Generative AI

Introduction to Generative AI

What Is Generative AI?

- *“Generative AI refers to models that can create new data similar to the data they were trained on. Unlike traditional models that predict a label or a score, generative models output entire structures — a sentence, an image, a melody, or even a line of code.”*

Examples include:

- Generating realistic human language (ChatGPT, Claude)
- Creating images from text (DALL·E, Midjourney, Stable Diffusion)
- Music composition (Jukebox)
- Code generation (Codex, GitHub Copilot)

How Is It Different from Traditional AI?

Traditional AI	Generative AI
Classifies or detects	Creates new content
E.g., spam detection	E.g., write a new email

"Traditional AI is like a critic — it can tell you if something is good or bad. Generative AI is like a writer — it produces the content itself.

The Core Building Blocks

"Most generative models use these core techniques:"

- Transformers: The architecture powering most large-scale models (GPT, BERT, T5)
- Attention Mechanisms: Focus on important parts of input
- Language Models (LLMs): Trained on huge text datasets to predict the next word
- Latent Variables (in VAEs, GANs): Learn a compressed, abstract space for generation

Two Major Model Types

- Autoregressive Models (e.g., GPT):
  Generate one token at a time from left to right.
- Diffusion Models (e.g., Stable Diffusion):
  Start with noise and gradually “denoise” it into something meaningful.

Real-World Applications

“Generative AI is not theoretical — it’s powering real-world tools.”

- Content creation: Blogs, emails, ad copy
- Design: Images, brand assets
- Productivity: Summarizing documents, coding assistants
- Simulation: Synthetic data generation
- Personalization: Custom chatbots, dynamic learning experiences

Risks and Limitations

“With great power comes real challenges.”

- Hallucination: Making things up
- Bias: Learned from biased data
- Ethics: Deepfakes, misinformation
- IP and copyright: Legal questions around generated content

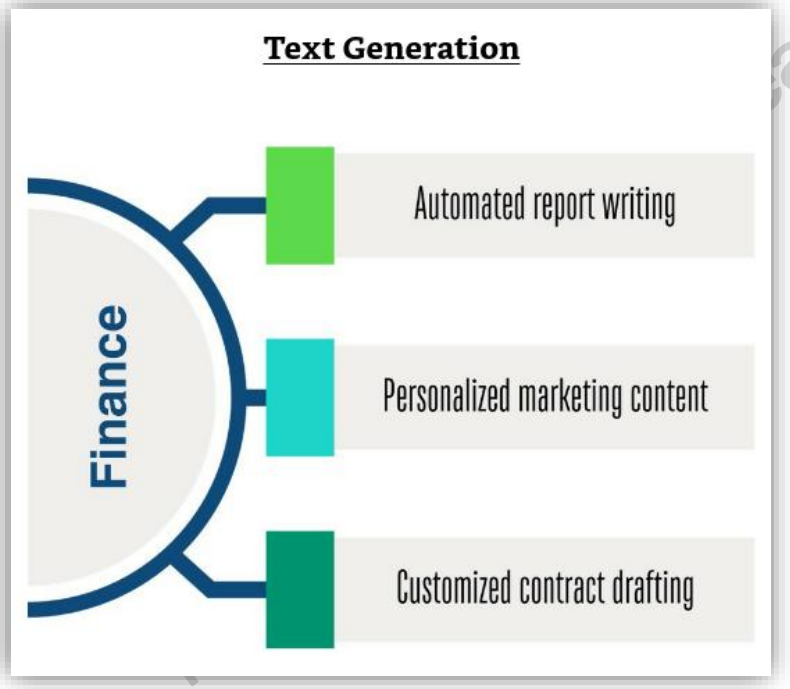
Responsible Use and Future Trends

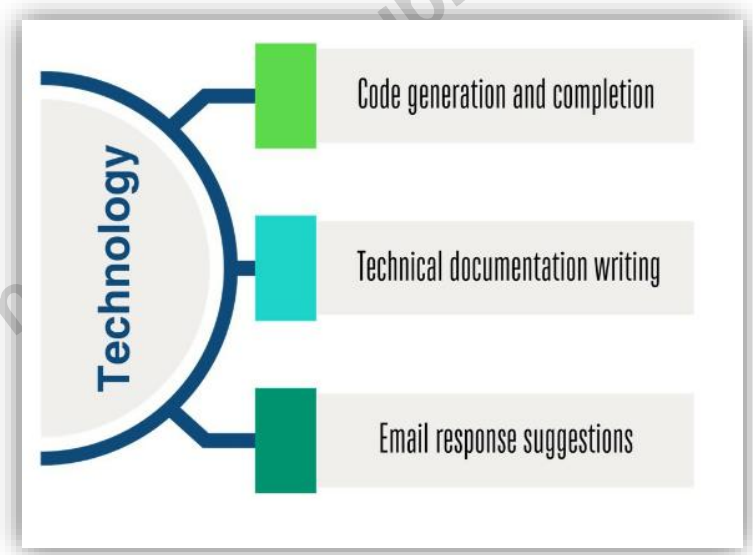
"The future of Generative AI is not just about capability — it's about control, transparency, and human alignment."

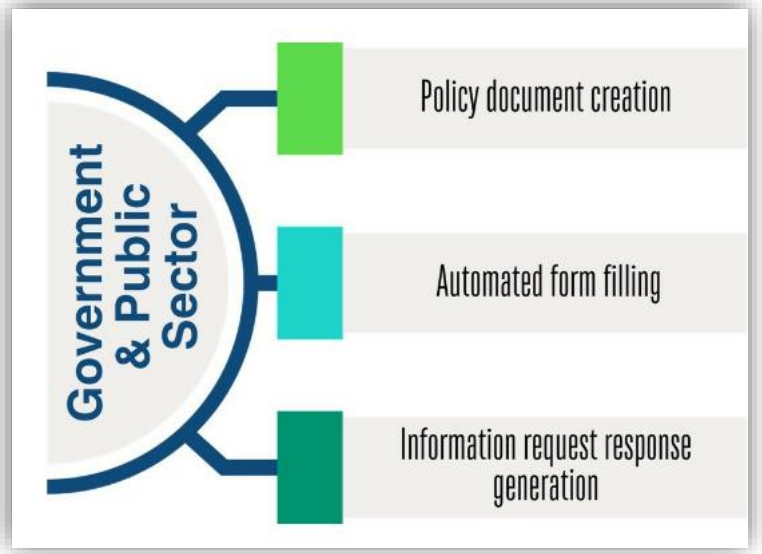
Trends:

- Smaller, domain-specific models
- Retrieval-Augmented Generation (RAG)
- Multi-modal generation (text + image + audio)
- Open-source models and fine-tuning

Generative AI Ideation Canvas

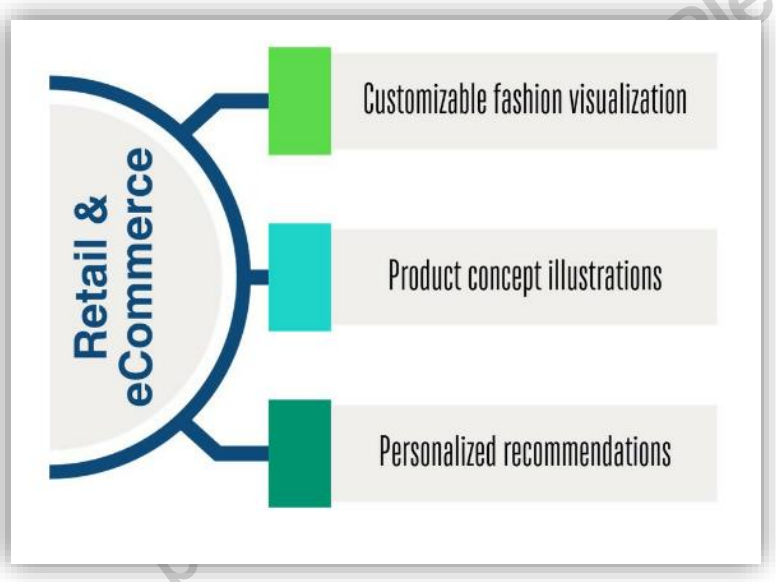


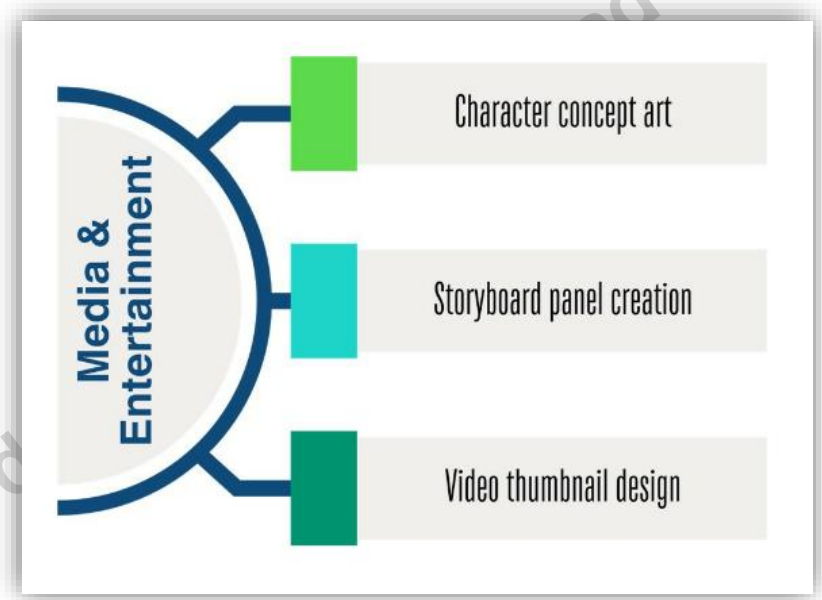


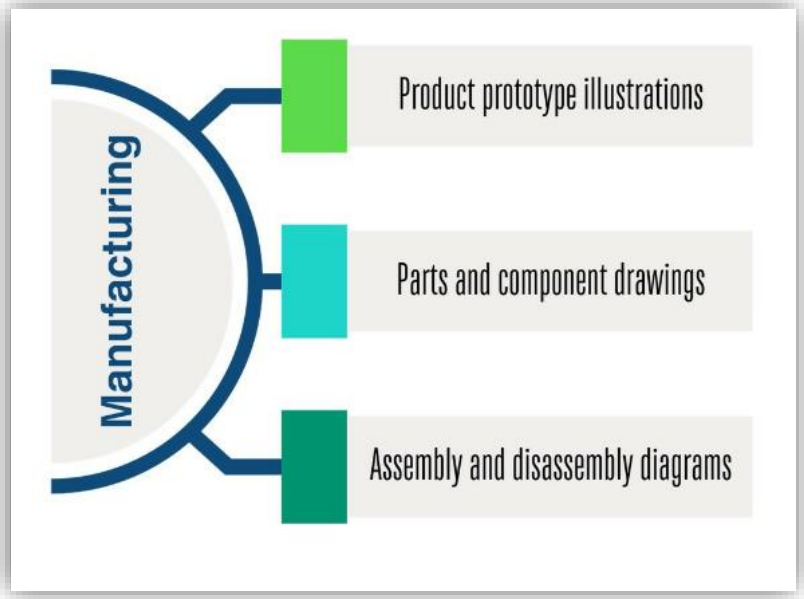


Generative AI Ideation Canvas

Image Generation







Wrap-up

- “Generative AI is not just a tool — it’s a collaborator.
  It extends human creativity, speeds up workflows, and changes the way we think about intelligence.”

Large Language Models (LLM)

INTRODUCTION TO LLMS

Large language models have rapidly risen to prominence as one of the most promising and transformative technologies in artificial intelligence.

OpenAI's GPT

The best LLM overall

Meta LLama 3

Best value LLM

GPT-4o

Best multimodal LLM

GitHub Copilot

Best LLM for coding

Qwen

Best LLM for chatbots

Google Gemini

Best LLM for translation

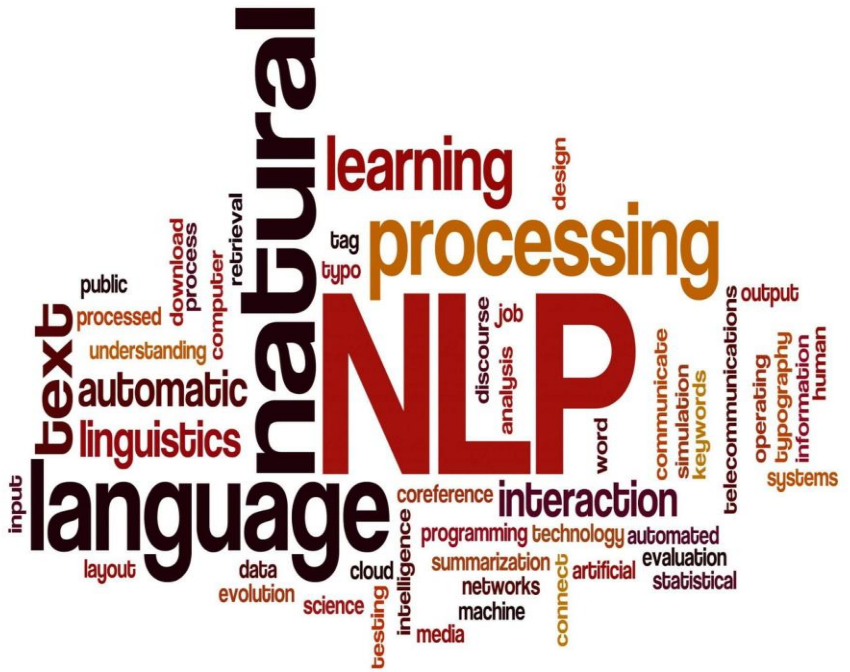
Claude 3

Best for business

The Quest for Language AI

Long before the era of deep learning, researchers were interested in statistical language modeling—developing mathematical models that could predict likely sequences of words based on patterns in text.

