# Generative AI Pipeline

1. Data Acquisition

* Available Data(csv,text,pdf, docx,xlsx or others)
* Other Data (DB, internet, API, web scrapping)
* No Data (create your own data) 🡪 LLM to generate Data  
  Note: if you have less data then you perform data augmentation,  
  ex: Replace with synonyms (I am Data Scientist 🡪 I am AI engineer)
* **BIGRAM FLIP**I am Bappy 🡪 Bappy is my name
* **BACK TRANSLATE**
* **ADDITIONAL DATA/NOISE**I am a data Scientist. I love my job.

1. Data Preparation/ Preprocessing

* CLEANUP: HTML,EMOJI, SPELLING CORRECTION
* BASIC PRE-PROCESSING:  
  Tokenization (a) SENTENCE b) WORD
* OPTIONAL PREPROCESSING –( Stop word removal,  
   steaming(less used today), Lemmatization(more used)  
  **steaming 🡪 play, played and playing 🡪 root form Play**
* Lemmatization
* PUNCTUATION REMOVAL
* LOWER CASE
* LANGUAGE DETECTION

ADVANCE PRE\_PROCESSING

* Part of Speech Tagging
* Parsing
* Coreference resolution

1. Feature Engineering

TEXT VECTORIZATION ( TFIDF, Bag of words, word2vec, one hot encoding, transformers models)

1. Modeling

Choose between open source and paid one

1. Evaluation

Intrinsive Evaluation (GEN AI Engineer)

Extrinsic Evaluation – After doing the deployment

1. Deployment
2. Monitoring and model updating/retraining

## Text Representation

1. What is feature extraction from text/images? Edges patterns
2. Why we need it? Numbers or vectors
3. Why it is so difficult ? (Test data)
4. What is the core idea?
5. Some techniques?

Example audio

A diagram of a frequency response

AI-generated content may be incorrect.

Audio and videos can have fixed data as table but for text data is very variable (that is input size is variable).

Ways to solve it

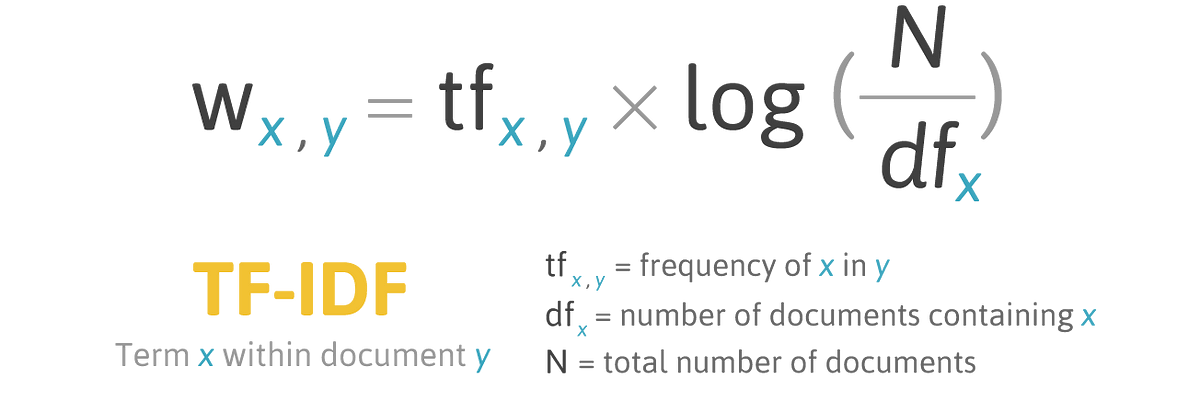
1. One- hot encoding
2. Bag of words

One-Hot Encoding 🡪 Drawbacks 🡪 Sparcity 🡪 No fixed size 🡪 out of vocabulary issue 🡪 Does not capture Semantic meaning 🡪 Don’t follow

Bag of Word  
Other methods TFIDF, Word2Vec, And Transformer models

N-Gram- It is a concept were you consider N-consecutive elements as 1 unit/token

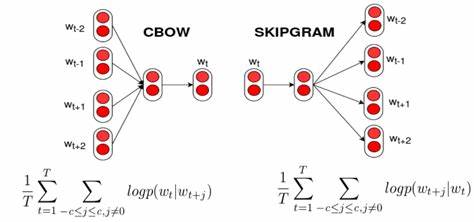
TF-IDF- works on one statistical Equation



TF-IDF🡪 Drawback 🡪 Cannot contain semantic information

**Word2Vec**

Word2Vec uses 2 kinds of architecture CBOW and SKIPGRAM



**ZERO-Shot Learning**

"The approach using a single command to get an LLM to take on a behavior is called Zero Shot Learning"

