Generative AI for Everyone

[Free E-Book]

Natural Language Processing **3**
Key areas of NLP include: 3
How is NLP Useful in Generative AI? 3
Examples of NLP in Generative AI Tools 4
AI systems and tools **5**
The AI layers **6**
1. Supervised learning 8
2. Unsupervised learning 9
3. Semi-supervised learning 9
Deep learning **10**
Here's how ANNs work: 10
Introduction to generative AI **13**
Generative AI: Some fascinating metrics 15
How generative AI works 18
ML model vs. gen AI model **19**
Journey from traditional programming to neural networks to generative AI **20**
Large Language Models: Powering generative AI **22**
How do LLMs work? 22
Different Large Language Models 23
Components of an LLM **25**
How do LLMs learn? 26
Building an LLM application 27
LLMs use cases 28
Content creation 28
Education 29
Customer service and support 29
Research and development. 29
Entertainment and media 29
Limitations of LLMs **30**
LLM Hallucinations **30**
LLM hallucination mitigation strategies 32
1. Fine-tuning 32
2. Prompt engineering 33
3. Retrieval Augmented Generation (RAG) 33
RAG chunking strategies 35
Developing a Large Language Model **36**

Vector databases **37**
Vector Embeddings 38
How vector databases work 39
Traditional databases vs. vector databases **39**
The vector database landscape 41
How vector databases search and retrieve data 42
Some notable generative AI frameworks + tools **44**
LangChain 44
LlamaIndex 46
How LlamaIndex works 46
Hugging Face **47**
1. Model Hub: 47
2. Datasets: 47
3. Model Training & Fine-tuning Tools: 48
4. Application Building: 48
5. Community & Collaboration: 48
The Rise of Small Language Models **49**
Size: 50
Focus: 50
Limitations: 50
Prompt Engineering **51**
Generative AI Developer Stack **53**
Using Generative AI Responsibly **54**
Best Practices for Responsible Generative AI Use: 54
Data: 54
Development and Deployment: 55
Content and User Interaction: 55
Societal Impact and Governance: 55
Multimodal Models **56**
Examples of multimodal models 57
LLM-Powered Autonomous AI Agents **57**
Deploying GenAI Applications on Kubernetes: A Step-By-Step Guide! **58**
Why Deploy GenAI Applications On Kubernetes? 58
Let's Build a Full-Stack AI Application in React **59**
What is the Elegance SDK? 59
Key features 60
SingleStore for Generative AI Applications **60**
Using SingleStoreDB as a full-context vector database 60
Notable Companies Using SingleStore 61

Welcome to the exciting world of generative Artificial Intelligence (AI), a frontier in technology that is not just transforming how we interact with machines, but also reshaping the very fabric of creativity, design and data synthesis. This eBook is your doorway to understanding the fundamentals, advancements and immense potential of this groundbreaking field.

"**Generative AI: A Beginner's Guide**" is designed to be accessible yet comprehensive, providing you with a clear understanding of the concepts, applications and implications of generative AI. Whether you're a student, professional or just a curious mind, this guide aims to equip you with the knowledge and insight to appreciate and engage with one of the most exciting technological advancements of our time.

Natural Language Processing

What is Natural Language Processing (NLP)?
Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language. The goal of NLP is to enable computers to understand, interpret, and generate human language in a way that is both meaningful and useful.

Key areas of NLP include:

Text Processing: Cleaning and preparing text data for analysis, including tokenization, stemming, and lemmatization.

Syntax and Semantics: Analyzing the grammatical structure of sentences and understanding the meaning of words and sentences.

Named Entity Recognition (NER): Identifying and classifying key elements in text, such as names of people, organizations, locations, dates, etc.

Sentiment Analysis: Determining the sentiment or emotion expressed in a piece of text.
Machine Translation: Automatically translating text from one language to another.

Speech Recognition: Converting spoken language into text.

Text Generation: Creating human-like text based on given prompts or contexts.

How is NLP Useful in Generative AI?

Generative AI refers to AI systems that can generate new content, such as text, images, or music, based on learned patterns from existing data. NLP plays a crucial role in generative AI, particularly in the generation of text and language-based content.

Here’s how NLP is useful in generative AI:

- ***Text Generation***: NLP techniques are used to build models that can generate coherent and contextually relevant text. This includes generating news articles, stories, poems, and even code.

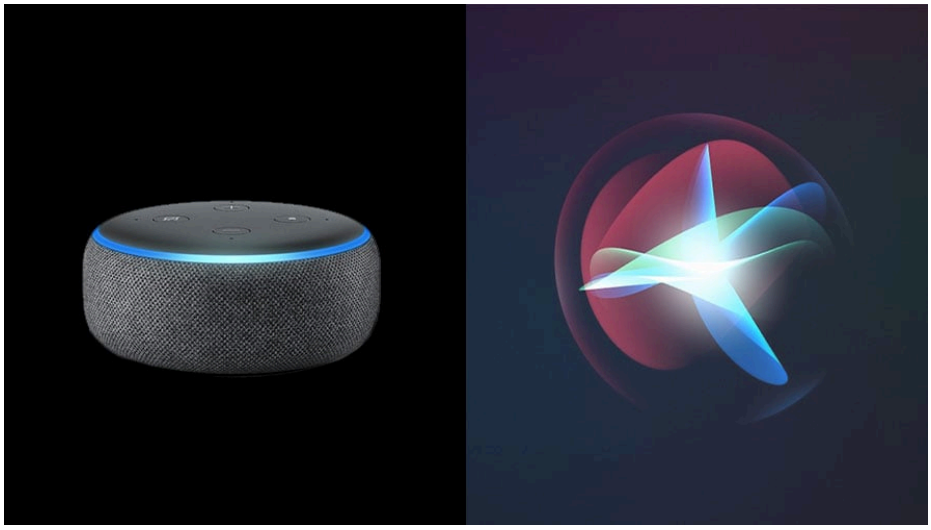
- ***Chatbots and Conversational Agents***: NLP enables the development of sophisticated chatbots and virtual assistants that can understand and respond to user queries in natural language. These systems rely on NLP for intent recognition, response generation, and maintaining context in conversations.
- ***Language Translation***: Generative models powered by NLP can provide accurate and fluent translations between different languages, enhancing communication and accessibility.
- ***Summarization***: NLP techniques can be used to create generative models that can summarize long documents into concise versions, extracting key information while retaining the original meaning.
- ***Content Personalization***: NLP helps generative AI systems to tailor content according to user preferences and behaviors, such as personalized news feeds, recommendations, and marketing content.
- ***Creative Writing Assistance***: NLP-driven generative AI can assist writers by suggesting ideas, generating plot lines, or even drafting content based on specified criteria.

Examples of NLP in Generative AI Tools

- **GPT-3**: Developed by OpenAI, GPT-3 is a state-of-the-art language model that uses NLP to generate human-like text. It can perform tasks such as writing essays, answering questions, and generating code snippets.
- **BERT**: Although primarily used for understanding text, models like BERT can be fine-tuned for text generation tasks, enhancing the capability of generative AI systems.
- **Transformer Models**: These models, which form the basis for many state-of-the-art NLP systems, are crucial for generative AI. They enable the generation of coherent and contextually aware text by understanding and modeling the relationships between words in a sentence.

By leveraging NLP, generative AI systems can create more natural, accurate, and contextually appropriate content, thereby enhancing user experiences across various applications.

AI systems and tools



Believe it or not, most of us have been using AI systems and tools for a very long time in our homes and day-to-day lives. Yes, we are talking about Alexa and Siri, two of the most popular AI assistants on the market. They are both capable of a wide range of tasks including setting alarms, playing music, making calls and controlling smart home devices.

Alexa is a virtual assistant developed by Amazon. It was first released in 2014, and is now available on a wide range of devices including the Amazon Echo, Echo Show and Echo Dot. Alexa can be used to control smart home devices like lights, thermostats and locks. It can also be prompted to play music, get news and weather updates, and set alarms.

It's Apple counterpart, Siri,is a virtual assistant developed by Apple and first released in 2011. Siriis available on a wide range of Apple devices including the iPhone, iPad, Apple Watch and HomePod. It can be used to make calls, send texts, set alarms, get directions and more — and like Alexa, can also be used to control smart home devices.
Both Alexa and Siri are constantly updated with new features and capabilities., continuing to make them powerful tools that make our lives easier and more convenient.

While Alexa and Siri were great advancements in their time and today can answer questions and follow basic commands, generative AI excels at an entirely new pace — particularly in creating fresh content. Think writing poems, composing music or even generating code. Instead of relying on pre-programmed responses, generative AI leverages its understanding of language to produce truly original outputs, making it a powerful tool for creative exploration and innovation.

The AI layers

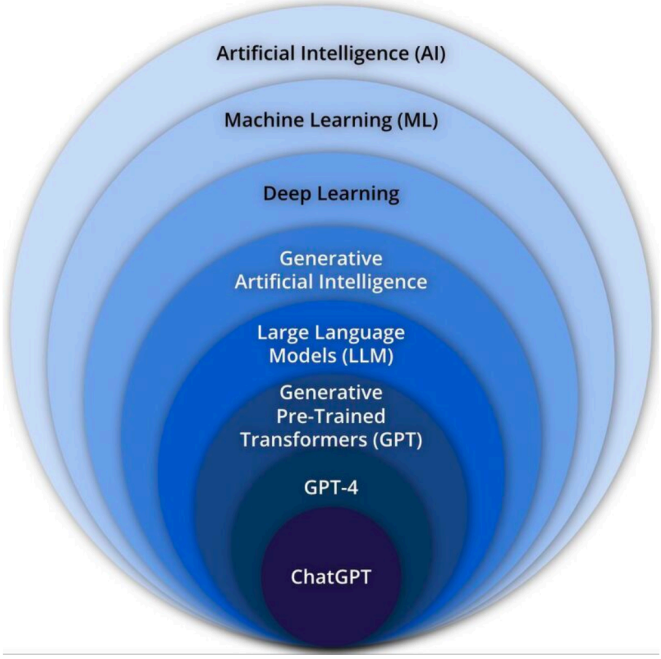


Image credits: Google Cloud Tech

The different layers of AI.