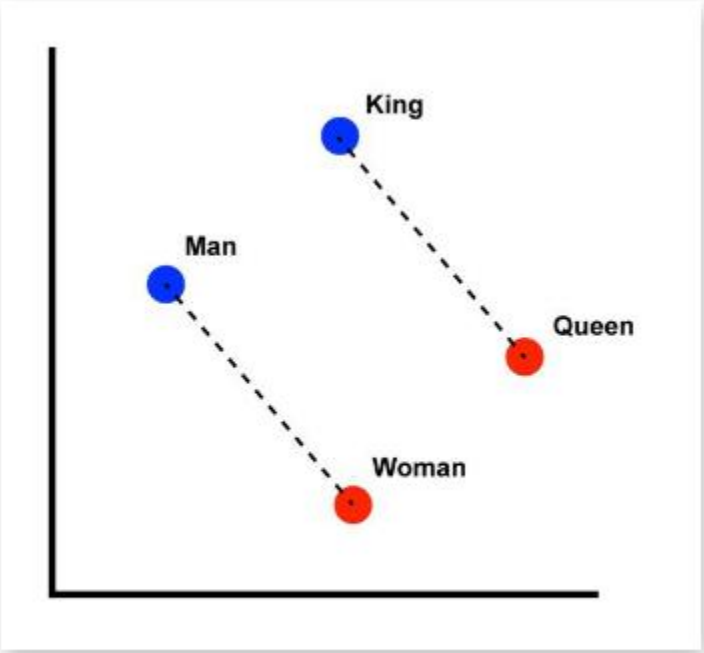
Though limited to local statistical patterns, these models were successfully used in applications like spell check and word prediction.



The Quest for Language AI

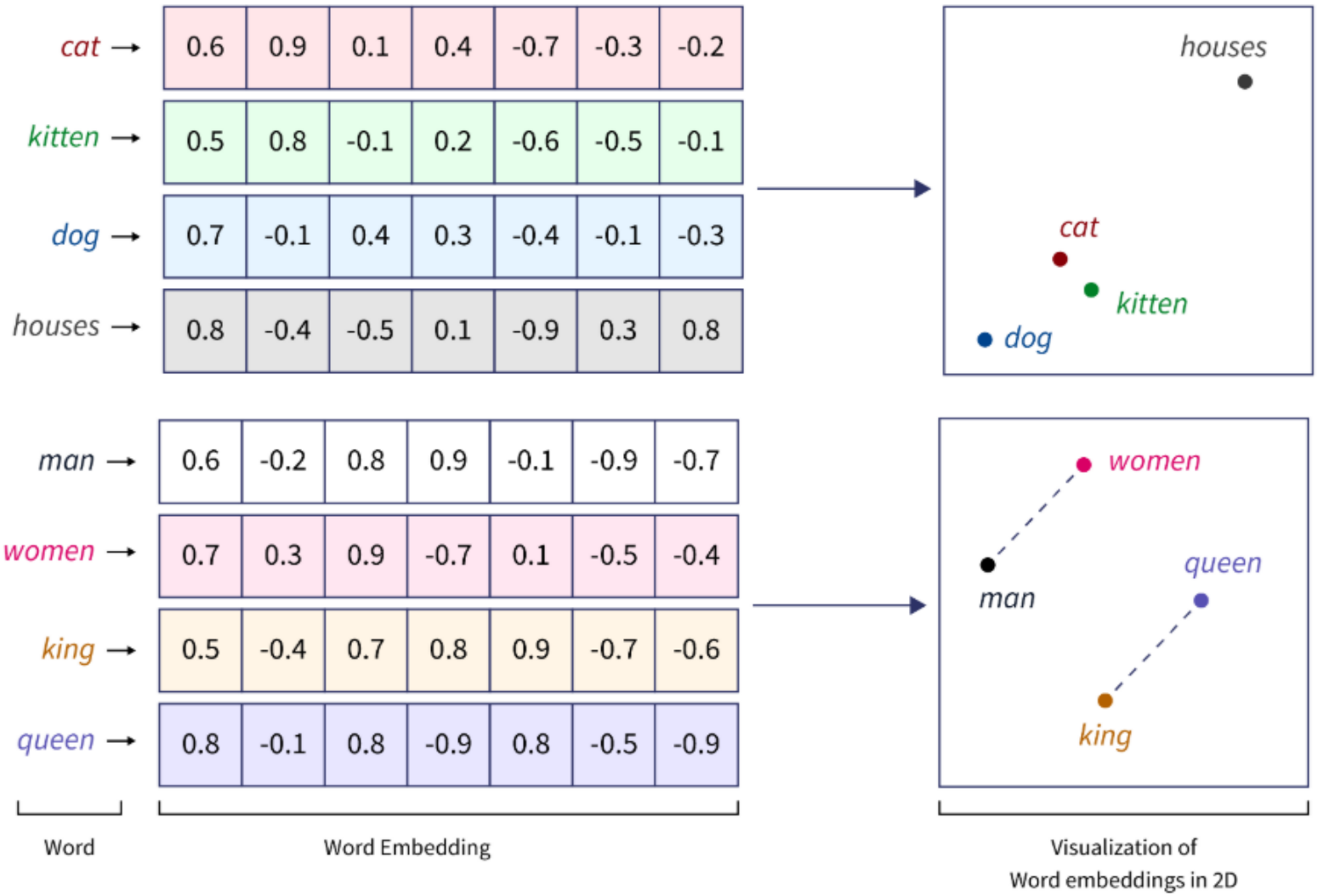
In the late 2000s, a major shift happened in the way we understood and processed language with the help of computers. One of the standout models from this era was called Word2Vec .

Think of Word2Vec as a method that teaches computers to understand the relationship and context between words like how humans do. Instead of just reading words as isolated bits of information, Word2Vec processes large amounts of text to determine how words relate to one another. This process results in creating 'embeddings'.



Word vector illustration.

Word Embeddings



The Rise of Transformers

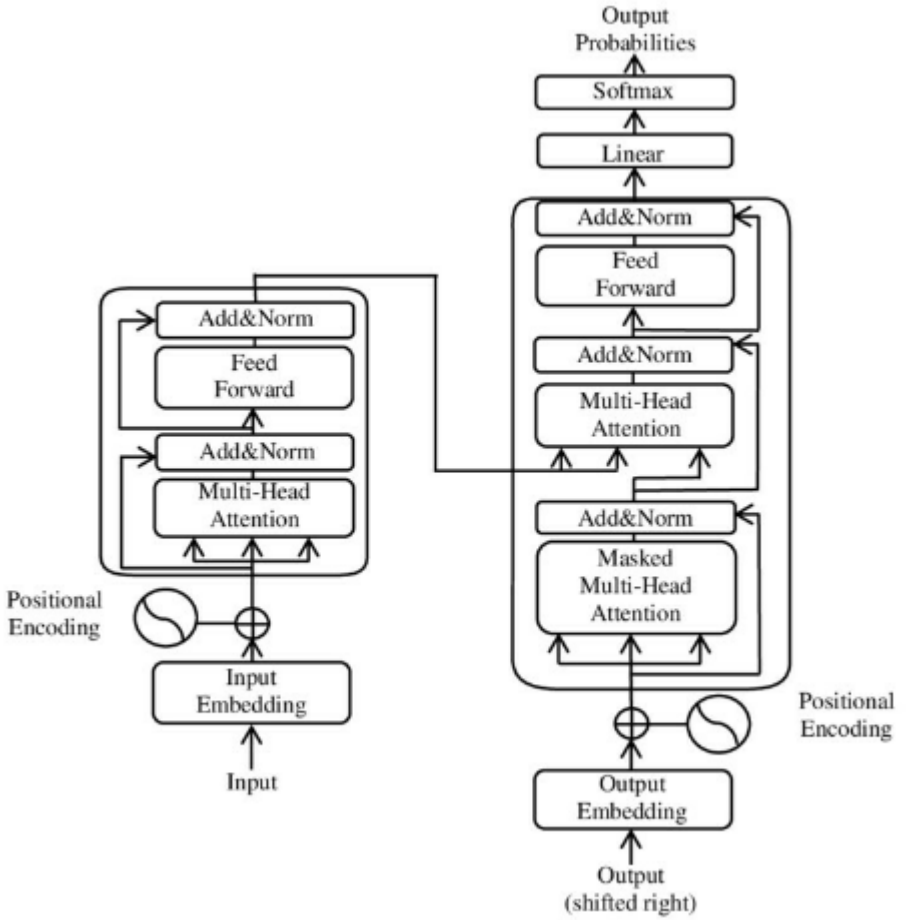
The world of language processing witnessed a paradigm shift in 2017, a revolutionary change brought by introducing the "transformer" architecture.

This advancement has, over recent years, underpinned some of the most significant strides in artificial intelligence.

At the heart of the transformer architecture is a mechanism known as "self-attention".

[1706.03762] Attention Is All You Need

https://arxiv.org/abs/1706.03762



Transformer Model Architecture

LLMs

GPT-1, released in 2017, was the first transformer-based LLM. It had 117 million parameters—think of parameters like the knobs and levers that control how the machine processes language data.

Though tiny compared to modern LLM sizes, GPT-1's 117 million parameters learned enough complex language relationships to outperform substantially previous statistical models at tasks like answering questions and drawing inferences from passages.

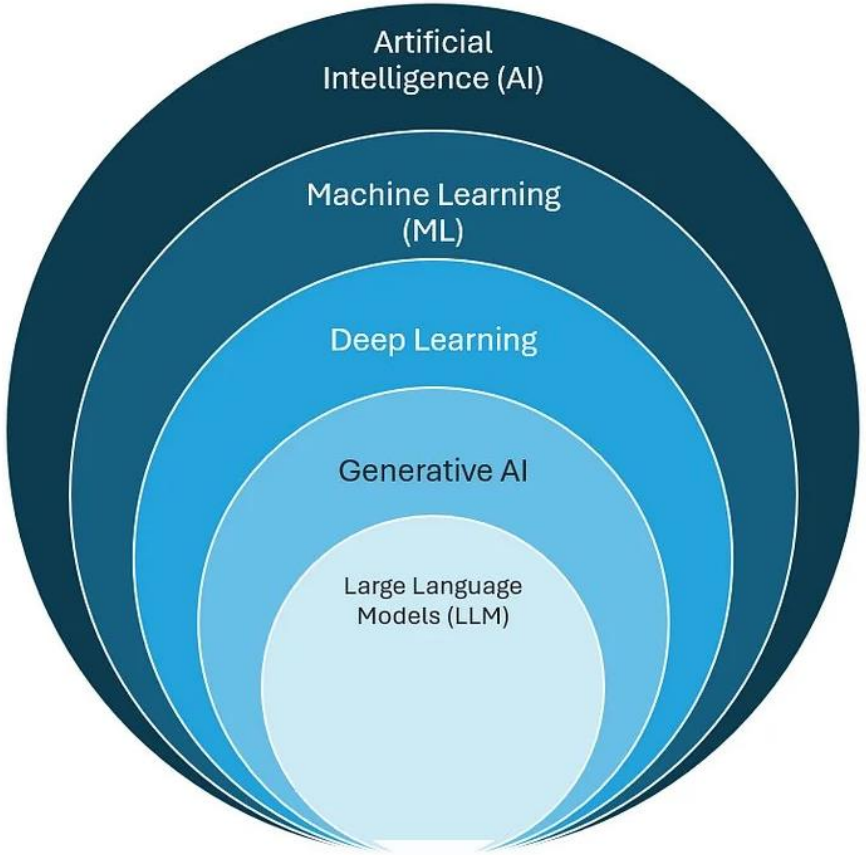
- In 2018, Google introduced BERT (Bidirectional Encoder Representations from Transformers with 110 million parameters
- GPT-2, released by OpenAI in 2019, benefited from 10x more parameters and 10x more data than GPT-1.
- GPT-3 took another leap in 2020, scaled up to 175 billion parameters trained on over a trillion words.
- GPT-4 continues the trend of exponential growth, likely employing a mixture of expert models, together reaching the trillion-parameter scale and multi-trillion word training datasets.
- Beyond OpenAI, other organizations like Google, Meta, Anthropic, Baidu, and more have trained LLMs at scale.

LLMs

LLMs fall under the subset of AI called generative AI.

LLMs are a leading example of Generative AI because they can generate original, human-like text after training on large text corpora. Other types of generative AI include systems that generate images, videos, music, 3D shapes, and more, based on analyzing datasets of visual content.

Generative AI, fueled by advances in LLMs, has rapidly risen to prominence given its ability to automate content creation in affordable and customized ways.



Specialized LLMs

BloombergGPT , a large language model specially trained on financial data to understand nuanced business and finance language.

With 50 billion parameters trained on 360 billion tokens of financial text (comparable to number of words) and 345 billion tokens of general text, BloombergGPT achieves state-of-the-art results on financial NLP tasks like question answering and named entity recognition.

Specialized LLMs

Anthropic's Constitutional AI model incorporates a technique called Constitutional AI to improve capabilities like honesty, harmless, helpfulness, and avoiding harmful stereotypes. During training, the model is recursively prompted to edit its own responses until they demonstrably uphold principles in Anthropic's Bill of Rights for AI—seeking truth, upholding dignity, and promoting empathy.

Open or Closed and Why?

Meta released its open source large language model LLaMA in 2023, with 65 billion parameters trained on massive text and code datasets.

By making Llama available to everyone, Meta is encouraging researchers and developers to experiment with the model and find new and creative ways to use it.

OpenAI has not released their popular GPT-3 model as open source, instead providing API access to the model.