**Few Shot Learning**  
*In addition, to just providing an instruction it can be helpful to show the model what you want by adding examples this is called few shot learning because we showed the model a few examples*

**Like here is a prompt for translating from English to French**  
*First, we provide an instruction as shown below*  
***Convert the text from English to French***  
*Then we give some examples establishing the text pattern*

**Few Shot Learning**

***Convert the text from English to French*** ← *Instruction*

**Examples Establishing the Text Pattern:**

* **Peppermint:** menthe poivrée
* **Desert cactus:** Cactus du désert
* **Potato:** pomme de terre
* **Lipstick:** Rouge à lèvres
* **Orange Juice:** du jus d'orange
* **Sparkling water:** Eau gazeuse

**How ChatGPT was trained?**

Internally using a LLM which is gpt-3.5 or gpt-4  
It has trained on a large amount of data which is available all over the internet.

1. Generative pre-training
2. Supervised fine-tuning
3. Reinforcement learning

A screenshot of a computer

AI-generated content may be incorrect.

**Stage 1 - Generative Pre-Training**

* **Input:** Internet text data + Document text data
* **Description:**  
  A base GPT model is trained on a large corpus of text data from the internet using the Transformer architecture.

"This base model thingy (GPT) was trained on a bunch of stuff from the Internet for a whole bunch of different things by using the Transformer Architecture."

**Stage 2 - Supervised Fine Tuning (SFT)**

* **Description:**  
  Human AI trainers have conversations where they act as both the user and the assistant to fine-tune the model.

"Next, with the human AI trainers, you get to have conversations where they play both sides – you and an AI assistant."

**Stage 3 - Reinforcement Learning through Human Feedback (RLHF)**

* **Description:**  
  The model is optimized further by comparing responses and training it using a reward model.

"Next, let’s take the model to the next level by optimizing it even more with Reinforcement Learning by training it against a reward model."

A screenshot of a computer screen

AI-generated content may be incorrect.

**🔶 Title: Generative Pre-Training**

**🧠 Process Overview:**

* **Input Sources:**
  + 🌐 Internet text data
  + 📄 Documents text data
* **Model Architecture:**
  + 🔷 Transformer
* **Output:**
  + ➡️ **Base GPT Model**

**🔍 Key Insights:**

* The Transformer is trained on massive text corpora to **create a base GPT model**.
* This base model captures language structure and general knowledge.

**🔧 What it can do (Reality):**

* ✅ Text Summary
* ✅ Sentiment Analysis
* ✅ Sentence Completion
* ✅ Translation

**🎯 What we want it to do (Expectation):**

* 🗨️ **Chat and conversation**

A diagram of a process flow

AI-generated content may be incorrect.

**🔶 Title: Supervised Fine-Tuning (SFT)**

**🧠 Process Overview:**

1. **Crafted Conversations**
   * A **human agent** (acting like an ideal AI assistant) creates **ideal responses** to prompts (requests).
   * Example:
     + Request 1 → Ideal Response 1
     + Request 2 → Ideal Response 2
     + Request 3 → Ideal Response 3
2. **SFT Training Data Corpus**
   * These prompt-response pairs are collected as training data.
   * The conversation history and ideal next responses are visually marked.
3. **Training**
   * The **Base GPT model** is fine-tuned using this corpus.
   * The **Stochastic Gradient Descent (SGD)** algorithm is used for optimization.
   * Output: **SFT ChatGPT Model**

Human Agent → Prompt-Response Pairs → Base GPT Model → SGD → Fine-Tuned (SFT) ChatGPT Model

This stage helps the model learn **how to respond more helpfully and safely**, forming the foundation for human-like dialogue.

A diagram of a project

AI-generated content may be incorrect.

**🔶 Title: Reinforcement Learning through Human Feedback (RLHF)**

**🧠 Process Overview:**

1. **Start with the SFT-trained ChatGPT Model**
   * It generates multiple **alternate responses (A–D)** to a given **user request**.
2. **Human Agent Evaluation**
   * Human evaluators **rank the responses** based on quality (e.g., helpfulness, correctness, harmlessness).
   * Example Ranking:
     + Alt Response B > A > D > C
3. **Reward Model Training**
   * These rankings train a **Reward Model** that learns to assign scores to responses:
     + Score\_A, Score\_B, Score\_C, etc.
   * Input: Conversation history + response
4. **Policy Optimization**
   * This reward model is used to fine-tune the model using **Reinforcement Learning**, guiding it to prefer higher-ranked (better) responses.

A diagram of a diagram

AI-generated content may be incorrect.

