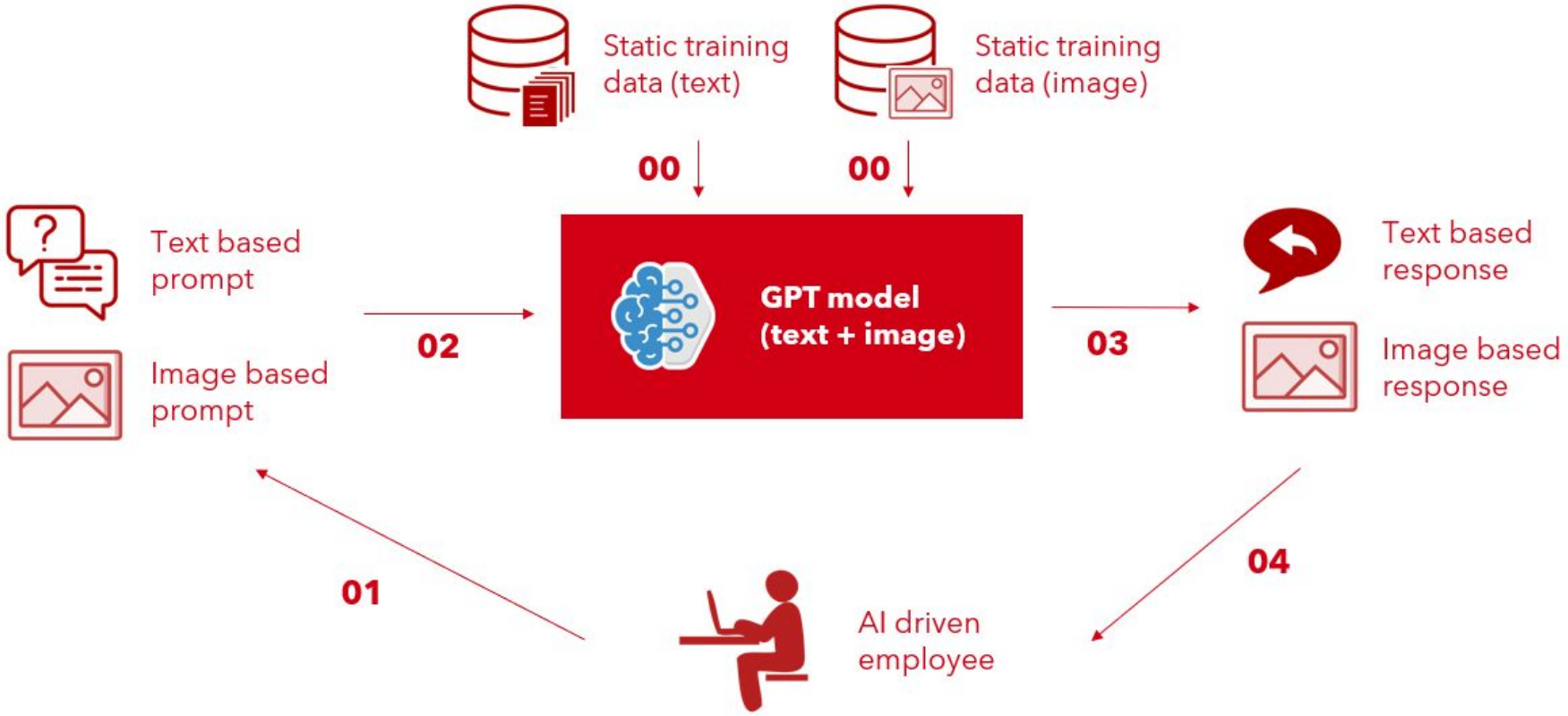
How RAG works



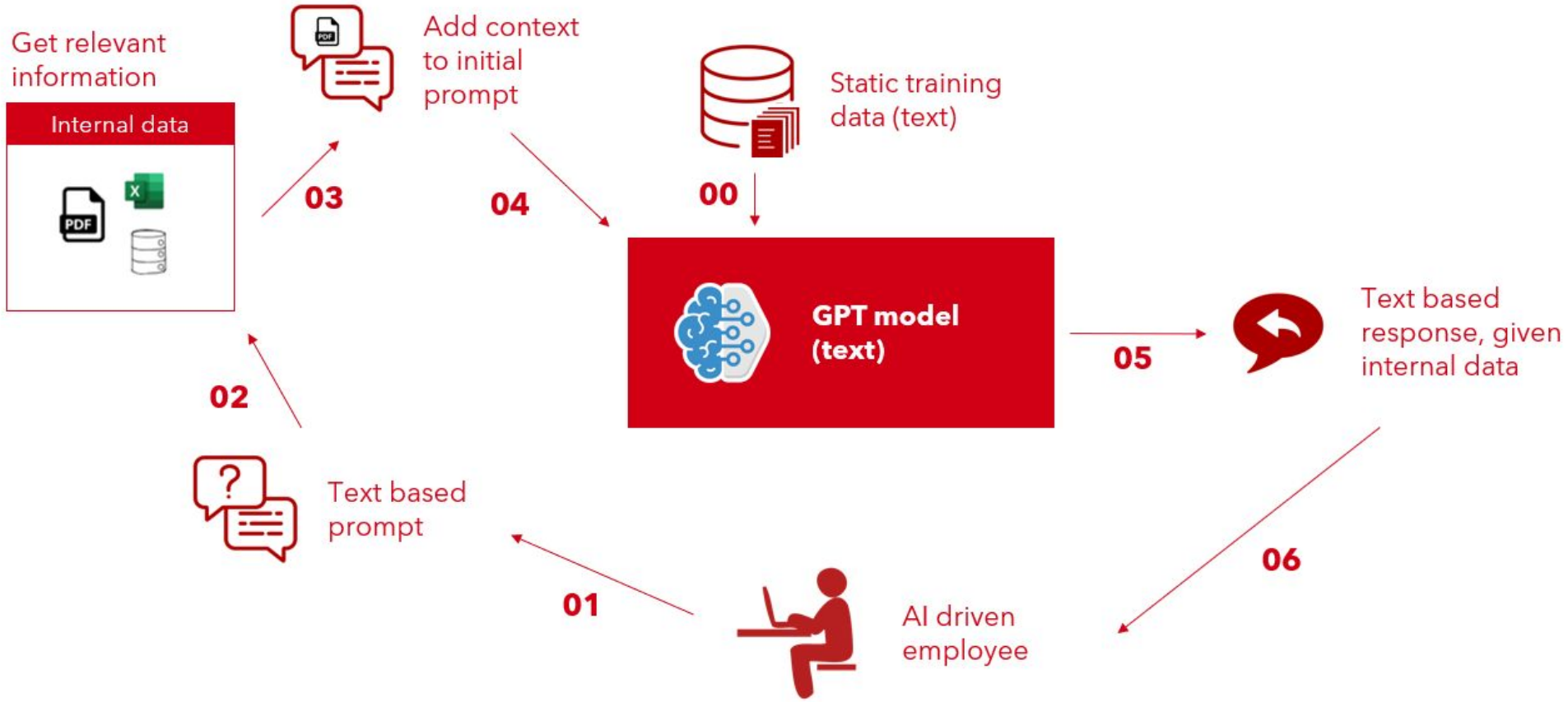
How RAG works



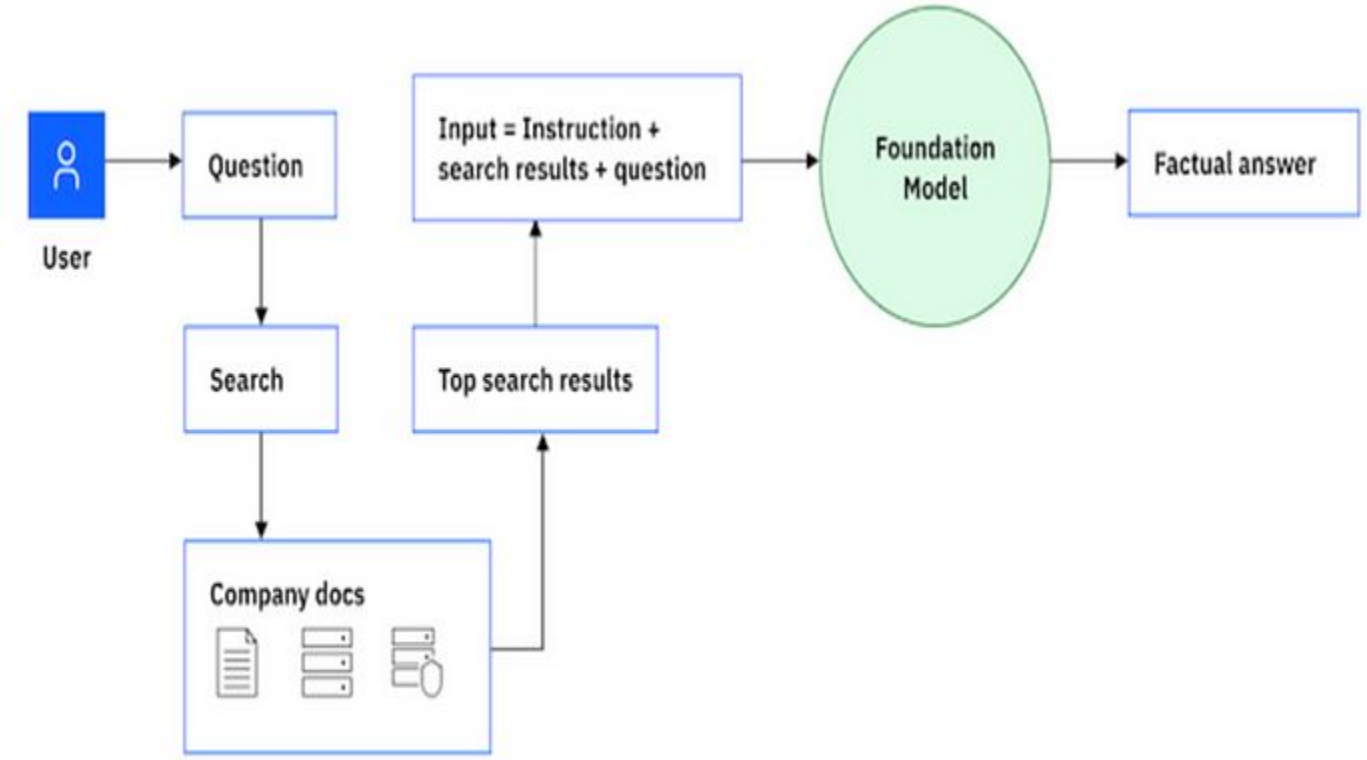
How RAG works



How RAG works



RAG application architecture



Source: IBM Developer. (n.d.). https://developer.ibm.com/articles/awb-retrieval-augmented-generation-with-langchain-and-elastic-db/

[Step-by-Step Guide to Building a Simple RAG Application Using LangChain and Streamlit | Medium]

Agents in GenAI

🔍 What are AI Agents?

AI Agents are **autonomous systems powered by Generative AI** that can **perceive, reason, and take actions** to achieve specific goals **without human intervention**. Unlike traditional AI models that generate static responses, AI Agents can:

✅ **Plan multi-step actions** based on goals.
✅ **Interact with external tools, APIs, and databases**.
✅ **Continuously learn and adapt** from feedback.

🔍 What Makes an AI Agent Different?

Unlike traditional Generative AI models that only generate text based on prompts, **AI agents are designed to:**

- ✅ **Set goals** based on user input or internal triggers.
- ✅ **Break down tasks into actionable steps.**