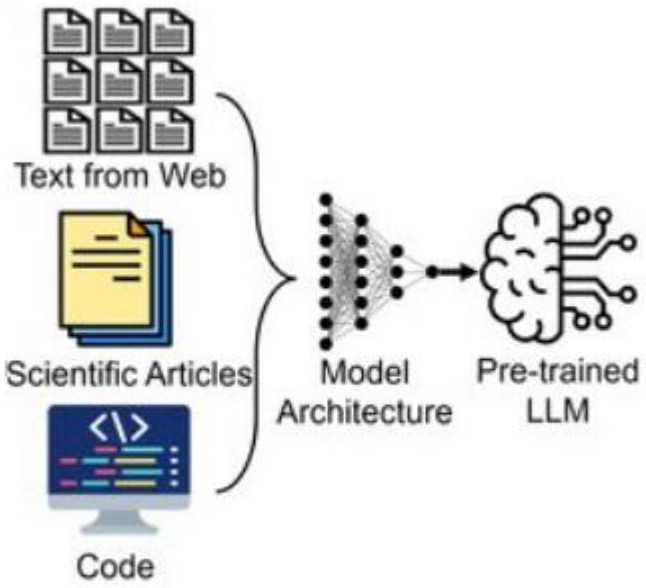
OpenAI may be hesitant to provide open access to their most advanced models and lose their competitive edge.

Key Stages of Development

Large language models go through several key stages of development to learn language skills.

The first stage is **pre-training**. In pre-training, the LLM trains on massive datasets of unlabeled text from sources like websites and books.

This stage teaches the LLM general linguistic knowledge and representations of language structure and use.

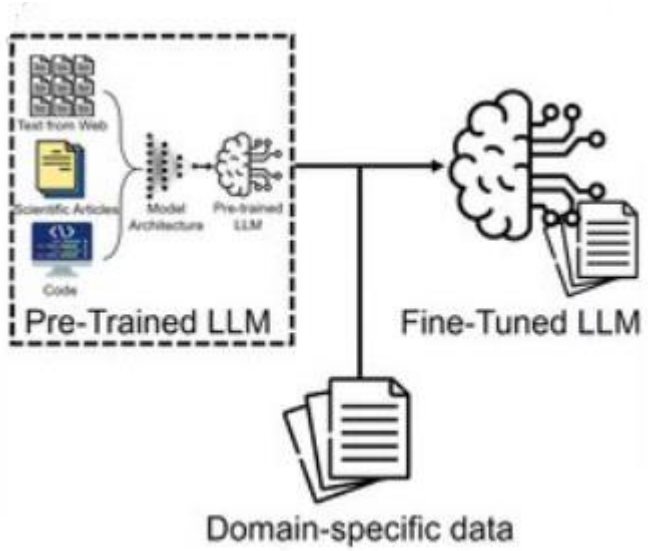


Key Stages of Development

The second stage is **fine-tuning**.

In fine-tuning, the model trains on human-labeled examples for specialized tasks.

This stage tunes the model to excel at specific applications versus just general language modeling.

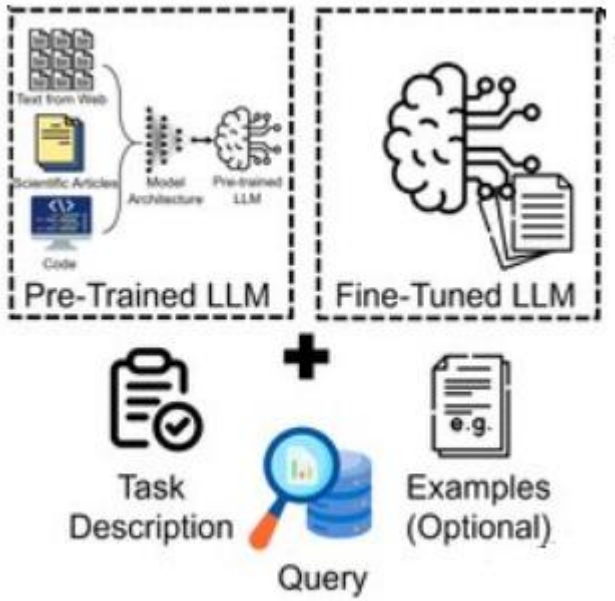


Key Stages of Development

The third stage for some models is **reinforcement learning from human feedback**.

Here, the LLM interacts with real users

and gets feedback on its responses. It then iteratively improves based on that feedback.



Training Challenges

- Training is highly computationally intensive
- Curating the massive datasets required is expensive
- Training stability and reproducibility is difficult
- Effectively testing and auditing model logic grows more challenging as scale increases.

Build or Buy?

A base LLM refers to a pre-trained, publicly available model like GPT-3 or LLaMA 2 that provides general language capabilities.

Building on top of these base models can save significant time and resources compared to developing a new LLM from zero.

Building a custom LLM enables tight control over training data and objectives. But it demands extensive data curation, engineering, compute resources, significant budget, and time.

Benefits of Pretrained Model

Time and Resource Efficiency

Generalization

Flexibility

How To Leverage Pretrained Models

Define your use case

Model Selection

Compute Resources

Model Size

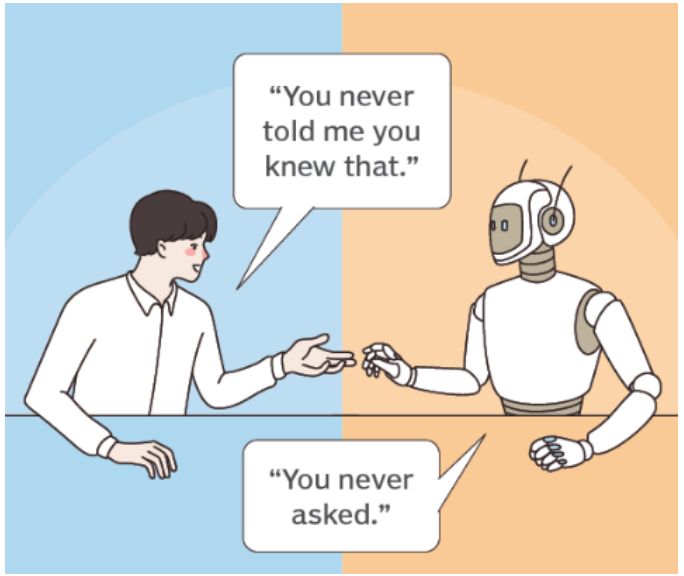
Data Bias

Evaluation

Prompt Engineering

What is Prompt Engineering?

- Think of it like programming — but using natural language instead of code."



Why Does This Matter for IT Professionals?

Prompt engineering helps you:

- Get consistent and accurate results
- Reduce hallucinations
- Align output with your task or domain
- Prototype faster without retraining models

Types of Prompts

- **Zero-shot:**
  "Just ask directly. Great for straightforward tasks."
  → Example: "Translate this to French: I am hungry."
- **One-shot:**
  "Include one example to show the model what to do."
- **Few-shot:**
  "Provide several examples to teach the pattern."
- **Instruction-based: "Give clear instructions like:**
  'Summarize the following text in bullet points.'

Prompt Templates

- Templates let you define a reusable structure that the model can fill in dynamically.

"Write a professional email to {client_name} apologizing for a delay in {issue_topic}. Offer a solution."

Best Practices for Effective Prompts

Best Practices for Effective Prompts

- ✅ Be specific: *Don't say 'Summarize this,*

  ' say 'Summarize in 3 bullet points.'
- ✅ Set the format: *Ask for JSON, Markdown, tables if needed.*
- ✅ Use delimiters: *Quote or wrap input in triple backticks.*
- ✅ Iterate: *Prompting is a design process — test and refine.*

Industry Use Cases

- Customer Support: Generate ticket replies or FAQs from templates.
- Software Development: Generate functions or explain code snippets.
- Data Analysis: Extract structured data from logs or free text.
- Compliance: Standardize policy interpretation across documents.

Prompting as a Skill

- “This is a fusion of technical and communication skills — that’s why IT professionals are uniquely suited to lead this space.”

Wrap up

- As IT professionals, learning to write better prompts gives you the power to build smarter apps, automate more tasks, and extract more value from LLMs — without needing to train your own models

Understanding Hallucination in LLMs

Hallucination in LLMs

Understanding When and Why AI Gets It Wrong

What is a Hallucination?

- *A hallucination is when a language model generates text that is factually incorrect, fabricated, or nonsensical — even though it sounds believable*

  - In AI, a hallucination = when a model generates false or made-up information
  - It sounds fluent and confident — but it's wrong

Example:

Q: Who won the 2023 Nobel Peace Prize?
A: Dwayne "The Rock" Johnson (Incorrect!)

Why Do LLMs Hallucinate?

- LLMs predict the next word, not the right answer
- They don’t “know” facts — they imitate patterns
- Their training data may be outdated or noisy
- They’re optimized for fluency, not truth

Types of Hallucination

1. Factual Errors – Incorrect real-world facts
   "Paris is the capital of Italy."
2. Fake Citations – Inventing sources or links
   "According to the book Quantum Cats by Dr. Who..."
3. Illogical Answers – Fluent but nonsensical logic
   "Two plus two equals five."
4. Overgeneralization – Vague, oversimplified answers
   "All dogs can fly."

Why Is This a Problem?

- Hallucination can be harmful in:
  - Education – Students might trust false facts
  - Healthcare – Misinformation could be dangerous
  - Legal – Wrong citations or interpretations
  - General use – Trust issues and spread of misinformation

- We risk spreading false info if we trust LLMs blindly

How to Prevent Hallucination

Technique	What it Does
RAG (Retrieval-Augmented Generation)	Uses external sources to ground facts
Fine-tuning on clean data	Reduces error-prone outputs
Verification or Fact Checking	Adds a second pass for accuracy
Prompt engineering	Gives better context to reduce errors

What Should We Do?

- Just because an LLM sounds confident, doesn't mean it's correct.
- Our job as humans:
  - Be skeptical
  - Double-check
  - Use AI as a tool, not a source of truth

Wrap-Up

- LLM hallucinations are common — and avoidable
- Be critical thinkers and responsible users
- Use AI, but don’t lose your human judgment!

Understanding Temperature in LLMs

What is Temperature?

- Temperature = randomness or creativity in model output
- It's like an "imagination knob" for the AI
- Controls how the model picks words

Higher temperature = more surprising or diverse outputs
Lower temperature = more predictable, safer outputs

How It Works

- If temperature = 1, the model uses the original probabilities.
- If temperature < 1 (like 0.2), the model leans heavily on the most likely words.
- If temperature > 1 (like 1.5), the model spreads out and considers less likely words more often.

Fun Analogy – Spaghetti Example

- Prompt: "The chef cooked a plate of _____"
- Temp 0.2: "pasta" (expected)
- Temp 1.0: "noodles", "risotto", "spaghetti"
- Temp 2.0: "sunlight", "jazz", "ideas“

The higher the temperature, the more creative — and possibly weird — the response.

When to Use Different Temperatures

Use Case	Temperature	Why
Factual Q&A	0.2–0.5	Reliable, accurate
Creative writing	0.7–1.3	More diverse and fun
Brainstorming	0.8–1.5	Generates wild ideas
Summarization	0.3–0.7	Keeps things clear

Why It Matters

- You can fine-tune the behavior of the same model by simply adjusting temperature:
  - For a chatbot giving factual answers: Use low temp
  - For writing poetry or jokes: Use high temp
- You don’t need a different model — just change temperature!
  - Use temperature to match your task's goals

Wrap-Up

- Temperature = your creativity control knob
- Low = focused and safe
- High = wild and free

Understanding Top-k and Top-p in LLMs

Why Control AI Output?

- Language models generate words by choosing from many possibilities
- Not all outputs are equally likely — some are better than others
- Top-k and Top-p help us decide which options the model can choose from

What is Top-k Sampling?

- In Top-k sampling, we tell the model:
- Look at only the top k most likely words — and pick randomly from them.
- So, if k=3, the model ignores everything else and chooses randomly among the top 3 candidates

Analogy: *Like picking your favorite ice cream from the top 5 most popular flavors*

Example — Top-k

- Prompt: “The cake was so…”

Top 5 predictions might be:

[delicious (0.4), sweet (0.3), good (0.1), moist (0.08), expensive (0.05)]

If k=2 → only “delicious” and “sweet” are in the running.

✔ Good for avoiding rare, weird words

❌ But can still feel repetitive with low k

What is Top-p (Nucleus) Sampling?

- Instead of choosing a fixed number of top words like Top-k, Top-p chooses dynamically:

  Pick the smallest set of words whose combined probability is at least p (like 90%).

- This is more flexible and adapts based on how confident the model is.

Analogy: *Like eating from a buffet until you're 90% full — you don't count how many dishes, just how satisfied you are*

Example — Top-p

Prompt: “He opened the door and saw...”

Top predictions might be:

[a dog (0.35), a ghost (0.25), his friend (0.15), nothing (0.1), a dragon (0.05)]

If p = 0.9, the model might include the top 4 choices since together they cover 90% of the probability. The "dragon" gets excluded.

- ✔ Adapts to the context
- ✔ Allows for more diversity when the model is uncertain
- ❌ Can include low-probability words in some situations if p is set too high

Comparing Top-k and Top-p

Feature	Top-k	Top-p (Nucleus)
Fixed size?	Yes (always k words)	No (changes based on probs)
Flexibility	Rigid cutoff	Dynamic, context-aware
Use case	More control, less surprise	More natural, adaptive text

Why It Matters

- Controls how predictable or surprising the model's output is
- Helps avoid nonsense while keeping creativity
- Can be combined with temperature for extra control

Visual Example

Prompt: "The sky is..."