Machine learning (ML)
ML is a type of AI that allows software applications to become more accurate in predicting outcomes without being explicitly programmed to do so. ML algorithms use historical data as input to predict new output values.

Deep learning (DL)
DL is a type of ML that uses artificial neural networks to learn from data. Neural networks are inspired by the structure and function of the human brain, made up of layers of interconnected nodes — each of which performs a simple mathematical operation.

Generative AI
Generative AI is a type of AI that can create new content including text, code, images and music. Generative AI models are trained on large datasets of existing content, learning to identify patterns in data and using those patterns to generate new content.

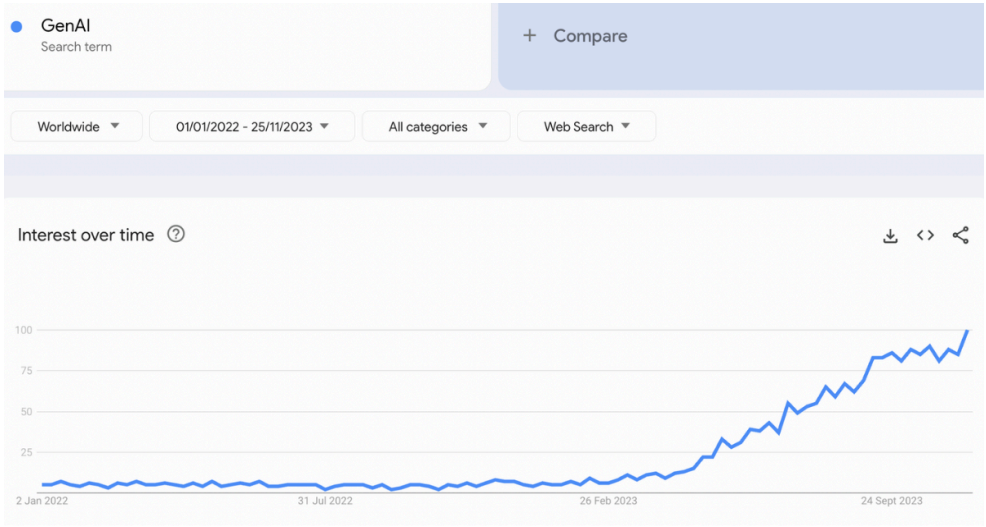
Large Language Models (LLMs)
LLMs are a type of generative AI model trained on massive datasets of text and code. LLMs can generate text, translate languages, write different kinds of creative content and answer your questions in an informative way.

Generative Pre-trained Transformers (GPTs)

GPTs are a type of LLM that use a transformer architecture. Transformers are a neural network architecture well-suited for natural language processing tasks.

GPT-4 and ChatGPT
GPT-4 and ChatGPT are two examples of GPT models. GPT-4 is an LLM developed by OpenAI, while ChatGPT is an LLM (also developed by OpenAI) that is specifically designed for chatbot applications.

The rise of generative AI



Generative AI, a subset of artificial intelligence focused on creating entirely new content like text, code, images and music, is experiencing a stratospheric rise. This advancement is fueled by powerful algorithms trained on massive datasets, allowing them to learn complex patterns and generate outputs indistinguishable from human-made creations.

Previously, AI systems largely relied on pre-programmed responses and instructions, limiting their ability to innovate and truly be creative. But generative AI breaks this mold. Not only can generative AI models answer questions and follow commands, but they can also imagine and produce entirely new concepts — giving way to exciting possibilities and advancements across fields and industries.
From designing personalized learning experiences in education to developing cutting-edge medical treatments in healthcare, the potential applications of generative AI are vast and evolving. As this technology continues to mature, we can expect to see its impact transform every aspect of our lives, ushering in a new era of creativity and innovation.

Just imagine, AI-powered tools that can:

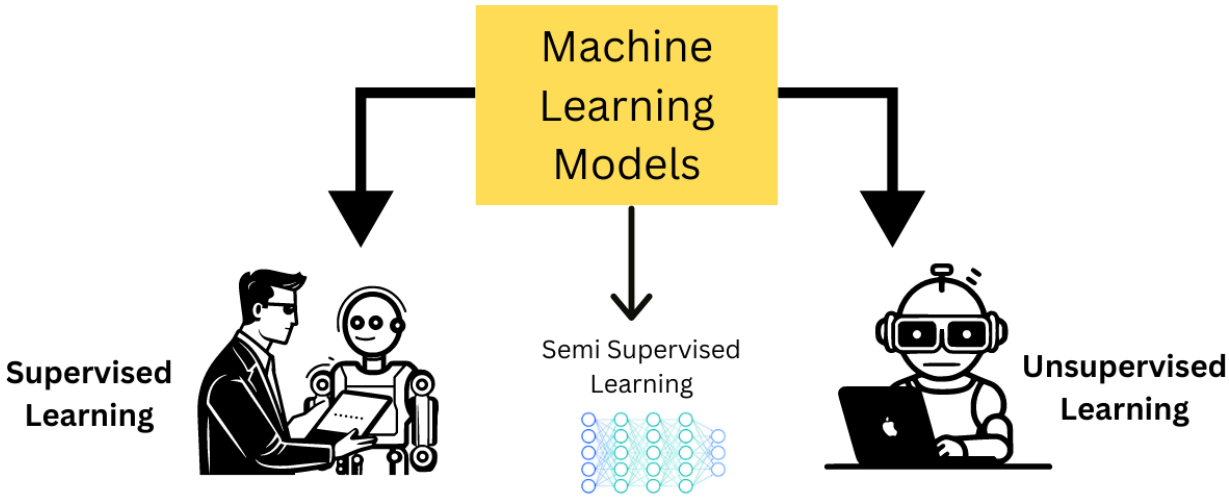
- Write compelling marketing copy and engaging blog posts
- Compose music tailored to your specific preferences
- Generate realistic and detailed images for any purpose
- Develop innovative software applications with previously unimaginable features

However, the rise of generative AI is not without its challenges. Concerns around ethical implications, potential biases and job displacement are all important considerations. As we embrace this powerful technology, it is crucial to address these concerns and develop responsible frameworks for its development and use.

By harnessing the potential of generative AI while mitigating its risks, we can unlock a future brimming with creativity, innovation and progress.

Machine learning

Machine learning (ML) is the most common foundation for advanced AI systems. It's a branch of artificial intelligence that allows computers to learn and improve their performance without explicit programming. Unlike traditional programming where you provide instructions to a computer, ML algorithms can learn from data and make predictions or decisions on their own.



Machine learning thrives on different techniques for extracting knowledge and patterns from data. Here are three main approaches:

1. Supervised learning

Imagine a teacher showing her students pictures of cats and dogs, labeling each one. Supervised learning follows a similar approach. It uses labeled data, where each input has a known and desired output. The algorithm analyzes this data to learn the relationship between inputs and outputs, allowing it to predict the output for new, unseen data.

Some real-world examples include:

- **Spam filtering**. Identifying spam emails based on features learned from labeled examples.
- **Image recognition.** Classifying images as containing a specific subject (like cats or dogs) based on labeled training data.
- **Linear regression.** Predicting house prices based on features like size and location.

2. Unsupervised learning

Unsupervised learning is about discovery. Think of a child exploring a new toy box, discovering its features and functionalities on their own. Mirroring this exploration, unsupervised learning uses unlabeled data where the algorithm finds patterns and relationships on its own, without any predetermined outcome. This is useful for tasks like:

- **Customer segmentation.** Grouping customers with similar characteristics for targeted marketing campaigns.
- **Anomaly detection.** Identifying unusual patterns in data, like detecting fraudulent transactions.
- **Dimensionality reduction.** Compressing high-dimensional data into a lower-dimensional space for efficient analysis.

3. Semi-supervised learning

Semi-supervised learning is a powerful machine learning technique that sits between supervised and unsupervised learning. It combines the strengths of both approaches, utilizing a small amount of labeled data in combination with a large amount of unlabeled data to train a model.

Here's how it works,
Labeled data provides the basic understanding. Imagine a teacher showing her students a few pictures of cats and dogs, labeling them accordingly. This initial labeled data gives the algorithm a basic understanding of the task at hand.

Unlabeled data reinforces and expands knowledge. Then, the students explore a pile of unlabeled pictures, using their existing knowledge to identify more cats and dogs. Similarly, the unlabeled data in semi-supervised learning helps the algorithm refine its understanding and generalize its knowledge beyond the initial, limited labeled examples.

Here are some common applications of semi-supervised learning:

- **Image classification.** Classifying large image datasets with minimal labeled examples.
- **Text classification.** Categorizing documents and emails with limited labeled data.
- **Anomaly detection.** Identifying unusual patterns in large datasets — even with limited labeled examples of anomalies.
- **Recommendation systems.** Recommending products or services to users based on their past behavior and the behavior of similar users, even with limited explicit feedback.