# HUGGINGFACE (Transformers)

* 1. Pipeline
  2. NLP tasks
  3. Tokenization
  4. Dataset and spaces
  5. FineTuning LLM with huggingFace
  6. Project Implementation

2 procedure explained  
1) Taking pre-trained model from hugging face.  
2) Fine- tune model (banking data or other kind of specialized data)

**ChatCompletion API and Completion API**

* **Completion API**: Hooks you up with text completions from a single prompt.
* **Chat Completion API**: Nails it in the chat game, keeping the conversational flow intact

**🧠 Key Differences**

* **Completion API**:

*Hooks you up with text completions from a single prompt.*

* + Designed for **one-shot** or **single-turn** tasks.
  + Input: a block of text.
  + Output: continuation of that text.
  + Example use: Autocomplete, summarization, code generation.
* **ChatCompletion API**:

*Nails it in the chat game, keeping the conversational flow intact.*

* + Structured for **multi-turn** interactions (like a chatbot).
  + Uses a list of messages with roles: user, assistant, and system.
  + Maintains **conversational context** across multiple inputs.

**Function Calling in OpenAI**

**JSON**

Model

OpenAI   
API

JSON

Function Calling

**OPEN AI supports third party software**

**PROMPT Engineering**

Best Practices to design a prompt

* Clear instruction
* Adopt a persona
* Specify the format
* Avoid leading the answer
* Limit the scope

Types of Prompting

* Zero-shot prompting
* Few-shot prompting

# Vector Database

**Unstructured data as Text**

**Embedding model**

| **Feature** | **King** | **Queen** | **Man** | **Woman** | **Monkey** |
| --- | --- | --- | --- | --- | --- |
| Gender | 1 | 0 | 1 | 0 | 1 |
| Wealth | 1 | 1 | 0.5 | 0.3 | 0 |
| Power | 1 | 0.7 | 0.5 | 0.2 | 0 |
| Weight | 0.8 | 0.5 | 0.7 | 0.5 | 0.3 |
| Speed | 1 | 1 | 1 | 1 | 0 |

king ⇒ [1, 1, 1, 0.8, 1]

queen ⇒ [0, 1, 0.7, 0.5, 1]

man ⇒ [1, 0.5, 0.7, 0.7, 1]

→ OpenAI Embedding → GPT model

→ Hugging Face Embedding → Open source LLM

→ Llama 2 Embedding 🡪 Meta

→ Google PaLM Embedding 🡪 Google

A diagram of a diagram

AI-generated content may be incorrect.

In the context of **vector embeddings**, a **vector index** is a data structure that enables efficient storage, retrieval, and similarity search of high-dimensional embedding vectors.

**Explanation based on the image:**

1. **Input**:
   * You start with **unstructured data** (e.g., text, images, audio).
2. **Embedding Generation**:
   * This data is passed through a model that converts it into **embedding vectors**—numerical representations in high-dimensional space.
   * For example:  
     "document A" → [0.4, 0.7, 0.9, 0.1, 0.5, ...]
3. **Vector Index**:
   * These embedding vectors are stored in a **vector index**, which assigns a unique identifier to each vector and stores them for **fast similarity search**.
   * This index enables you to:
     + **Quickly find nearest neighbors** (e.g., using cosine similarity or Euclidean distance).
     + **Perform semantic search**, where similar meaning returns similar vectors.
4. **Use Case**:
   * When you query something new, you embed the query and then use the vector index to find the most similar stored embeddings (i.e., similar documents, images, etc.).

**Key Benefits of Vector Indexes:**

* **Speed**: Handles large-scale search efficiently.
* **Scalability**: Can manage millions of embeddings.
* **Versatility**: Used in search engines, recommendation systems, and chat-based retrieval (RAG pipelines).

**Tools for Vector Indexing:**

* **FAISS** (Facebook AI Similarity Search)
* **Pinecone**
* **Weaviate**
* **Milvus**
* **Annoy** (Approximate Nearest Neighbors)

**Some Very Popular & Powerful Open Source LLMs**

* Meta Llama 2
* Falcon

Github for open source llms - https://github.com/eugeneyan/open-llms

A diagram of data processing

AI-generated content may be incorrect.

The image lists **use cases of Vector Databases (Vector DB)** as follows:

1. **Long-Term memory for LLMs**
   * Store and retrieve contextual knowledge to extend language model capabilities.
2. **Semantic Search**
   * Enables search **based on meaning**, not just exact keyword matches.
3. **Similarity Search**
   * Works across various modalities: **Text, Images, Videos, Audios** to find related items.
4. **Recommendation engine as well**
   * Powers personalized suggestions by comparing user/item embeddings.