Without filtering: Could pick anything (even "banana")

Top-k (k=3): Picks from: "blue", "cloudy", "clear“

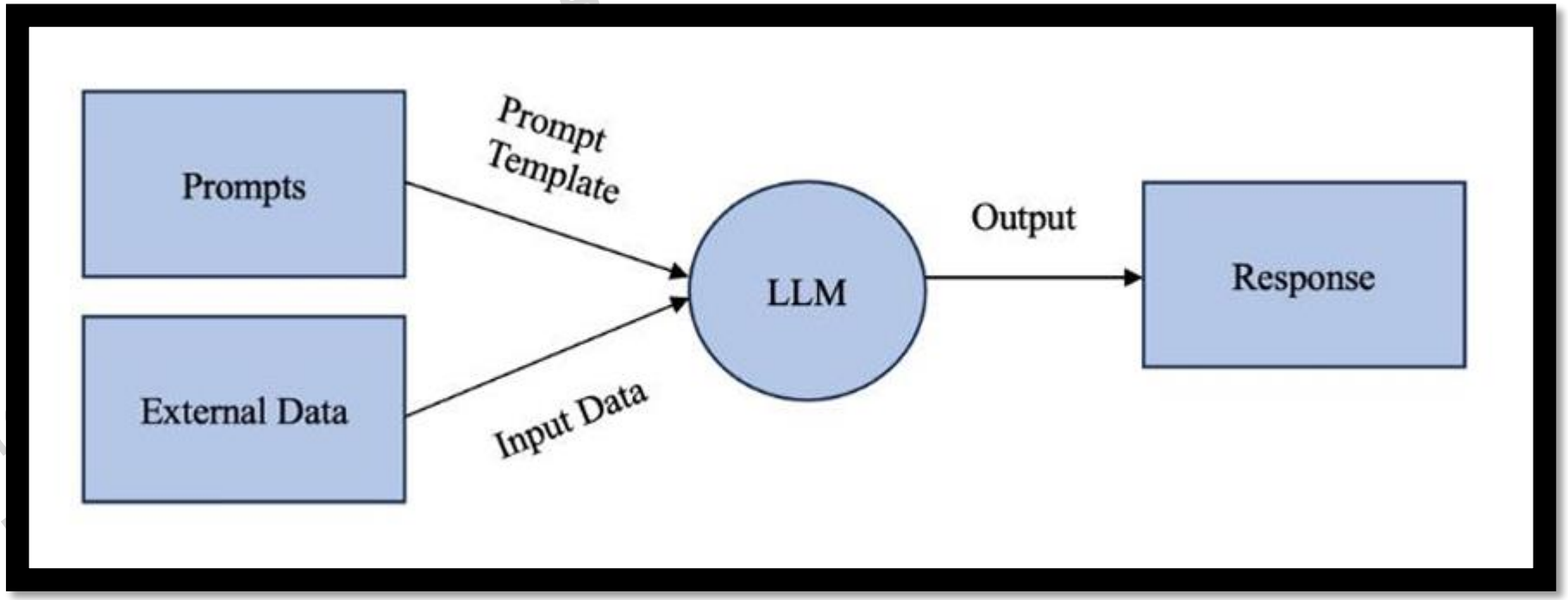
Top-p (p=0.9): Picks from: "blue", "bright", "stormy" (based on real probabilities)

Wrap up

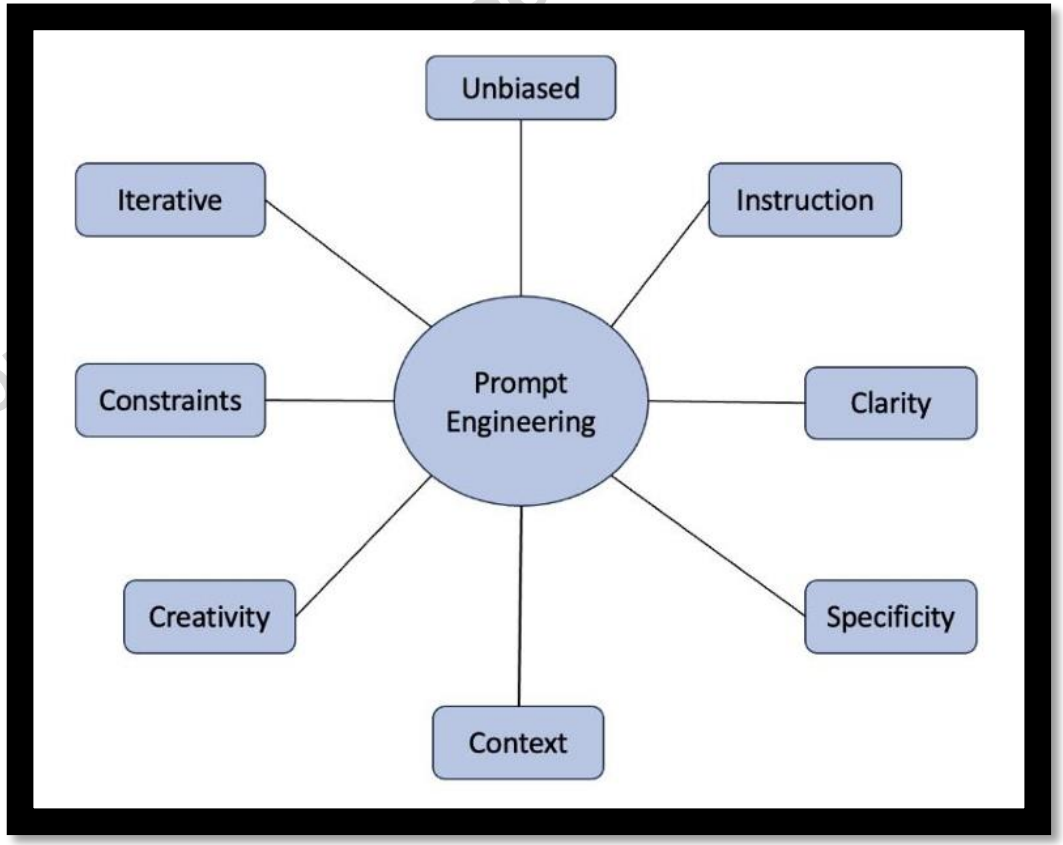
Top-k and Top-p give us control over randomness

- Use Top-k for more predictability
- Use Top-p for more flexibility and creativity
- Both help fine-tune what kind of content your LLM generates

Prompt Engineering



Key Elements of Prompt Engg.



Clarity

instead of asking a vague question like "Green Revolution," which lacks clarity, a better-engineered prompt for you would be "Describe the importance of Green Revolution."

Specificity

example, instead of asking a general question like "Tell me about Martin Luther King," which could result in a broad range of information, it's better to be more specific. A prompt like "Tell me about the achievements of Martin Luther King's family and his contributions to America" provides clear guidance to the LLM and increases the likelihood of obtaining the desired information.

Context

An example for a contextual prompt is, "At what time does flight number AB12 bound for New York to Florida leave from JFK airport tomorrow morning?"
The contexts given in the prompts are Flight# AB12, bound for New York to Florida, from JFK Airport and tomorrow morning.

Instructional Prompt

An example for an instruction-based prompt

"Generate a Scala function for Fibonacci series which takes a number as a parameter and returns the series generated in the form of an array back to the called method".

This statement gives you instructions to generate the method as per the parameters specified.

Eliminating Bias

For example, instead of using a prompt like "Why is Harrisburg, PA the best place for spending retirement life?" which assumes Harrisburg, PA is the ideal location for retirees, a less biased prompt would be "What are the advantages and disadvantages of spending retirement life in Harrisburg, PA?" This revised prompt allows for a more balanced exploration of the topic without presuming superiority or bias towards any specific location.

Creativity and Constraint

"Draw a picture of a lion standing in a thick forest while looking at a deer in black and white pencil drawing"

we're providing specific constraints. by explaining the background of the image.

Picture drawn by Open AI's Cosmic

Sequential Prompts

an instructional sequential prompt might involve asking the model to:

a) Generate a Scala function for the Fibonacci series called generateFibonacci, which takes an input number as a parameter and returns the series generated in an array format.
b) Create a main method to call the generateFibonacci function and pass a numeric value as a parameter.
c) Add comments to explain the purpose of the program.

a conversational sequential prompt could simulate a dialogue between a passenger and a flight booking agent:

Passenger: "I'd like to book a flight."
Model (Agent): "Sure, could you please provide your departure and destination cities?"
Passenger: "I'm flying from New York to Los Angeles."
Model (Agent): "Great. When would you like to depart and return?"

Hands-on Examples

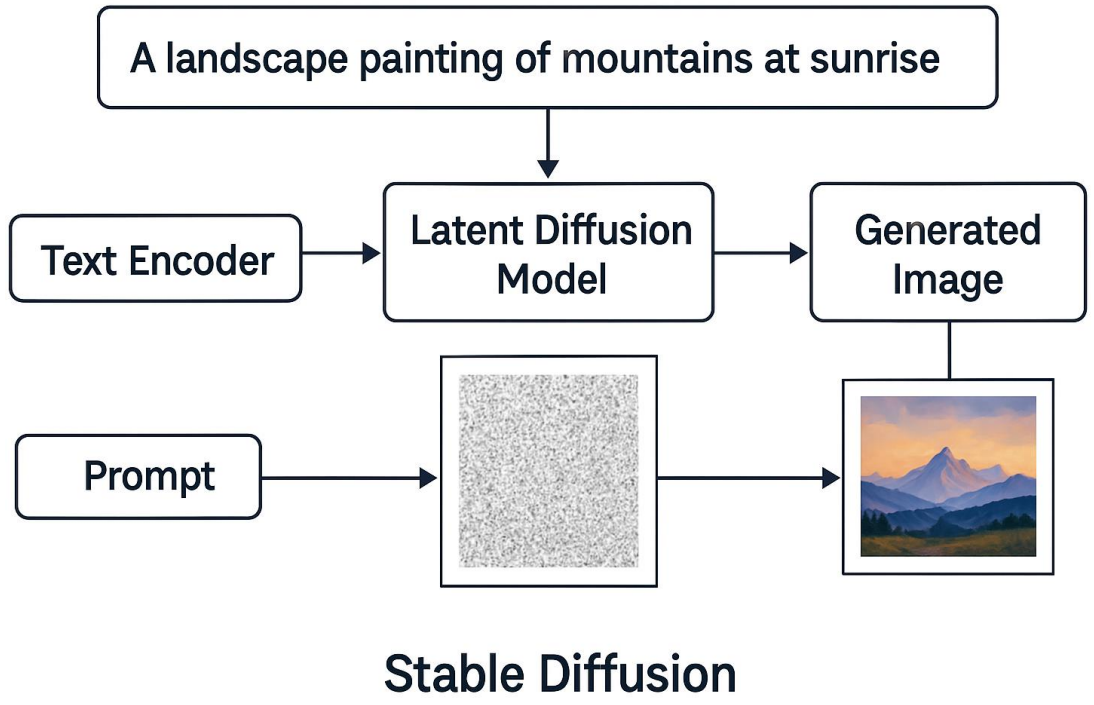
Stable Diffusion

What is Stable Diffusion?

- AI model that turns text prompts into images
- Open-source and efficient
- Used in creative, commercial, and research settings

High-Level Pipeline

1. Prompt → Text Encoder
2. Latent Noise → Denoising (U-Net)
3. Decode → Image



Stable Diffusion

What is Diffusion?

- Start: Add noise to an image until it’s pure static
- Goal: Learn how to reverse the process

Latent Diffusion Model

- Works in compressed "latent space"
- Faster and more memory efficient than pixel-space models

Components

Component	Role
Text Encoder (CLIP)	Converts prompt into embedding
U-Net	Denoises latent image
VAE Decoder	Turns latent output into an image
Scheduler	Manages noise removal steps

The Role of Noise

- Noise = Random data added to obscure the image
- Model learns to remove noise step-by-step

What Is Noise?

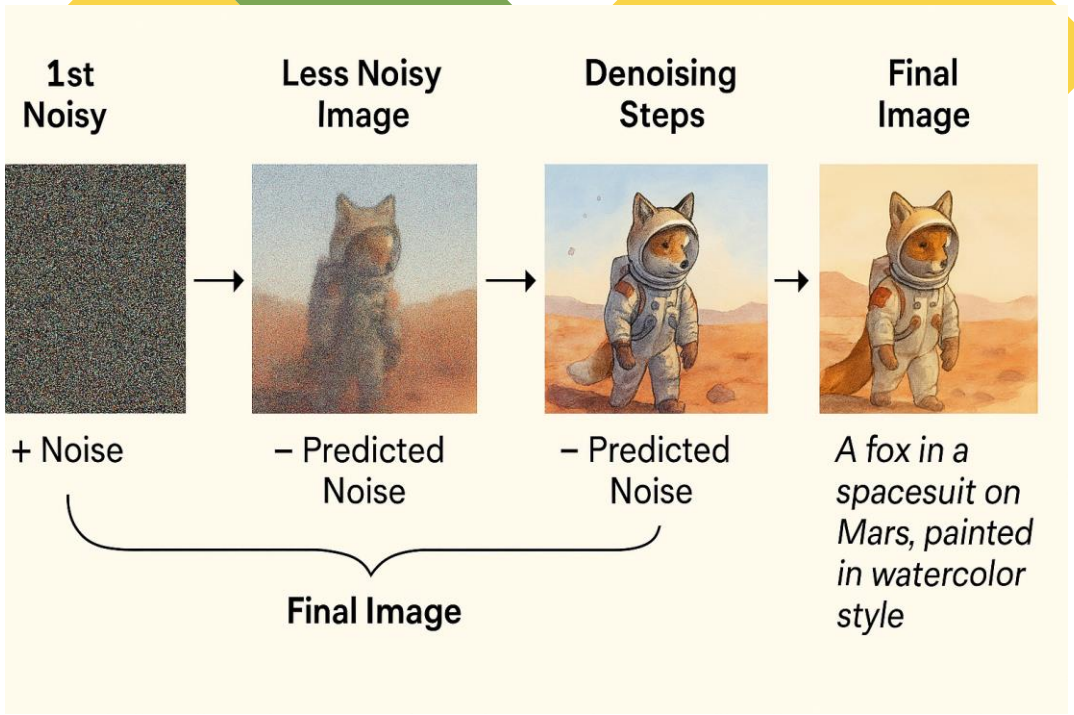
- Noise = Random pixel values from a normal distribution
- Used to obscure image content during training
- Acts as the “raw material” the model reshapes into an image

Why Noise Matters

- Models are trained to remove noise step-by-step
- Each step makes the image clearer and closer to the prompt
- Teaches the model to “understand” visual structure

Visual Example of Noise Process

- 0% noise: Original image
- 25% noise: Slightly degraded
- 50% noise: Blurry/static
- 100% noise: Complete randomness