Each technique has its strengths and weaknesses, and the choice depends on the specific task and data availability.

Deep learning

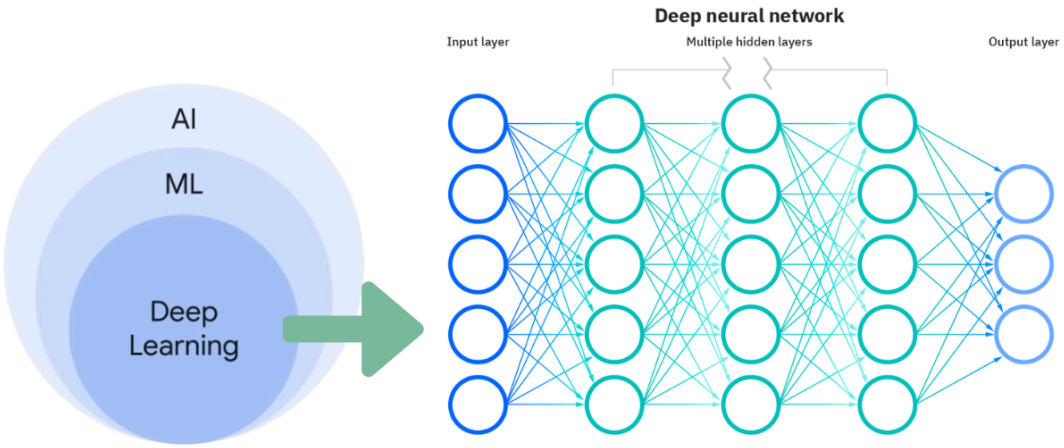


Image credits: Google Cloud Tech

Let’s talk about the next layer in the AI world, deep learning.

Deep learning, a subfield of artificial intelligence, has revolutionized numerous fields with its ability to learn complex patterns and relationships from data. This remarkable capability is fueled by the power of artificial neural networks (ANNs).

ANNs are loosely inspired by the structure and function of the human brain. They consist of interconnected nodes — called neurons — arranged in layers. Each neuron performs a simple mathematical operation on its input, passing the output to other neurons in the next layer.

Here's how ANNs work:

- **Data enters the input layer.** The input layer receives raw data, like an image or text string.
- **Information flows through hidden layers.** Each neuron in a hidden layer performs a calculation on its inputs, passing the result to the next layer. This process allows the network to learn complex representations of the data.
- **Output is generated.** The output layer produces the final result, such as classification or a prediction.

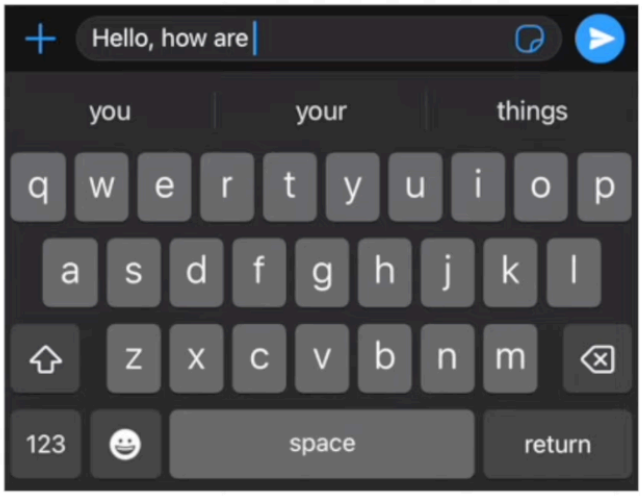
The network learns by adjusting the weights of the connections between neurons. These weights are updated based on a process called backpropagation, which works to minimize error between the network's output and the desired outcome.

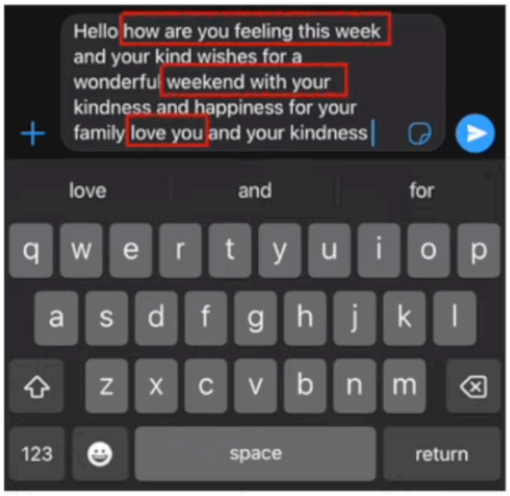
Here are some popular deep learning architectures:

- **Convolutional Neural Networks (CNNs).** CNNs are particularly well-suited for image and video recognition tasks.

- **Recurrent Neural Networks (RNNs).** RNNs are effective for processing sequential data, including natural language and time series data.
- **Generative Adversarial Networks (GANs).** GANs are used to generate new data includingimages, music and text.

Despite their impressive capabilities, earlier generations of neural networks suffered from a critical limitation: lack of context. They often operated in isolation, analyzing data without considering its broader context or relationship to other information.





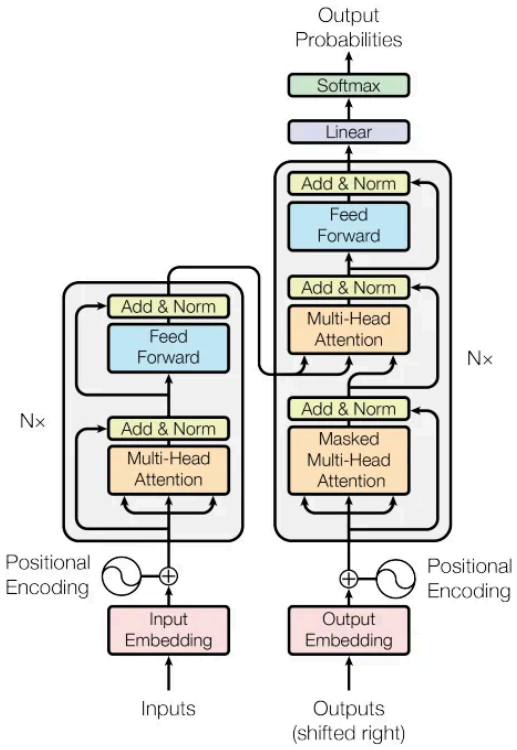
You can see what this looks like with a quick experiment. Open the text message app on your phone, and start typing. Now instead of crafting your own sentence, try typing the words your phone recommends until you create a sentence. It probably doesn't make much sense, right? This is because there is no self-attention mechanism embedded which leads to a lack of context — the neural network model is simply just trying to guess the next word.

This lack of context led to the creation of an in-built self-attention mechanism. In 2017, Google researchers introduced the Transformer model — taking the AI world by storm.

Transformer model

"The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely." — "Attention Is All You Need," Google

The Transformer model emerged as a response to several limitations of earlier deep learning models, particularly in the field of Natural Language Processing (NLP).



The Transformer model architecture is built on the concept of attention, allowing it to focus on relevant parts of the input sequence when making predictions. This enables the model to capture long-range dependencies and achieve superior performance on various tasks.

Here's a breakdown of the key components:

1. Encoder

This takes the input sequence and encodes it into a series of vector representations.
Each layer in the encoder uses self-attention mechanisms to attend to relevant parts of the input sequence. This allows the model to capture long-range dependencies and understand the input context.

2. Decoder

This uses the representation from the encoder to generate the output sequence.
Each layer in the decoder uses self-attention to attend to relevant parts of the encoded representation, and also uses encoder-decoder attention to attend to relevant parts of the decoder's own output. This allows the model to generate outputs that are coherent and consistent with the input.

3. Multi-head attention

This allows the model to attend to different aspects of the input data simultaneously.

Each attention head learns to focus on different features or patterns in the data, helping the model capture a deeper understanding of the input and generate ing more accurate outputs.

4. Positional encoding
Since the Transformer model does not process data sequentially, it needs to be provided with information about the order of elements. Positional encoding helps the model understand the relative positions of elements in the input sequence.

5. Residual connections and layer normalization
These techniques help improve the stability and training efficiency of the model.

The Transformer model's architecture has several advantages:

- **Parallel processing**. Allows for efficient training and inference on GPUs.
- **Long-range dependencies**. Captures relationships between elements far apart in the input sequence.
- **Context awareness**. Understands the broader context of the input data.

These advantages have made the Transformer model a foundational architecture for many modern deep learning models — and have significantly advanced the field of AI research.