- ✅ **Retrieve and process information from external sources.**
- ✅ **Interact with APIs, tools, or other AI models to take action.**
- ✅ **Evaluate results and refine their approach** autonomously.

How does an AI agent work?

1 Perception – Understanding the Task

- The AI **receives a goal or prompt** from a user.
- It **analyzes the request** and decides what needs to be done.
- Example: A user asks, **"Plan my trip to Paris and book a hotel."**

2 Planning – Creating a Multi-Step Action Plan

- The agent **breaks the goal into smaller sub-tasks.**
- Example:
  - 📝 Find the best flight options.
  - 🏨 Compare hotel prices & locations.
  - 📅 Book a hotel that fits the user's budget.

3 Retrieval – Gathering Relevant Information

- The AI **searches online**, accesses APIs (Google Flights, Booking.com), or retrieves information from databases.
- If more details are needed, it **asks follow-up questions** (e.g., "What's your budget for hotels?").

4 Execution – Taking Action

- The AI interacts with external tools:
  - Calls an API to check flight availability ✈️.
  - Uses an automation tool (Zapier, LangChain) to book the hotel 🏨.
  - Updates the user with real-time options.

5 Reflection & Self-Correction – Adapting Based on Results

- The agent evaluates whether the task was **successfully completed.**
- If a step fails (e.g., hotel fully booked), the AI **adjusts its strategy** and tries another option.

6 Learning – Improving Over Time

- The AI stores past interactions, **learning user preferences** for future tasks.
- Example: Next time, it **automatically selects preferred airlines or hotel chains.**

Exercise

1. Let’s see a simple RAG application
2. Your turn:

Wrap up

1. Background & fundamentals of GenAI
    a. What is GenAI & why is it so popular?
    b. Attention & Transformer models – how do they work
    c. Types of GenAI models – images, text, etc.
2. Transformer vs Natural Language Processing (NLP
    1. Explaining basics of NLP
    2. Basic vectorizing – TF-IDF
    3. Exercise: Modelling on text-based data using TF-IDF
4. Retrieval Augmented Generation (RAG) & Agentic workflows
    a. What is RAG & how does it work?
    b. What are agents & how do they work?
    c. Exercise: building a basic RAG application