Summary of Noise Concept

- Noise is added intentionally during training
- Stable Diffusion learns how to reverse it
- The model's "creativity" is in how well it removes noise to match the prompt

Training Data

- Trained on LAION-5B: billions of image-text pairs
- Model learns styles, objects, language associations

Prompt Parameters

- Guidance scale: How closely to follow the prompt
- Steps: How many noise removal iterations
- Seed: Makes output repeatable

Deployment Considerations

- Local, on-prem, or cloud
- Use libraries like diffusers, UIs like Gradio
- Needs GPU (8GB+ VRAM recommended)

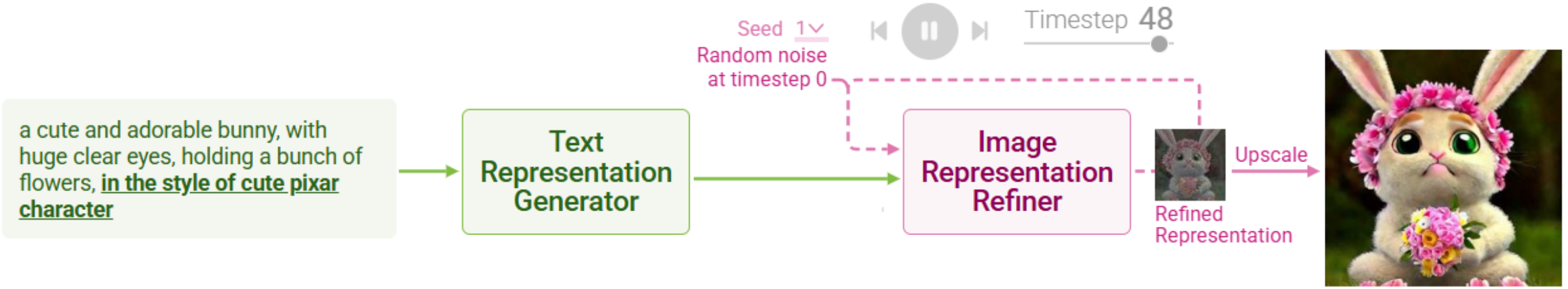
Summary

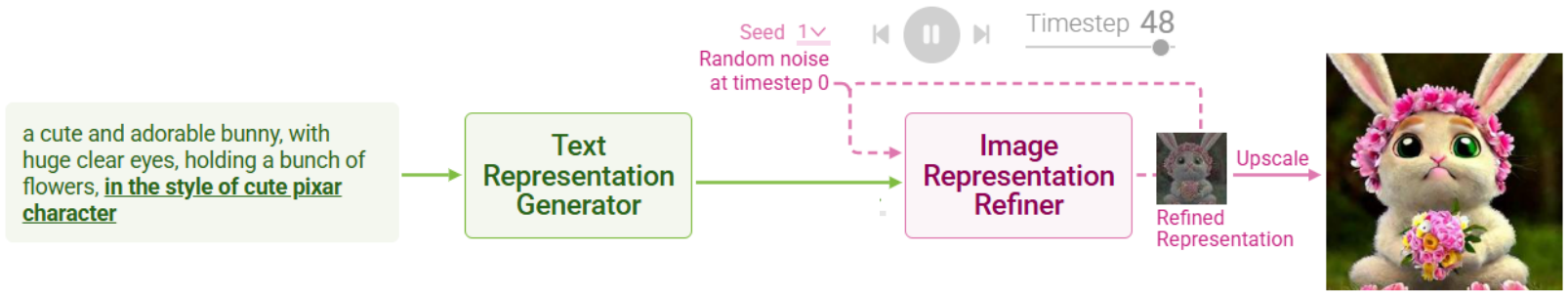
- Stable Diffusion: Efficient, flexible image generation
- Built with text encoders, U-Nets, VAEs
- Works by denoising random noise using language prompts

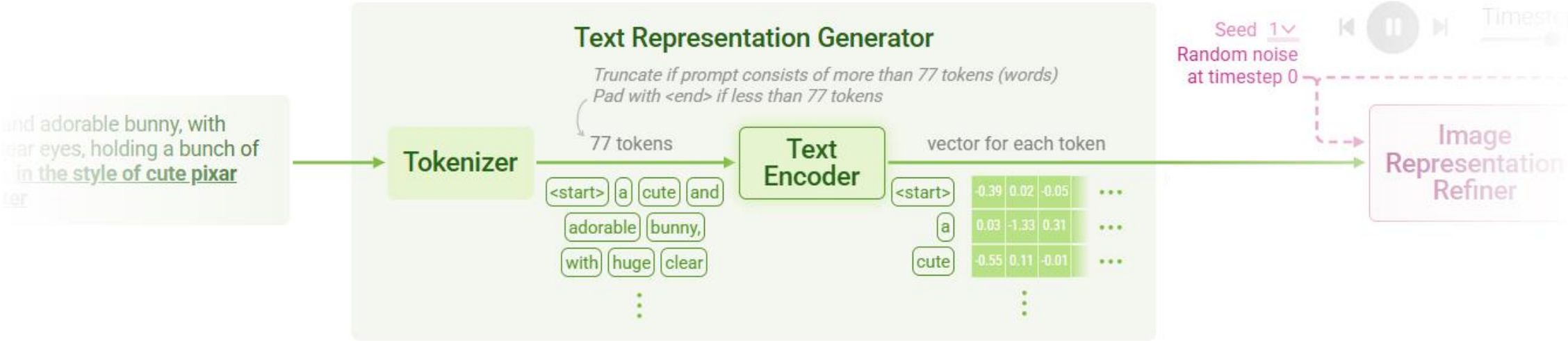
Stable Diffusion

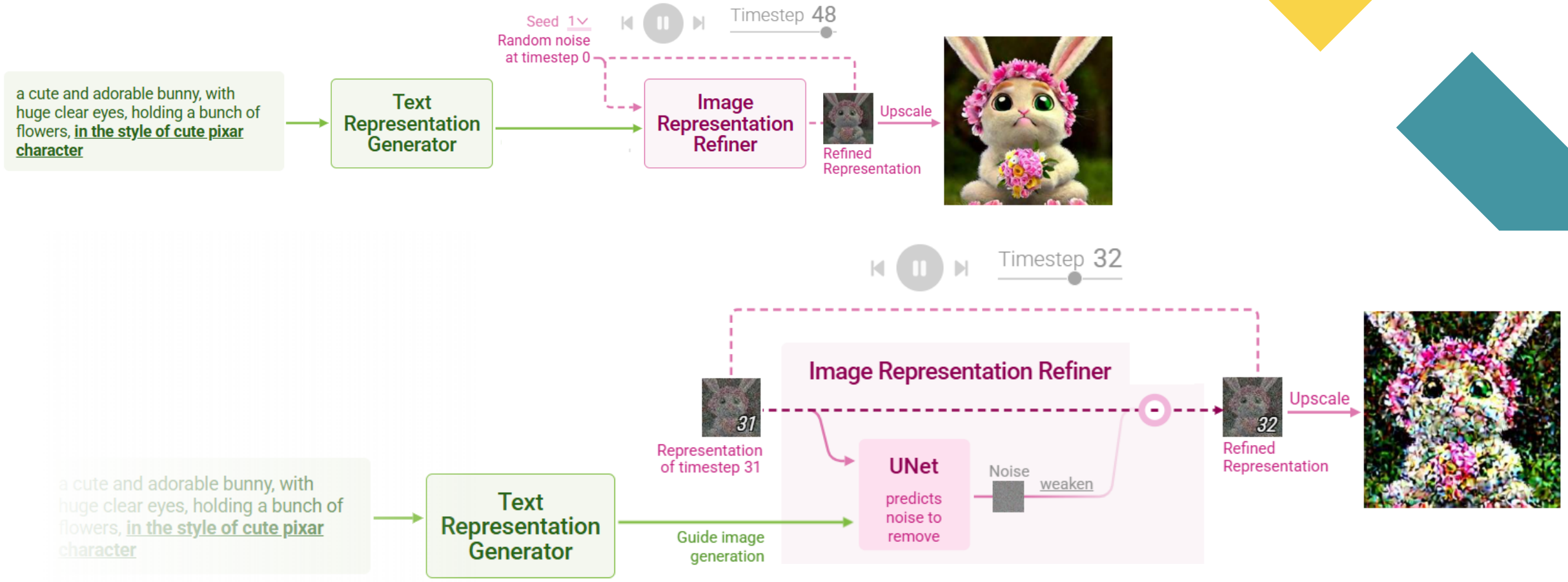
What is Stable Diffusion?

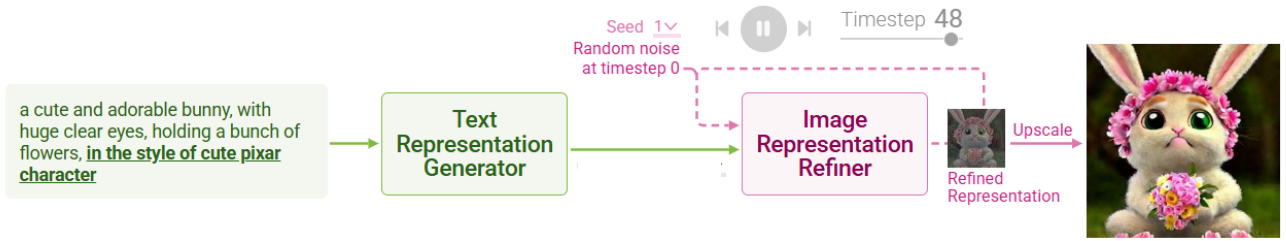
Stable Diffusion is a text-to-image model that transforms a text prompt into a high-resolution image.

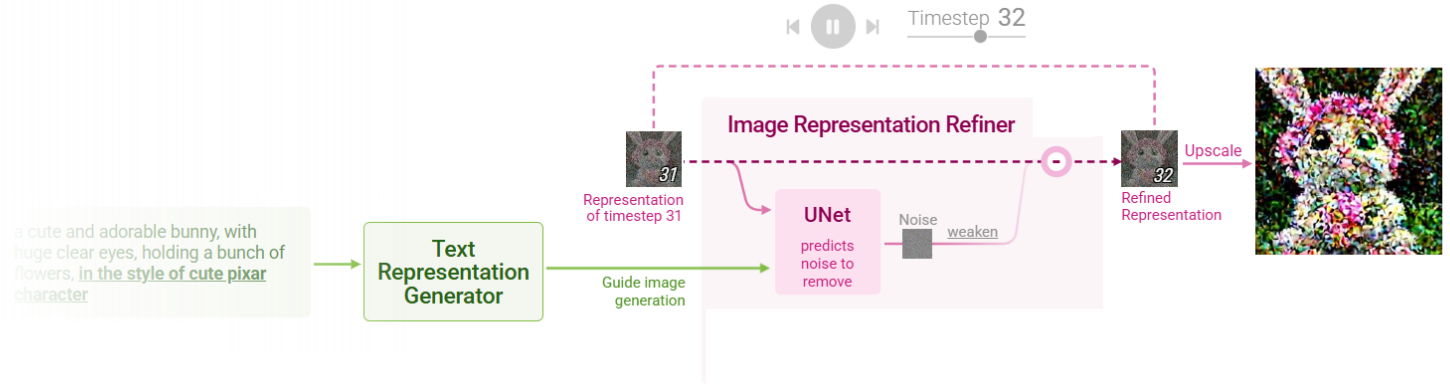


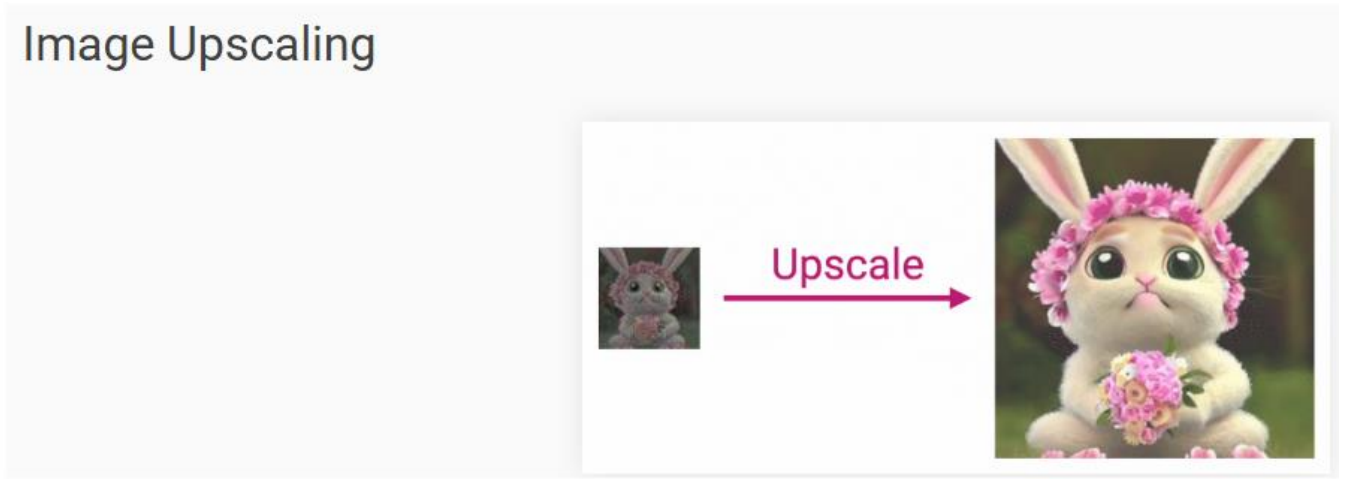

a cute and adorable bunny, with huge clear eyes, holding a bunch of flowers, in the style of cute pixar character
Text Representation Generator
Seed 1
Random noise at timestep 0
Timestep 48
Image Representation Refiner
Refined Representation
Upscale