The Transformer addressed these limitations by introducing the concept of attention:

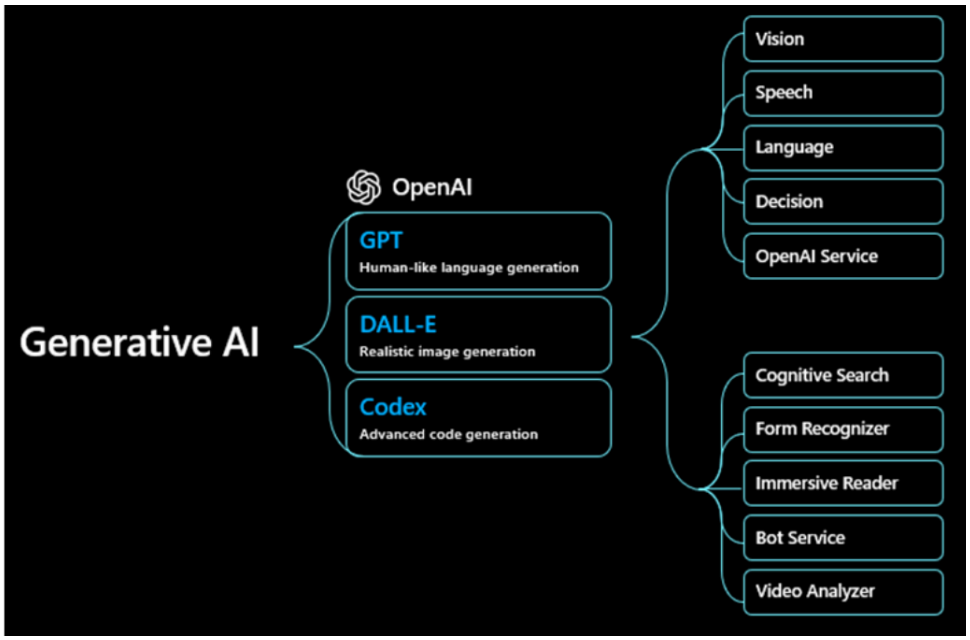
- **Self-attention.** This mechanism allows the model to focus on relevant parts of the input sequence when making predictions, enabling it to capture long-range dependencies.
- **Encoder-decoder architecture.** This architecture separates the encoding and decoding stages, allowing for more efficient parallel processing.
- **Positional encoding.** This technique helps the model understand the order of elements in a sequence, even without explicit RNN-style processing.
- **Multi-head attention.** This extension allows the model to attend to different aspects of the input data simultaneously, further enhancing its understanding of context.

As a result of addressing these limitations, the Transformer model achieved significant improvements in various NLP tasks, including machine translation, text summarization, and question answering. It has become a foundational architecture for many modern NLP models and has significantly advanced the field of AI research.

Introduction to generative AI

The advancements with the Transformer model paved the way for generative AI by providing the foundation for models like GPT-3 and ChatGPT. These models leverage the Transformer's capabilities to generate human-quality text, translate languages with nuanced understanding, write diverse creative content and answer questions in an informative and engaging way.

Generative AI is a subfield of artificial intelligence that focuses on creating entirely new content like text, images, music and code. Unlike traditional AI, which primarily focuses on analyzing and understanding existing data, generative AI models are trained on massive datasets to learn the underlying patterns and structures of the content they aim to create.



These models utilize this knowledge to generate new and original outputs that are statistically similar to the data they were trained on. This allows them to perform a variety of impressive tasks, including:

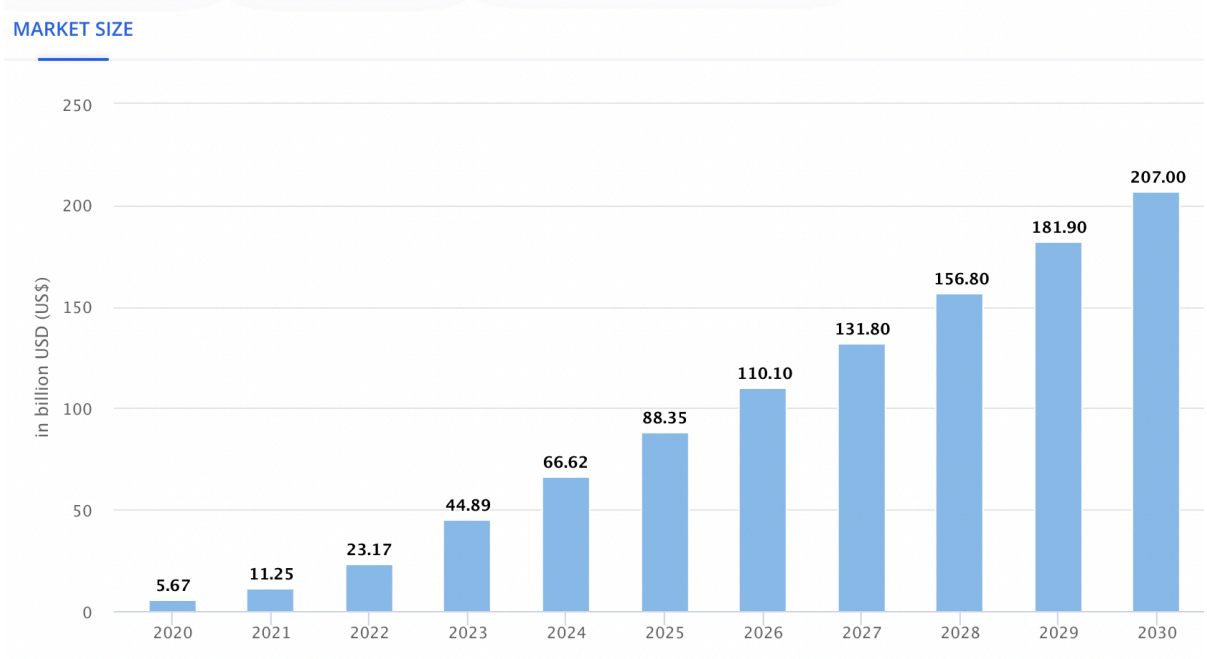
- **Writing creative content.** Generating poems, scripts, musical pieces and other forms of artistic expression.
- **Translating languages.** Translating text between languages while preserving nuance and context.
- **Developing software.** Automatically generating code snippets — and even entire applications.
- **Designing products.** Generating realistic and aesthetically pleasing product designs.
- **Personalizing experiences.**: Creating custom content and recommendations tailored to individual users.

The possibilities of generative AI are vast and constantly evolving, with new applications emerging across various industries. As this technology matures, we can expect its impact to revolutionize how we create, learn and interact with the world around us.

Generative AI: Some fascinating metrics

According to Statisa, the gen AI market size is expected to show an annual growth rate (CAGR 2023-2030) of 24.40%, resulting in a market volume of $207 billion by 2030. (as of Aug 2023)

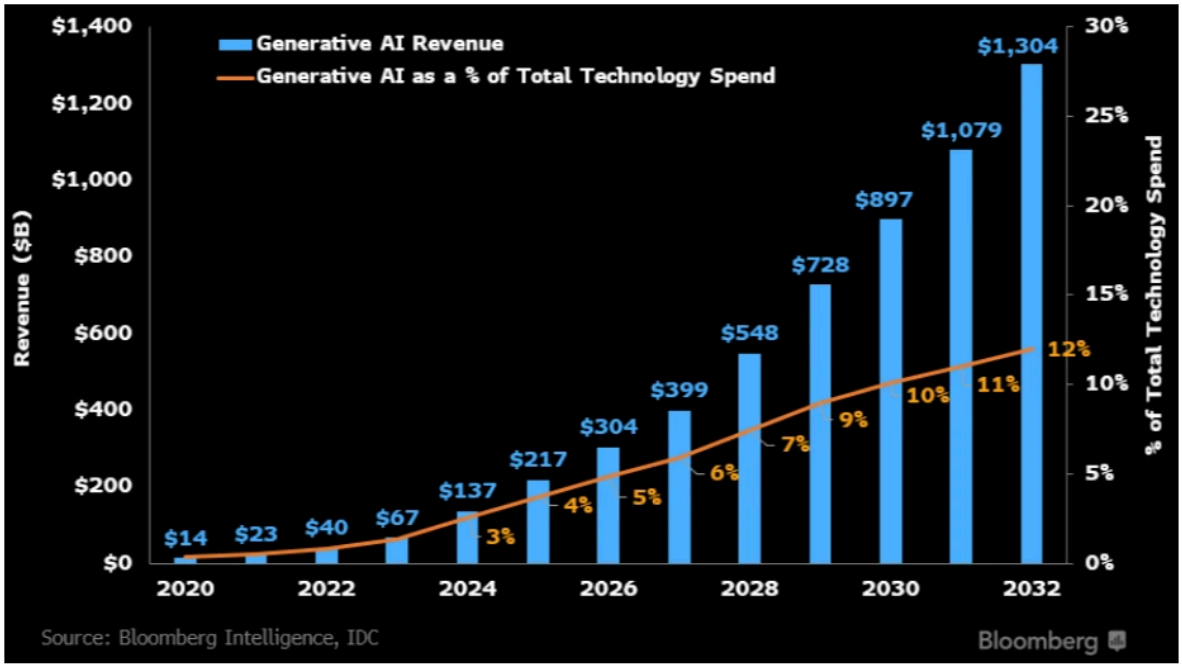
In global comparison, the largest market size will be in the United States ($16.14 billion in 2023).

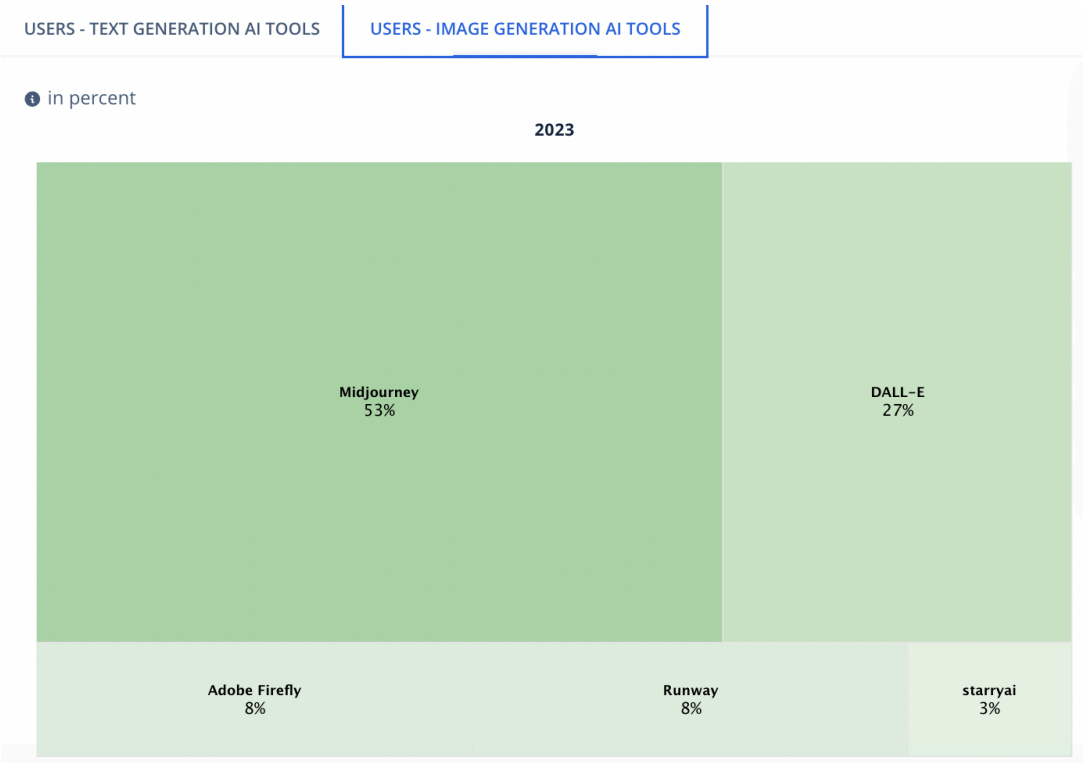


As of November 2023, ChatGPT is the most used text generation Ggen AI tool.

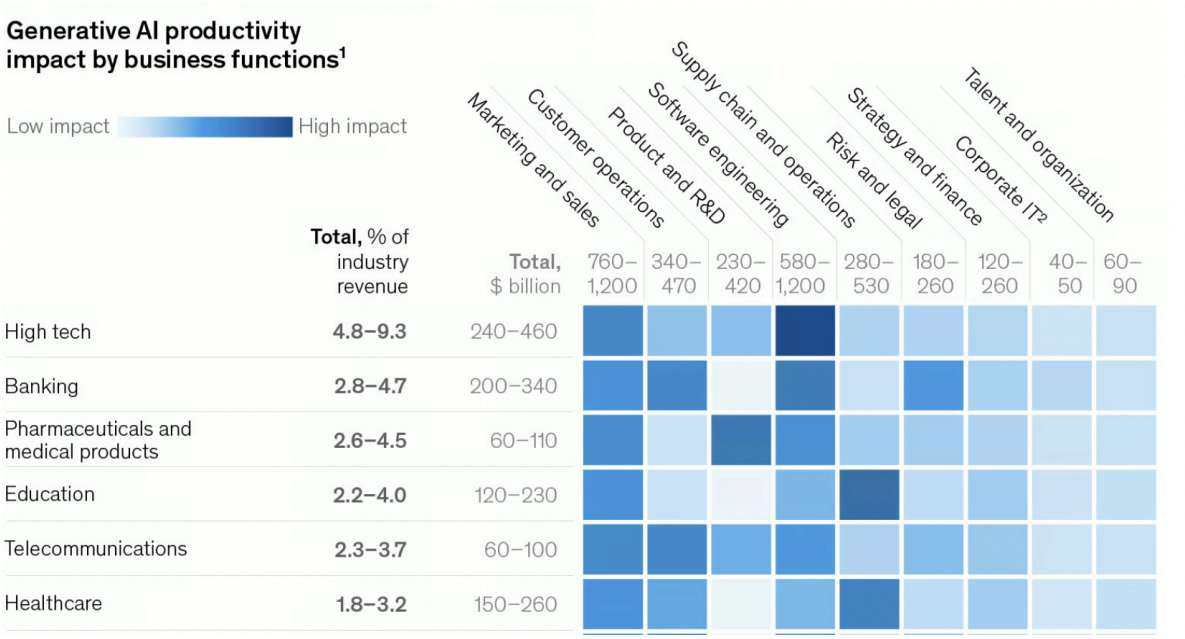
Similarly, Midjourney is the most used image generation gen AI tool (also as of November 2023).

Bloomberg estimates generative AI is poised to become a $1.3 trillion market by 2032.

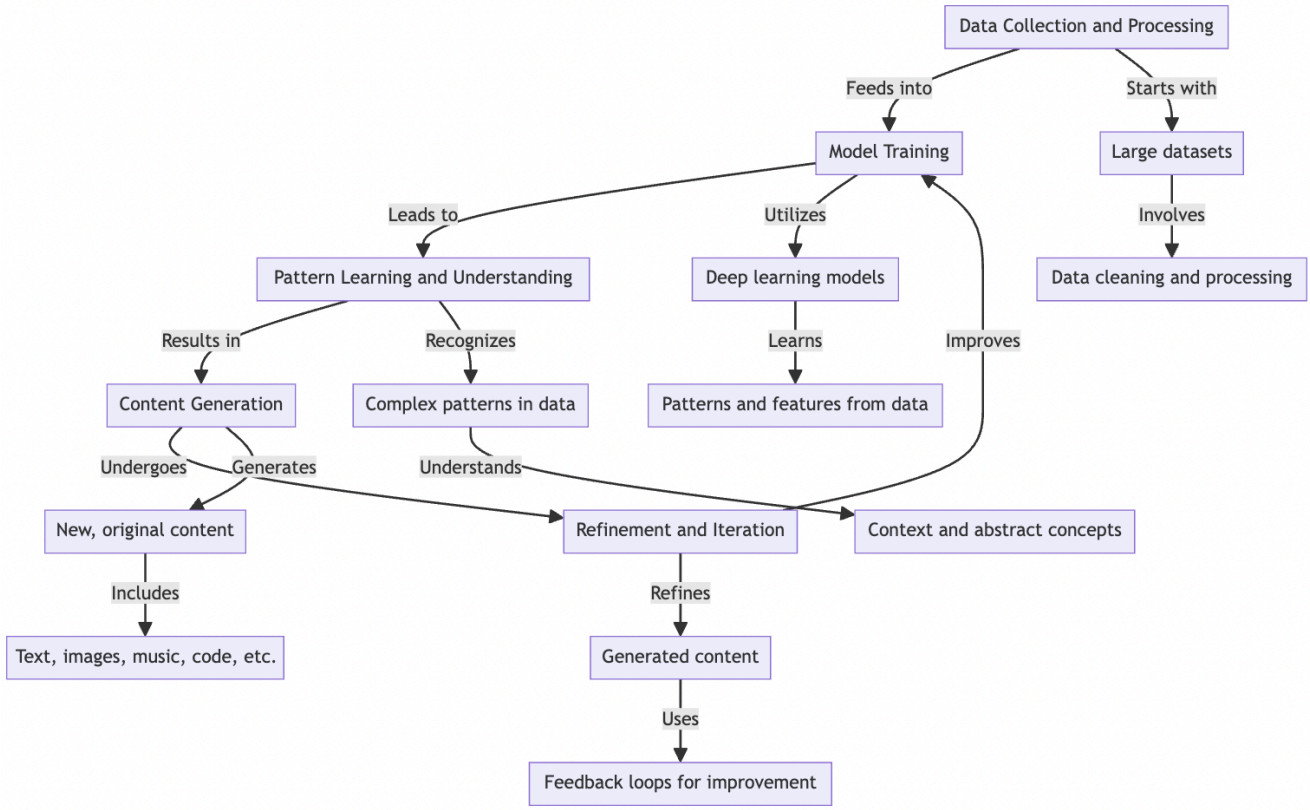




McKinsey & Company presented generative AI's impact on various business functions across industries — finding that that high-tech, banking, pharma, education and telecommunications are the top five areas where generative AI will have the most impact.



How generative AI works



The preceding diagram represents a rough overview how generative AI works, starting from initial stages to final output and refinement.

- **Data collection and processing.** The process begins with the collection and processing of large datasets. This stage is crucial since it forms the foundation for the AI model to learn and generate content. It involves gathering extensive data, cleaning and processing it to make it suitable for training the AI model.
- **Model training.** The next stage is model training, where deep learning models are utilized. These models learn patterns and features from the processed data. This is also the stage where the AI begins to understand the data structure, patterns and nuances.
- **Pattern learning and understanding.** Here, the AI model recognizes complex patterns in the data and starts understanding context and abstract concepts. This is a critical phase where the AI develops the capability to generate meaningful and coherent content.
- **Content generation.** Based on the learned patterns and understanding, the AI then generates new, original content. This content can vary widely — including text, images, music, code, etc. — depending on the AI's training and intended application.

- **Refinement and iteration.** The final stage involves refining the generated content and using feedback loops for improvement. This iterative process ensures the output is high quality, and meets desired standards or objectives.

The following diagram effectively illustrates this flow, showing how each stage leads to the next, with feedback loops indicating the continuous improvement and learning process inherent in generative AI systems.