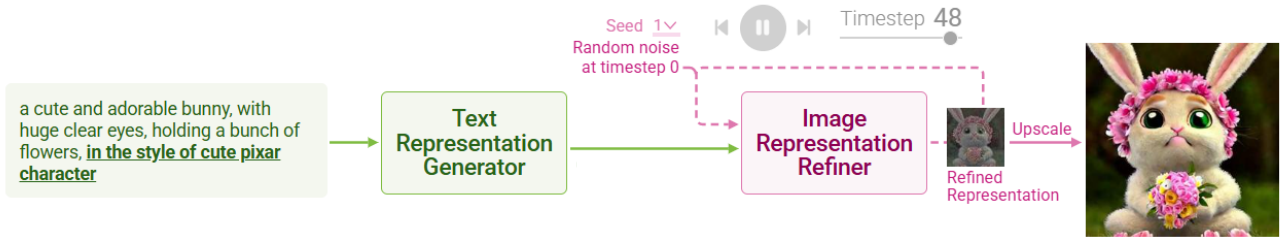

Text Representation Generator
Truncate if prompt consists of more than 77 tokens (words)
Pad with <end> if less than 77 tokens
Tokenizer
77 tokens
<start> a cute and
adorable bunny,
with huge clear
Text Encoder
vector for each token
<start> -0.39 0.02 -0.05
a 0.03 -1.33 0.31
cute -0.55 0.11 -0.01
Seed 1
Random noise at timestep 0
Image Representation Refiner

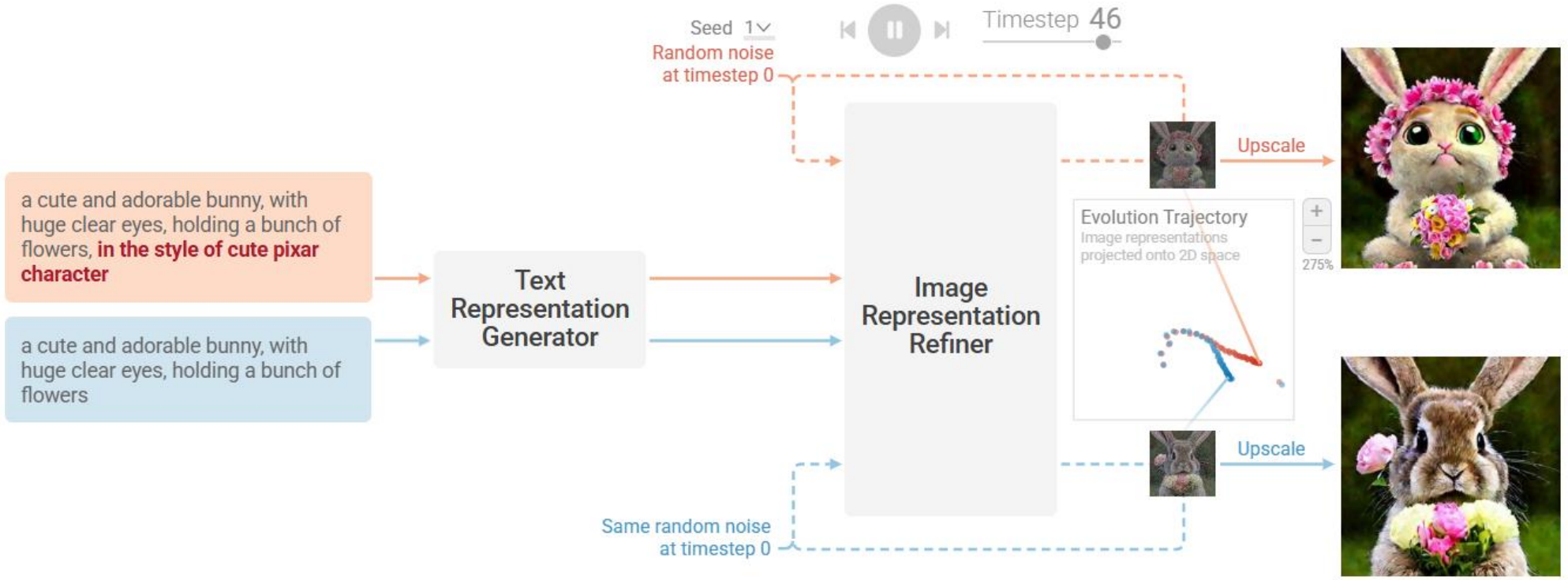

Seed 1
Random noise at timestep 0
Timestep 48
a cute and adorable bunny, with huge clear eyes, holding a bunch of flowers, in the style of cute pixar character
Text Representation Generator
Image Representation Refiner
Refined Representation
Upscale
Timestep 32
Image Representation Refiner
31
Representation of timestep 31
UNet
predicts noise to remove
Noise
weaken
32
Refined Representation
Upscale
a cute and adorable bunny, with huge clear eyes, holding a bunch of flowers, in the style of cute pixar character
Text Representation Generator
Guide image generation



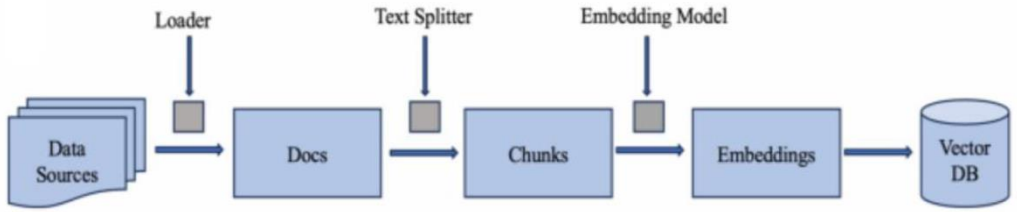





Seed 1
Random noise at timestep 0
Timestep 48
a cute and adorable bunny, with huge clear eyes, holding a bunch of flowers, in the style of cute pixar character
Text Representation Generator
Image Representation Refiner
Refined Representation
Upscale


Seed 1
Random noise at timestep 0
Timestep 46
a cute and adorable bunny, with huge clear eyes, holding a bunch of flowers, in the style of cute pixar character
a cute and adorable bunny, with huge clear eyes, holding a bunch of flowers
Text Representation Generator
Image Representation Refiner
Same random noise at timestep 0
Upscale
Upscale
Evolution Trajectory
Image representations projected onto 2D space
275%

Steps for loading Data into Vector DB



Documents

Vector DB can process various types of documents such as text, images, videos, audio, PDF files, and more. These documents are stored in Vector DB as encoded vector embeddings.

Chunks

The documents that feed in into the AI model is divided into multiple chunks based on the text splitter. While splitting the document into multiple chunks, we can overlap fixed number of number of characters between the adjacent chunks to provide the continuity. The number of characters to overlap can be set through the parameter while splitting the documents.

Vector Embeddings

Vector embeddings are the numerical representation of data stored in the vector store. The objects such as text data, image, audio, video can be converted into numerical vectors using embedding model and stored in the vector store for performing similarity search, grouping and categorization of data.

LangChain

At its core, LangChain is a framework built around LLMs.

We can use it for chatbots, Generative Question-Answering (GQA), summarization, and much more.

The core idea of the library is that we can “chain” together different components to create more advanced use cases around LLMs.

Chains may consist of multiple components from several modules:

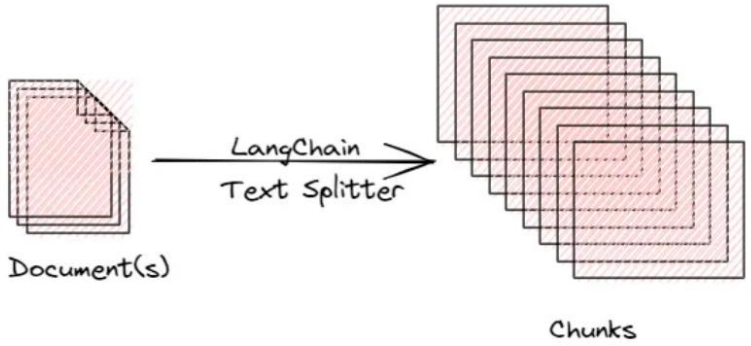
- Prompt templates
- LLMs
- Agents
- Memory

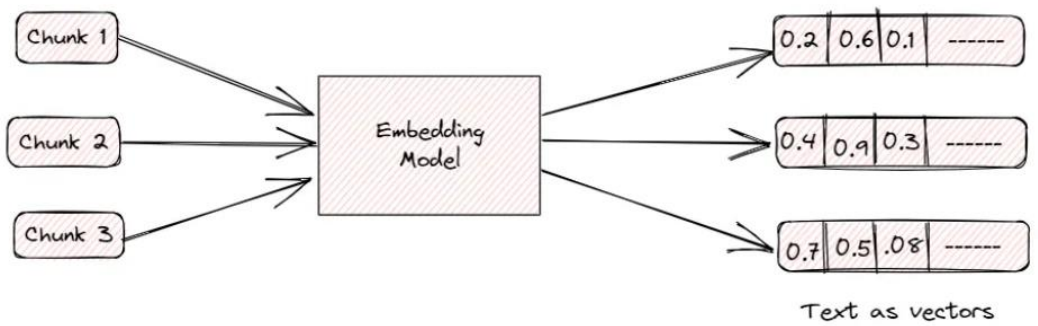
FAISS

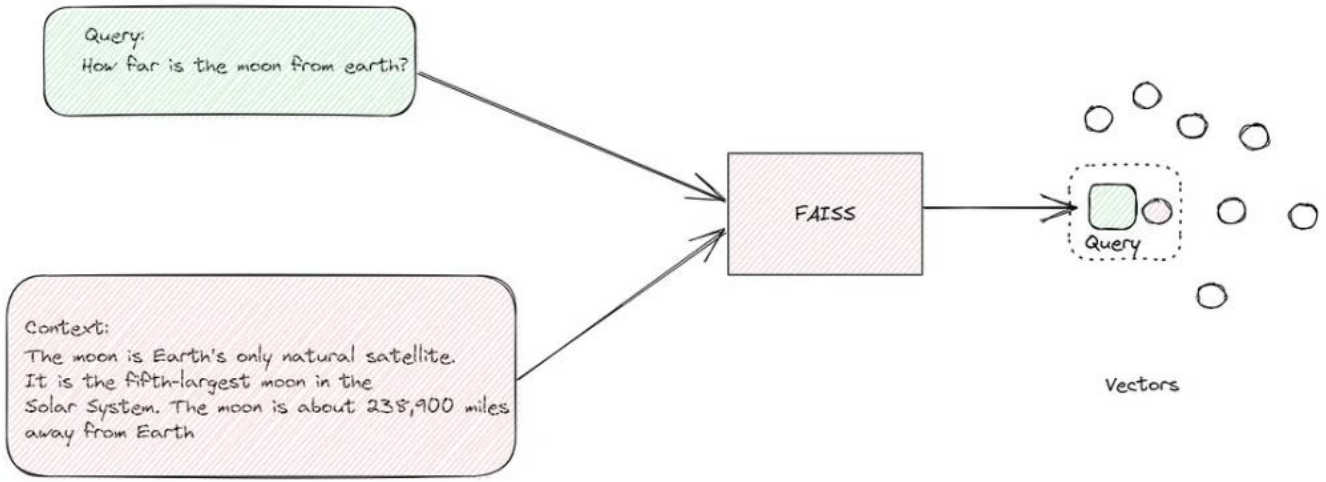
- Facebook AI Similarity Search
- Library of similarity search algorithms
- Enables retrieval of relevant documents based on semantic similarities
- High-dimensional indexing capabilities and fast search performance

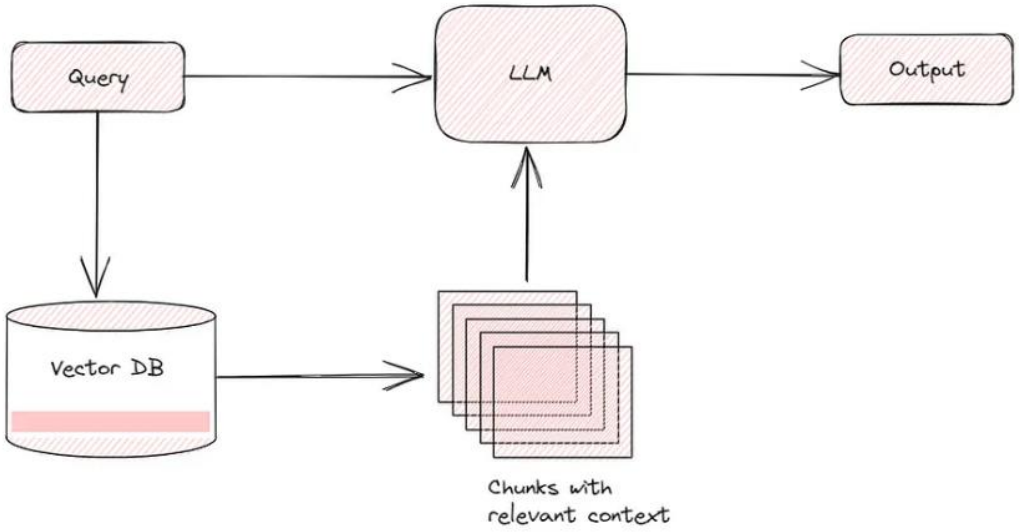
Answering questions from a document

- Splitting the document into smaller chunks.
- Convert text chunks into embeddings.
- Perform a similarity search on the embeddings.
- Generate answers to questions using an LLM.