ML model vs. gen AI model

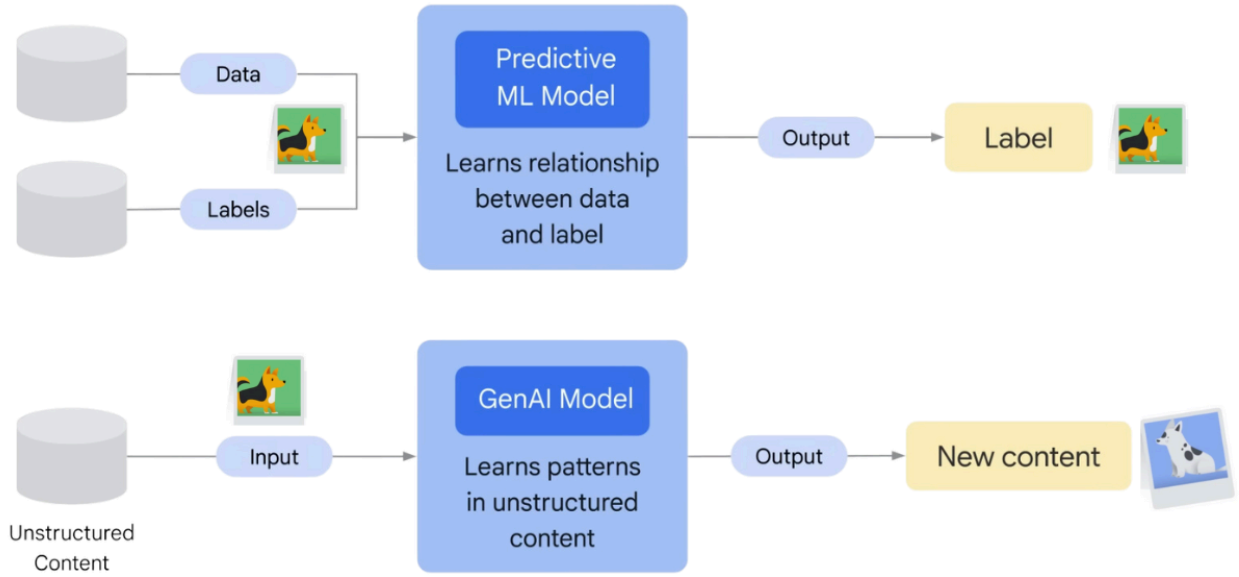


Image credits: Google Cloud Tech

Generative AI models, like GPT-4 or DALL-E, differ significantly from traditional machine learning (ML) models in their objectives, training approaches, complexity and applications. Traditional ML models focus on predicting outcomes by learning the relationship between input data and output labels, making them ideal for tasks like classification and regression.

These models often require structured datasets with clear labels, using techniques like supervised learning. In contrast, generative AI models are designed to create new content by learning patterns in unstructured data. They are trained on vast datasets using unsupervised or self-supervised learning methods, enabling them to generate text, images and more.

These models typically use complex architectures — like deep neural networks — and are particularly suited for creative tasks like writing, image generation and music composition. While traditional ML models excel in structured data analysis, generative AI models stand out in their ability to produce novel and diverse content.

Breaking down key generative AI feature differences

Feature	Machine Learning	Generative AI
Purpose	Analyze existing data to make predictions or decisions	Create entirely new content
Learning Approach	Primarily relies on labeled data	Utilizes unlabeled or partially labeled data
Model Architecture	Task-specific algorithms (linear regression, decision trees, etc.)	Deep learning architectures (GANs, Transformers)
Output	Predictions, classifications, decisions	New text, code, images, music
Examples	Spam filtering, image recognition, credit card fraud detection	Text generation (GPT-3, ChatGPT), image generation (DALL-E 2), music composition (Jukebox)
Strengths	High accuracy on specific tasks, interpretable outputs	Creativity, personalization, efficiency
Weaknesses	Limited to existing data, requires extensive feature engineering	Potential for bias, less transparent decision-making
Applications	Healthcare, finance, marketing, manufacturing	Content creation, design, education, entertainment
Overall Goal	Gain insights from data	Generate new and innovative content

Journey from traditional programming to neural networks to generative AI

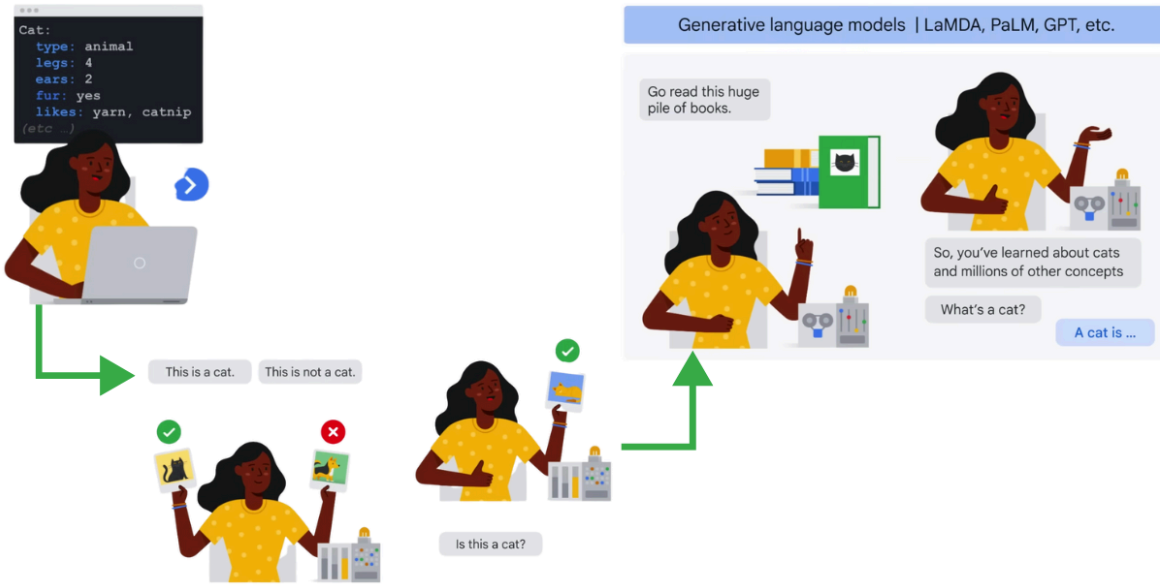


Image credits: Google Cloud Tech

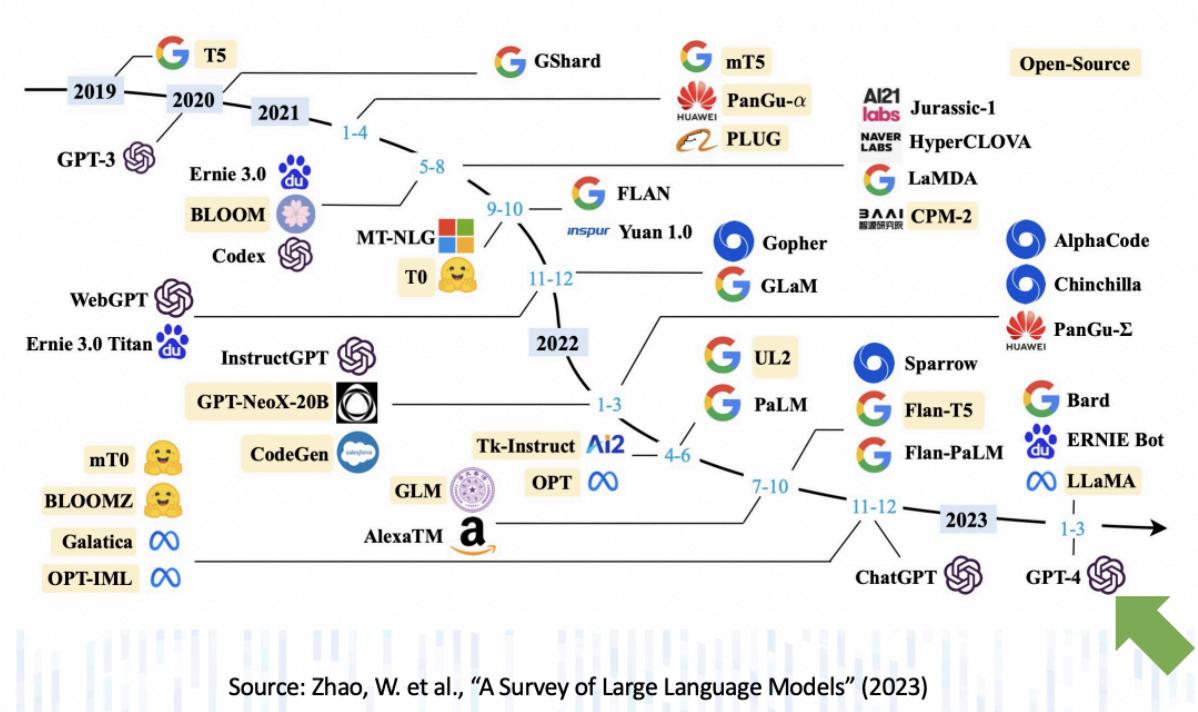
The evolution of artificial intelligence and programming methodologies has been remarkable, showcasing a significant transition from traditional programming to the advanced realms of neural networks and generative AI models. In the era of traditional programming, the approach was primarily rule-based, where programmers had to hard-code specific rules to enable object identification and differentiation. This method, while effective in certain scenarios, was limited by the necessity of explicitly programming each potential outcome or behavior.

The advent of neural networks marked a pivotal shift in this approach. Inspired by the structure and function of the human brain, neural networks allowed for a more dynamic, flexible way of processing information. Instead of relying on hard-coded rules, neural networks learn from a dataset, adapting their internal parameters to recognize patterns and make predictions. For example, when fed with images of cats and dogs, a neural network learns to differentiate between them, identifying whether a given picture was of a cat or a dog.