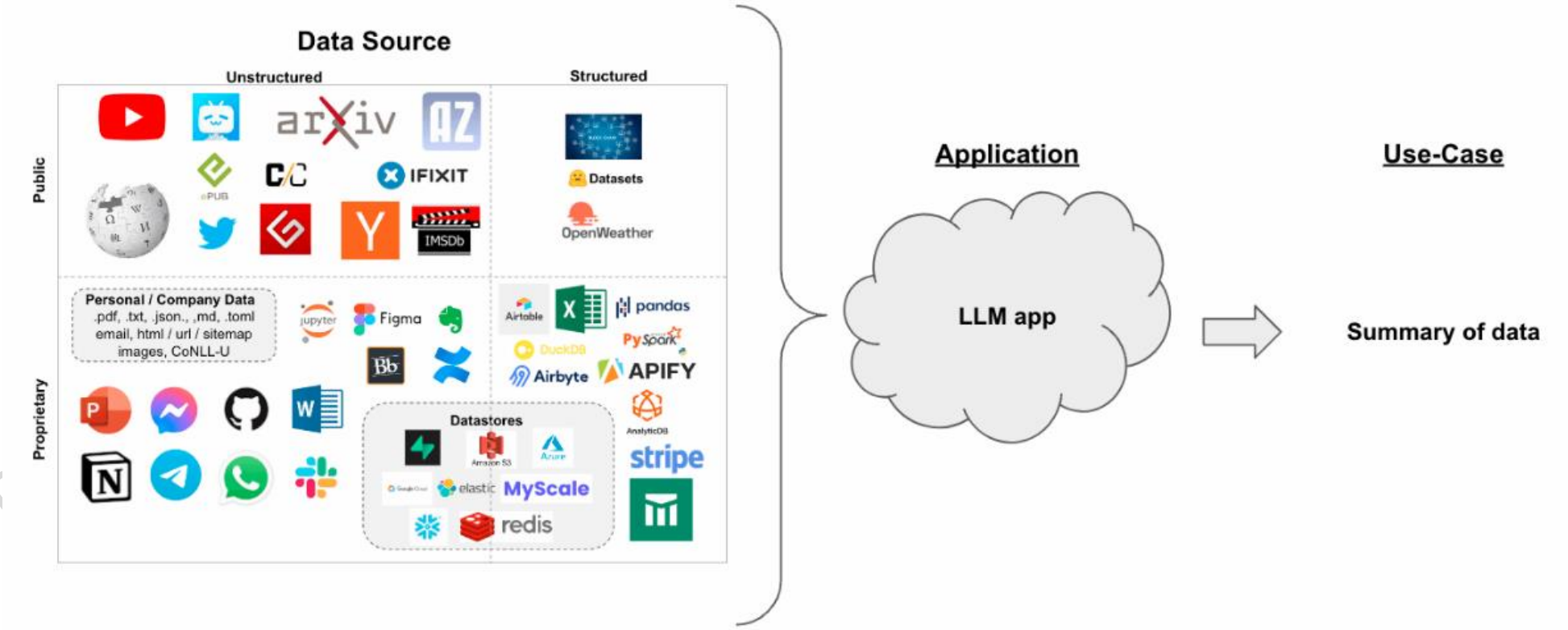

LangChain
Text Splitter
Document(s)
Chunks


Chunk 1
Chunk 2
Chunk 3
Embedding Model
0.2 0.6 0.1 ------
0.4 0.9 0.3 ------
0.7 0.5 .08 ------
Text as vectors

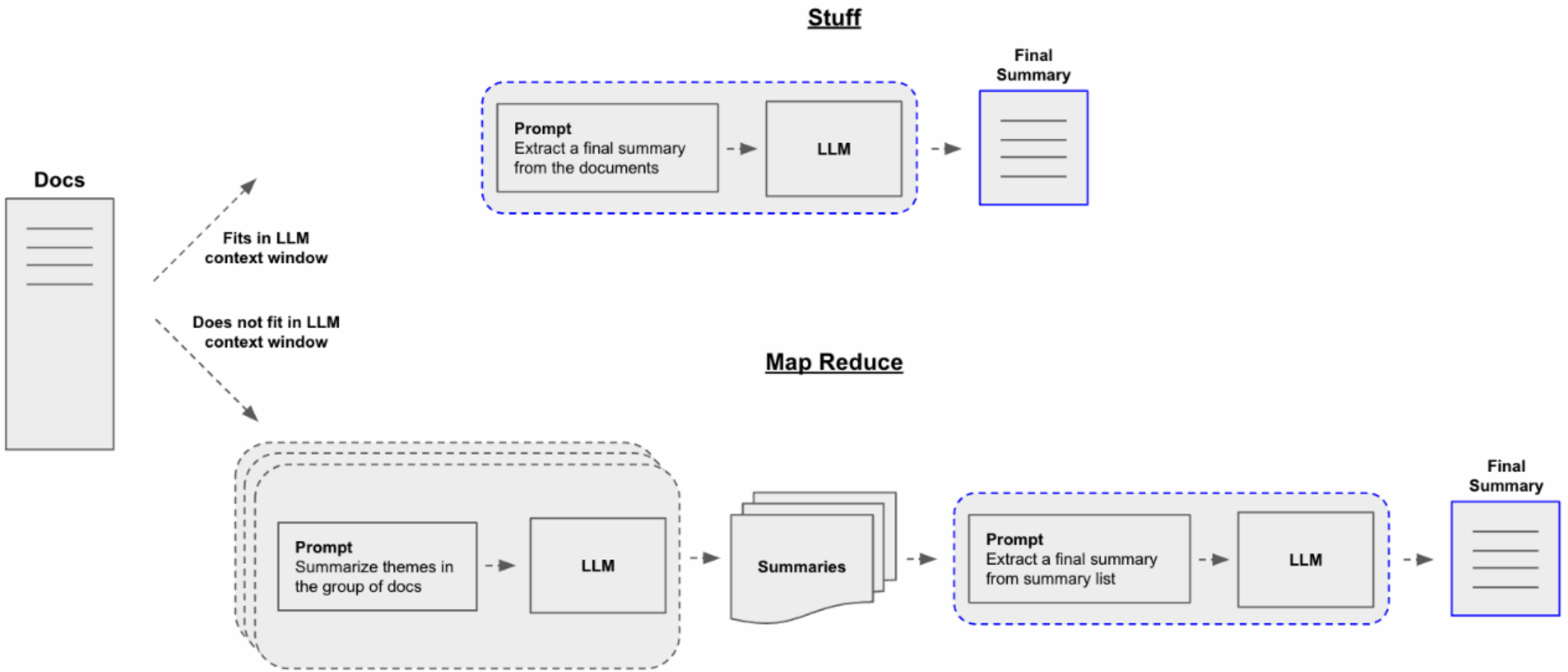

Query:
How far is the moon from earth?
Context:
The moon is Earth's only natural satellite. It is the fifth-largest moon in the Solar System. The moon is about 238,900 miles away from Earth
FAISS
Query
Vectors


Query
LLM
Output
Vector DB
Chunks with relevant context

Summarization



MapReduce


Stuff
Final Summary
Prompt
Extract a final summary from the documents
LLM
Docs
Fits in LLM context window
Does not fit in LLM context window
Map Reduce
Final Summary
Prompt
Summarize themes in the group of docs
LLM
Summaries
Prompt
Extract a final summary from summary list
LLM

LlamaIndex

LlamaIndex focuses on efficient data indexing and retrieval for LLM applications

Focus: Data ingestion, indexing, and retrieval for LLMs

Advantages:

- Efficient data indexing and retrieval.
- Simpler interface and easier to use for specific tasks like building search engines and knowledge bases.
- Built-in vectorstore.
- Good for applications that need to process large amounts of data.
- **If you're building a system that needs to quickly search and retrieve information from a large document collection, LlamaIndex might be a good choice.**

LangChain Vs LlamaIndex

- LangChain has a variety of loaders that can load text, PDFs, and even web pages.
- LangChain's splitters, called TextSplitters, allow you to customize how text is broken up—by characters, words, or sentences.
- LangChain's VectorStoreIndex is used to create an index from the document embeddings, enabling retrieval based on similarity searches.
- LangChain allows for flexible chains, enabling complex workflows with different components such as LLMChain for combining a language model with other tasks.
- If your application involves various types of data sources (e.g., text, APIs, PDFs) and you need to handle different retrieval strategies, LangChain's flexibility becomes very useful.

- LlamaIndex (formerly GPT Index) has a similar approach to loading documents and supports additional formats like Pandas DataFrames.
- LlamaIndex uses TokenTextSplitter for splitting documents based on token count, ensuring that the chunks fit within the model's token limits.
- LlamaIndex simplifies indexing through its GPTTreeIndex, which uses a tree-like structure for efficient retrieval.
- LlamaIndex provides a similar approach with its query engine, which combines retrieval and language model generation.
- If your use case revolves around text-heavy data or you are looking for a quick implementation with minimal setup, LlamaIndex simplifies the process.

BLEU and ROUGE

BLEU (Bilingual Evaluation Understudy)

BLEU is a precision-based metric that measures how many n-grams in the generated text also appear in the reference (ground truth).

- Commonly used in machine translation
- Focuses on matching exact words/sequences

How It Works

- Compares n-grams between generated text and reference.
- Calculates precision: how many predicted n-grams are correct.
- Uses a brevity penalty to avoid very short outputs.

Reference:

`the cat is on the mat`

Prediction:

`the cat is on mat`

Unigrams (1-grams):

- Prediction: `['the', 'cat', 'is', 'on', 'mat']`
- Reference: `['the', 'cat', 'is', 'on', 'the', 'mat']`
- Precision = 5/5 = 1.0

BLEU Score Formula

```
BLEU = BP × exp(Σ (weight_n × log precision_n))
```

Where:

- `BP` = Brevity Penalty (penalizes short predictions)
- `precision_n` = precision for n-gram
- `weight_n` = usually 0.25 for BLEU-4 (1- to 4-grams equally weighted)

✅ Strengths:

- Good for **exact matches**
- Useful in **translation**

❌ Limitations:

- Not good for **paraphrases**
- Ignores **meaning/synonyms**
- Sensitive to **word order**

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE is a set of recall-based metrics used to evaluate summarization and generation tasks.

- Focuses on how much of the reference content is covered by the generated text.
- Measures overlap of n-grams, sequences, or words.

ROUGE Variants

Type	Measures
ROUGE-N	Overlap of n-grams
ROUGE-L	Longest common subsequence (LCS)
ROUGE-W	Weighted LCS
ROUGE-S	Skip bigram co-occurrence (unordered)

Most Common

- ROUGE-1 → Overlap of unigrams (words)
- ROUGE-2 → Overlap of bigrams
- ROUGE-L → Based on longest common subsequence

Reference:

`the cat is on the mat`

Prediction:

`cat on mat`

- **ROUGE-1 recall** = How many words from the reference appear in the prediction?
  `['cat', 'on', 'mat']` vs `['the', 'cat', 'is', 'on', 'the', 'mat']`
  Matches: `['cat', 'on', 'mat']` = 3/6 = **0.5**
- **ROUGE-2 recall** = Match bigrams: `['cat on', 'on mat']`
  vs `['the cat', 'cat is', 'is on', 'on the', 'the mat']`
  Matches: 0 → **0.0**

✅ Strengths:

- Works well for **summarization**
- Captures **recall** (how much of the reference is recovered)
- ROUGE-L is **flexible with word order**

❌ Limitations:

- Sensitive to **surface form**
- May miss semantic similarity or paraphrases

BLEU vs ROUGE

Feature	BLEU	ROUGE
Focus	**Precision**	**Recall**
Good for	Translation	Summarization, QA
Penalizes	Short predictions	Missing reference parts

Vector Databases & Similarity Search

Understanding Vector Databases & Similarity Search

- Vector embeddings
- Vector search
- Cosine similarity & dot product

What Are Vector Embeddings?

- High-dimensional representations of data
- Capture semantic meaning
- Used in NLP, vision, and audio

What Is a Vector Database?

- Stores and indexes embeddings
- Searches by semantic similarity
- Replaces keyword match with vector math

What Is Similarity Search?

- Turn a query into a vector
- Find nearby vectors in the database
- Return top-k similar results

How Do We Measure Similarity?

- Dot Product
- Cosine Similarity
- Higher = more similar

Dot Product

- A · B = sum of element-wise multiplication
- Measures magnitude and alignment
- Simple and fast

```
A = [1, 2], B = [3, 4]
Dot Product = 1×3 + 2×4 = 11
```

Cosine Similarity

- Measures angle between vectors
- Normalizes for vector length
- Range: -1 to 1

$$\cos(\theta) = (A \cdot B) / (||A|| \times ||B||)$$

Popular Vector Databases

- FAISS – in-memory & fast
- Qdrant – open-source with filtering
- Pinecone – fully managed, scalable
- Weaviate – vector + metadata + APIs

How It All Works

1. Embed text or data
2. Store vectors
3. Embed the query
4. Search using similarity
5. Return top results

Use Cases

- Semantic search
- Chat with documents
- AI Q&A systems
- Product recommendations
- Image/audio retrieval

Summary

- Vector DBs store and retrieve high-dimensional embeddings
- Similarity search uses cosine or dot product
- Powering smarter search & AI apps

Similarity Search

What Is Similarity Search?

- Input is converted to a vector
- Compare it to stored vectors
- Return most similar vectors (Top-k)

Dot Product

- Formula:

$$\text{Dot}(A, B) = \sum_{i=1}^{n} A_i \cdot B_i$$

- Measures both direction **and** magnitude
- Larger = more similar

Cosine Similarity – Angle-Based Similarity

- Formula:

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

- Range: -1 (opposite) to 1 (same direction)
- Ignores magnitude, focuses on **angle**

Euclidean Distance – Straight-Line Distance

- Formula:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

- Measures the “as-the-crow-flies” distance
- Smaller = more similar

Comparing Similarity Metrics

Metric	Value Range	Similarity Behavior
Dot Product	`-∞ to +∞`	Higher = more similar
Cosine Similarity	`-1 to 1`	Closer to 1 = more similar
Euclidean Distance	`0 to ∞`	Smaller = more similar

When to use __ ?

Metric	Best Used For
Cosine Similarity	Text, NLP, semantic search
Dot Product	Large-scale search with normalized vectors
Euclidean Distance	Clustering, image search, spatial data

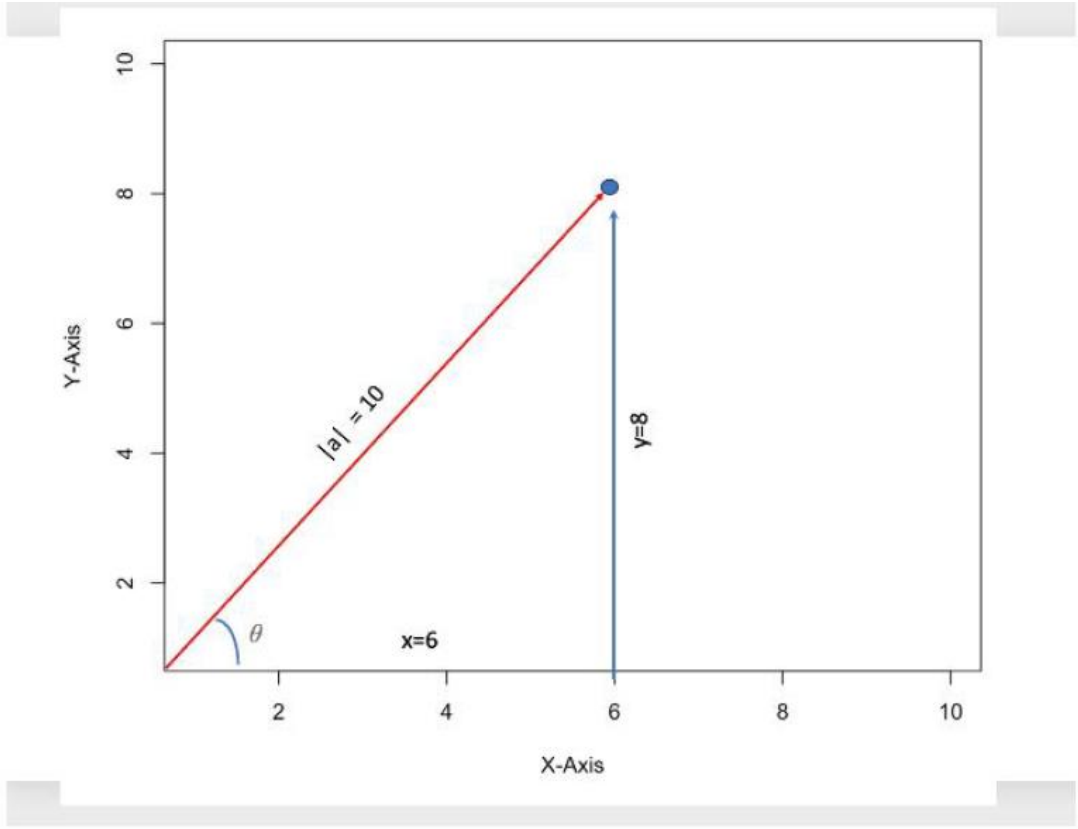
Use Case

Use Case	Recommended Metric
Chatbots (RAG)	Cosine or Dot
Text document search	Cosine Similarity
Large-scale fast retrieval	Dot Product + FAISS
Image similarity	Euclidean Distance
Embedding clustering (e.g., k-means)	Euclidean Distance

Vector represent both magnitude and direction. The magnitude of a vector is represented by the vector size or length. The norm of a vector is the square root of the sum of square of each value and is represented as

```
magnitude function(x) =   sqrt(sum(x^2))
```

For example, let's assume that we have a vector a = (6,8). The magnitude of the vector is calculated as follows



Magnitud

$$|a| = Sqrt((6*6) + 8*8))$$

$$|a| = Sqrt(36+64)$$

$$|a| = 10$$

The magnitude of the vector is 10.

Dot Product

The dot product is one of the most widely used vector operations for finding the similarity between the two vectors. The dot product is represented as it's a Euclidean magnitude of two vectors and the cosine of the angle between them. The dot product takes two equal lengths of sequences of numbers and returns a number.

$$a.b = |a| * |b| * cos(\theta)$$

Cosine similarity

Based on the above formula, the cosine similarity can be calculated as follows:

$$cos(\theta) = a.b / (|a| * |b|)$$

This formula is used to find the similarities between two vectors. The result of the cosine similarity always ranges between -1 and 1.

Angle between two vectors

We can derive the angle between two vectors θ. using the dot product formula as follows

$$\theta = \cos^{-1} [(\boldsymbol{a} \cdot \boldsymbol{b}) / (|\boldsymbol{a}| |\boldsymbol{b}|)]$$

Based on the above formulas, we understand the cosine similarity and the angle between two vectors which explains how far or how close they are against each other.

Cosine similarity is 1: When the value of cosine similarity is 1, the angle between the vectors becomes 0 degrees. This indicates that both vectors are the same which means that they point in the same direction.

Cosine similarity is 0: The angle between them is 90 degrees which means the two vectors are not related to each other.

Cosine similarity is -1: The angle between the 2 vector is 180 degrees which means that the two vectors are different and they point in opposite direction.

Example 1

3-dimensional vector for calculating the cosine similarity to find out what extent they are similar.

X = (1,2,3) and
Y = (4,-5,6)

The magnitude of X, |X| is

*|X| = Sqrt((1*1) + (2*2) + (3*3))*
|X| = Sqrt(1+4+9)
|X| = Sqrt(14)
|X| = 3.74 magnitude

magnitude of Y, |Y| is

*|Y| = Sqrt((4*4) + (-5*-5) + (6*6))*
|Y| = Sqrt(16+25+36)
|Y| = Sqrt(77),
|Y| = 8.77 magnitude

The dot product of the 2 vectors is derived as follows:

*X.Y = (1 x4) + (2 * -5) + 3 * 6)*
X.Y = 4 -10 + 18
X.Y = 12 dot product

Cosine similarity = cos(θ) =

*cos(θ) = 12 / (3.74 * 8.77) = 0.36*

Cosine similarity = cos(θ) = 0.36

θ, the angle between the two vectors is 'inverse of cosine'

$$\theta = \cos^{-1}\left[X.Y / |X| * |Y|\right]$$

$$\theta = \cos^{-1}[0.36] = 68.9$$

θ = 68.9 °

angle between the two vectors

Conclusion: *For the above two vectors X and Y, the cosine similarity is 0.36 which is less than 1 which means that the 2 vectors are not same.*

Example 2

3-dimensional vector for calculating the cosine similarity to find out what extent they are similar.

A = (1,2,3) and
B= (1,2,3)

The magnitude of A, |A| is

*|A| = Sqrt((1*1) + (2*2) + (3*3))*
|A| = Sqrt(1+4+9)
|A| = Sqrt(14)
|A| = 3.74

the magnitude of B, |B| is

*|B| = Sqrt((1*1) + (2*2) + (3*3))*
|B| = Sqrt(1+4+9)
|B| = Sqrt(14)
|B| = 3.74

The dot product of the 2 vectors is derived as follows:

*X.Y = (1 x1) + (2 * 2) + 3 * 3)*
X.Y = 1 + 4 + 9
X.Y = 14

*Cosine similarity = cos(θ) = X.Y / |X| * |Y|*

cos(θ) = 14 / (13.98)

Cosine similarity = cos(θ) = 1.00

θ, the angle between the two vectors is 'inverse of cosine

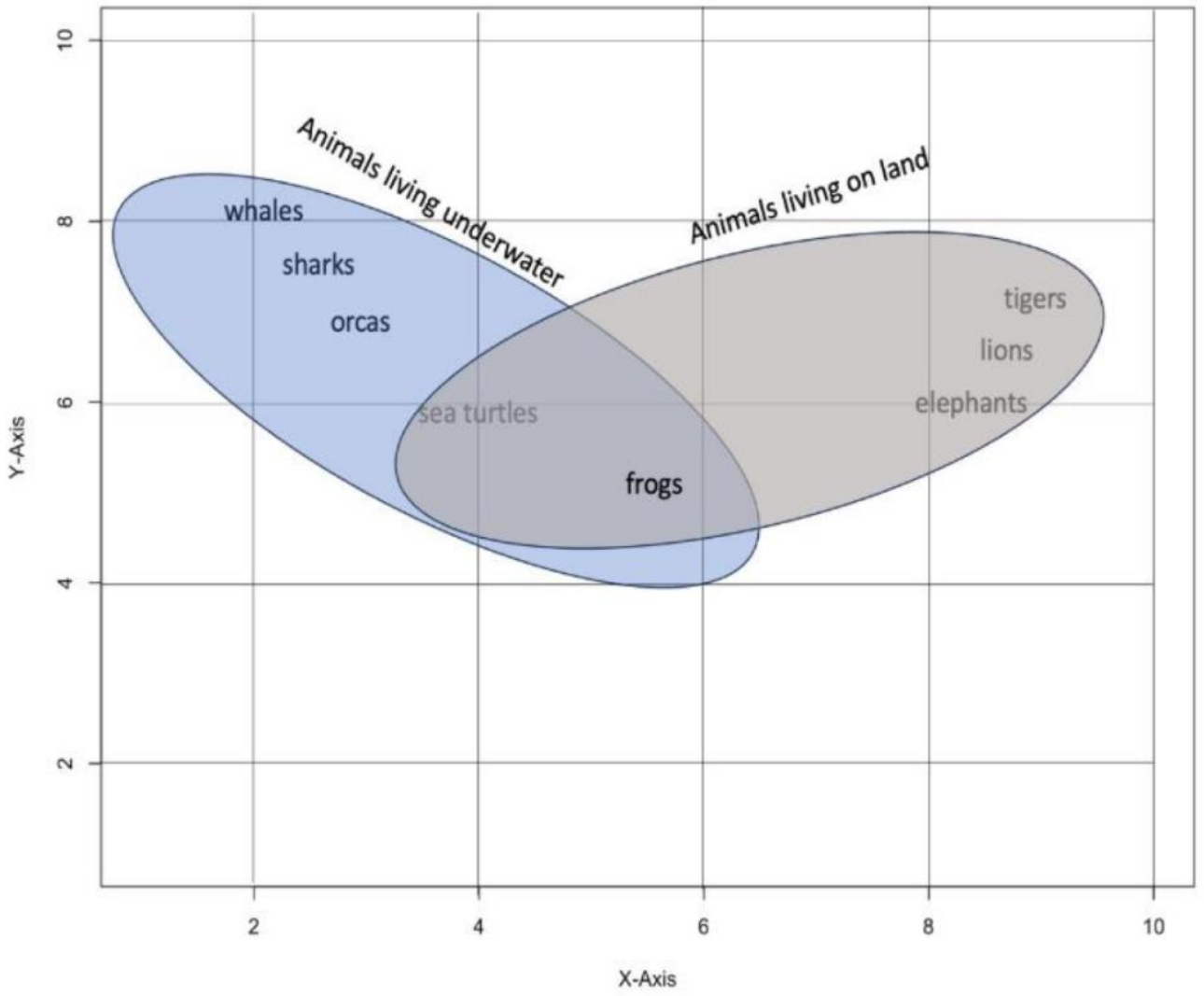
$$\theta = \cos^{-1}\left[X.Y / |X| * |Y|\right]$$

θ = 0 degrees.

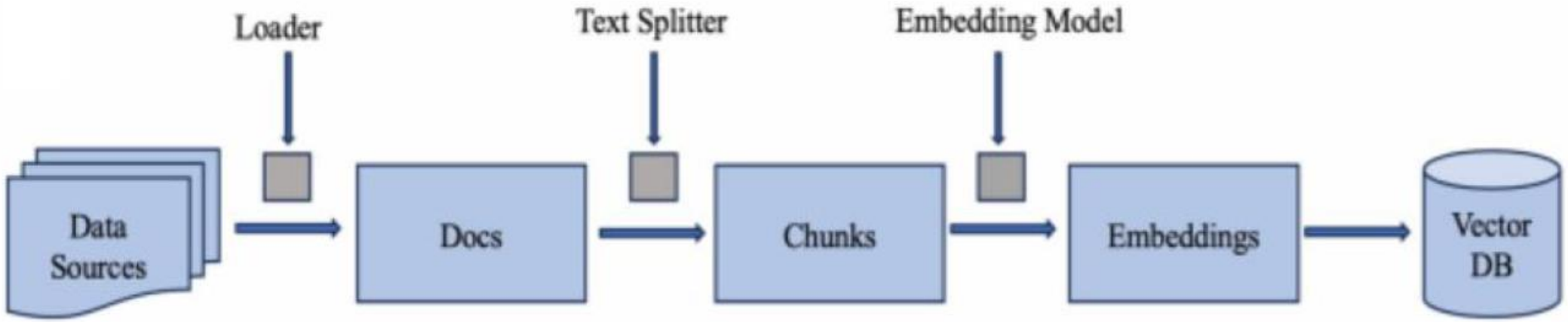
angle between the two vectors

The cosine similarity also indicates the nearest neighbors.

Sample Vector representation of text embeddings of underwater animals and land animals



Steps for loading Data into Vector DB



Retrieval-Augmented Generation (RAG)

What is RAG?

- Combines retrieval with text generation
- Connects LLMs to external data sources
- Dynamic, fact-aware output

RAG Architecture

1. Query → Embed → Retrieve
2. Retrieved docs → passed to LLM
3. LLM generates response with context

RAG Workflow

- Input: “What is vector search?”
- Embed query
- Retrieve top-k relevant docs
- LLM generates final answer using retrieved data

Why Use RAG?

- ✅ Grounded in facts
- ✅ Reduces hallucinations
- ✅ Easy to update (no retraining)
- ✅ Enables private data access

Tools for RAG

- Embeddings: Cohere, OpenAI, Sentence Transformers
- Vector Stores: FAISS, Qdrant, Pinecone
- Frameworks: LangChain, LlamaIndex

Use Cases

- AI customer support
- Chat with documents (PDFs, wikis)
- Legal and academic Q&A
- Internal knowledge bases
- Healthcare, research assistants

RAG vs Fine-tuning

	RAG	Fine-tuning
Data update speed	Real-time (just reindex)	Slow (need retraining)
Accuracy	High with good retrieval	Can drift or hallucinate
Cost	Lower (no retrain cost)	Higher (train + compute)
Flexibility	Can query many sources	Fixed to training data

Summary

- RAG = Retrieval + Generation
- Connects LLMs to external data
- Improves accuracy & real-world utility
- Built using embeddings, vector DBs, and LLMs