In this era of generative AI models, this concept is taken evenfurther. These models don't just recognize or categorize data but are capable of creating entirely new content based on the data they have been trained on — representing a significant leap in AI capabilities. Generative AI can produce original text, images, music and other forms of content, mimicking human creativity at a scale and speed unattainable for humans. From entertainment to education, this advancement opens up a myriad of possibilities across various sectors, marking a new frontier in the evolution of AI and computer programming.

Large Language Models: Powering generative AI

Large Language Models (LLMs), represent a significant advancement in the field of artificial intelligence, particularly in understanding and generating human-like text.



Source: Zhao, W. et al., "A Survey of Large Language Models" (2023)

LLMs are a specific type of deep learning model trained on massive datasets of text and code. These models possess extensive knowledge and understanding of language, allowing them to perform various tasks, including:

- **Text generation.** Create realistic and engaging text like poems, code snippets, scripts and news articles.
- **Translation.** Convert text from one language to another — accurately and fluently.
- **Question answering.** Provide informative, comprehensive answers to open-ended and challenging questions.
- **Summarization.** Condense large amounts of text into concise and informative summaries.

How do LLMs work?

LLMs typically utilize Transformer-based architectures we talked about before, relying on the concept of attention. This allows the model to focus on relevant parts of the input text when making predictions and generating outputs.

The training process involves feeding the LLM massive datasets of text and code. This data helps the model learn complex relationships between words and phrases, ultimately enabling it to understand and manipulate language in sophisticated ways.

Different Large Language Models

Generative AI models have become increasingly sophisticated and diverse, each with its unique capabilities and applications. Some notable examples include:

- **BARD (Bi-directional and Auto-Regressive Transformers)**
  Gemini (formerly Bard) is a Google AI chatbot that uses natural language processing to chat naturally and answer your questions. It can enhance Google searches and be integrated into various platforms, providing realistic language interactions.

- **ChatGPT**
  Developed by OpenAI, ChatGPT is a variant of the GPT (Generative Pre-trained Transformer) model, specifically fine-tuned for conversational responses. It's designed to generate human-like text based on the input it receives, making it useful for a wide range of applications including customer service and content creation. ChatGPT can answer questions, simulate dialogues and even write creative content.

- **DALL-E**
  Also from OpenAI, DALL-E is a neural network-based image generation model. It's capable of creating images from textual descriptions, showcasing an impressive ability to understand and visualize concepts from a simple text prompt. For example, DALL-E can generate images of "a two-headed flamingo" or "a teddy bear playing a guitar," even though these scenes are unlikely to be found in the training data.

- **MidJourney**
  Midjourney is a generative AI tool that creates images from text descriptions, or prompts. It's a closed-source, self-funded tool that uses language and diffusion models to create lifelike images.

- **Stable Diffusion:**

Stable Diffusion is a deep learning model that uses diffusion processes to create high-quality images from input images. It's a text-to-image model that can transform a text prompt into a high-resolution image.

The following is a more comprehensive list of LLMs:

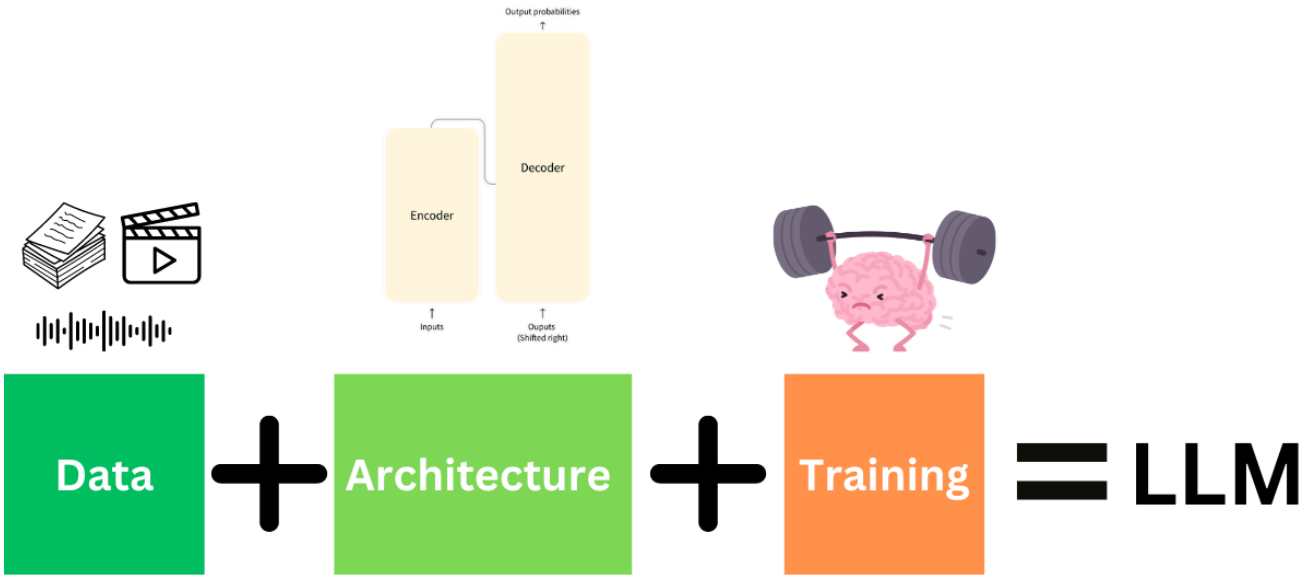
LLMs_List

Name	Developer	Year
BERT	Google	2018
XLNet	Google	2019
GPT-2	OpenAI	2019
GPT-3	OpenAI	2020
GPT-Neo	EleutherAI	March 2021
GPT-J	EleutherAI	June 2021
Megatron-Turing NLG	Microsoft and Nvidia	October 2021
Ernie 3.0 Titan	Baidu	December 2021
Claude	Anthropic	December 2021
GLaM (Generalist Language Model)	Google	December 2021
Gopher	DeepMind	December 2021
LaMDA (Language Models for Dialog Applications)	Google	January 2022
GPT-NeoX	EleutherAI	February 2022
Chinchilla	DeepMind	March 2022
PaLM (Pathways Language Model)	Google	April 2022
OPT (Open Pretrained Transformer)	Meta	May 2022
YaLM 100B	Yandex	June 2022
Minerva	Google	June 2022
BLOOM	Large collaboration led by Hugging Face	July 2022
Galactica	Meta	November 2022
AlexaTM (Teacher Models)	Amazon	November 2022
LLaMA (Large Language Model Meta AI)	Meta	February 2023
GPT-4	OpenAI	March 2023
Cerebras-GPT	Cerebras	March 2023
Falcon	Technology Innovation Institute	March 2023
BloombergGPT	Bloomberg L.P.	March 2023
PanGu-Σ	Huawei	March 2023
OpenAssistant	LAION	March 2023
PaLM 2	Google	May 2023

Each of these models represents a different facet of generative AI, showcasing the versatility and potential of these technologies. From text to images, these models push the boundaries of what's possible with AI, enabling new forms of creativity and problem solving.

Components of an LLM

LLMs are built upon three foundational pillars: **data**, **architecture** and **training**.

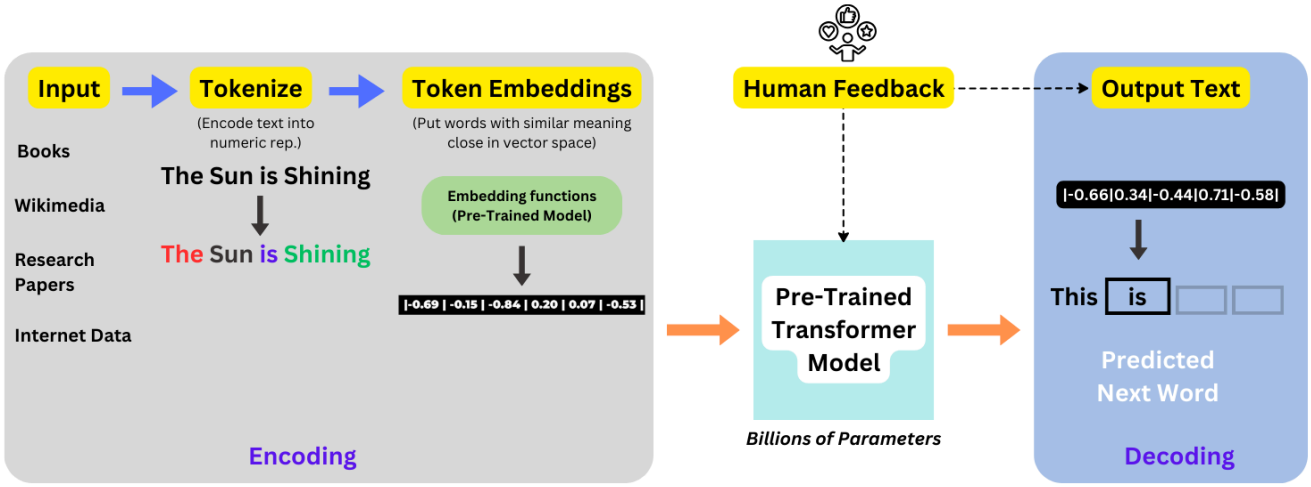


- **Data.** This is the cornerstone of any LLM. The quality, diversity and size of the dataset determine the model's capabilities and limitations. The data usually consists of a vast array of text from books, websites, articles and other written sources. This extensive collection helps the model learn various linguistic patterns, styles and breadth of human knowledge.
- **Architecture.** This refers to the underlying structure of the model, often based on a Transformer architecture. Transformers are a type of neural network particularly well-suited for processing sequential data like text — using mechanisms like attention to weigh the importance of different parts of the input data. This architecture enables the model to understand and generate coherent, contextually relevant text.
- **Training.** The final piece is the training process, where the model is taught using collected data. During training, the model iteratively adjusts its internal parameters to minimize prediction errors. This is often done through supervised learning, where the model is given input data along with the correct output and learns to produce the desired output on its own. The training process is not only about learning the language but also about understanding nuances, context and the ability to generate creative or novel responses.

Each of these components plays a crucial role in shaping the capabilities and performance of a Large Language Model. The harmonious integration of these elements allows the model to understand and generate human-like text, answering questions, writing stories, translating languages and much more.

How do LLMs learn?

It is very important to understand how these LLMs are trained, so let's take a closer look at the training process:



- **Input.** The model begins with a vast and diverse corpus of text data sourced from books, Wikimedia, research papers and various internet sites. This data provides the raw material for the model to learn language patterns.
- **Tokenize.** In this step the text is tokenized, meaning it is divided into smaller pieces — often words or subwords. Tokenization is essential for the model to process and understand individual elements of the input text.
- **Token embeddings.** Each token is transformed into a numerical representation known as a token embedding. These embeddings are vectors that encode the meaning of the tokens in a way that words with similar meanings are closer together in the vector space.
- **Encoding.** The embeddings are then passed through the encoding layers of the transformer model. These layers process the embeddings to understand the context of each word within the sentence. The model does this by adjusting the embeddings in a way that incorporates information from surrounding words using self-attention mechanisms.
- **Pre-trained Transformer model.**: The main architecture of the model is a Transformer, which has been pre-trained on the input data. It has learned to predict parts of the text from other parts, adjusting its internal parameters to minimize the prediction error. This model has layers of attention, feedforward neural networks and a large number of parameters to capture the complexities of language.

- **Human feedback.** This is an optional but important step where humans can provide feedback on the model's outputs, further fine-tuning the model's performance. The feedback loop can help align the model's outputs with human expectations and standards.
- **Output text.** When generating text, the model uses the pre-trained and possibly fine-tuned parameters to predict the next word in a sequence. This involves the decoding process.
- **Decoding.** The decoding step is where the model converts the processed embeddings back into human-readable text. After predicting the next word, the model decodes this prediction from its numerical representation to a word. The process of decoding often involves selecting the word with the highest probability from the model's output distribution. This prediction can be the final output or it can serve as an additional input token for the model to predict subsequent words, allowing the model to generate longer stretches of text.

This cycle of encoding and decoding, informed by both the original training data and ongoing human feedback, enables the model to produce text that is contextually relevant and syntactically correct. Ultimately, this can be used in a variety of applications including conversational AI, content creation and more.

Building an LLM application

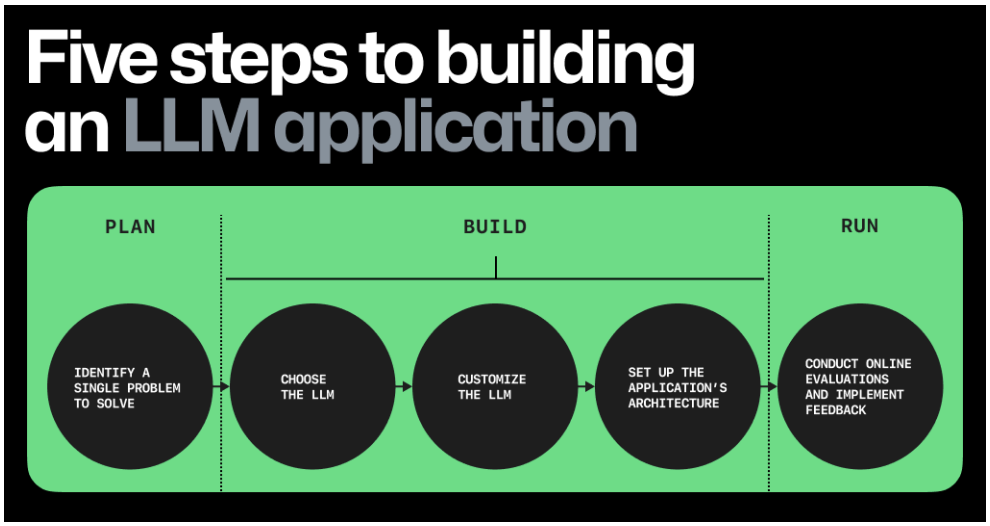


Image credits: ***GitHub***

Here are the high-level steps you need to know to build an LLM application:

1. Focus on a single problem first. The key? Find a problem that's the right size: one that's focused enough so you can quickly iterate and make progress, but also big enough so that the right solution will wow users.

2. Choose the right LLM. You're saving costs by building an LLM app with a pre-trained model, but how do you pick the right one? Here are some factors to consider:

- Licensing. If you hope to eventually sell your LLM app, you'll need to use a model that has an API licensed for commercial use. To get you started on your search, here's a community-sourced list of open LLMs that are licensed for commercial use.

- Model size. The size of LLMs can range from seven to 175 billion parameters — and some, like Ada, are even as small as 350 million parameters. Most LLMs (at the time of writing) range in size from 7-13 billion parameters.

3. Customize the LLM. When you train an LLM, you're building the scaffolding and neural networks to enable deep learning. When you customize a pre-trained LLM you're adapting the LLM to specific tasks, like generating text around a specific topic or in a particular style.

4. Set up the app architecture. The different components you'll need to set up your LLM app. Can be broadly divided into three categories - User input , Input enrichment & prompt construction tools and efficient and responsible AI tooling.

5. Conduct online evaluations of your app. These are considered "online" evaluations because they assess the LLM's performance during user interaction.

See more on the architecture of today's LLM applications.

LLMs use cases

LLMs have transcended their initial research roots and are now finding practical applications across various industries. These powerful models, trained on massive datasets of text and code, possess exceptional capabilities in language understanding, generation and manipulation. Here are some of the exciting use cases of LLMs:

Content creation

- **Personalized news articles.** LLMs can generate personalized news stories tailored to individual interests and reading preferences.
- **Marketing copywriting.** LLMs can craft engaging and persuasive marketing copy for various products and services.
- **Creative writing.** LLMs can assist writers by generating creative text formats like poems, scripts and musical pieces.
- **Code generation.** LLMs can generate code snippets and even entire programs, automating repetitive tasks and assisting developers.

Education

- **Interactive learning platforms.** LLMs can personalize learning experiences by adapting content and questions to individual student needs.
- **Automatic grading and feedback.** LLMs can analyze student work and provide automated feedback, saving time and effort for educators.
- **Virtual tutors and companions.** LLMs can provide personalized learning support and act as virtual tutors, answering questions and offering guidance.

Customer service and support

- **Conversational AI chatbots.** LLMs can be used to create chatbots that answer customer questions, resolve issues and provide 24/7 support.
- **Automated email and message response.** LLMs can draft personalized and natural-sounding email responses or automated messages, saving time and resources.
- **Sentiment analysis and customer insights.** LLMs can analyze customer feedback and social media data to identify trends, and gain valuable insights.

Research and development.

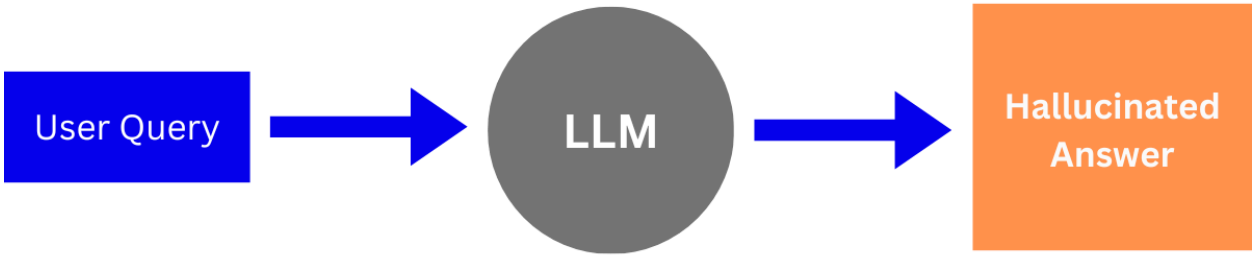
- **Scientific discovery.** LLMs can analyze massive amounts of research data to identify patterns and accelerate scientific discoveries.
- **Medical diagnosis and treatment.** LLMs can analyze medical records and images to assist in diagnosis, recommending personalized treatment options.
- **Literature review and analysis.** LLMs can quickly analyze vast amounts of literature, summarizing key findings and highlighting relevant passages.

Entertainment and media

- **Personalized entertainment experiences.** LLMs can personalize entertainment recommendations, suggesting new music and movies based on individual preferences.
- **Game development and storytelling.** LLMs can create interactive narratives and dialogue for video games, enhancing player immersion and engagement.
- **Content moderation and filtering**. LLMs can be used to detect and filter harmful content online, making the digital space safer for all.

These are just a few examples of diverse LLMs use cases. As the technology continues to evolve and become more accessible, we can expect even more innovative and impactful applications across various industries. With their ability to understand, generate and manipulate language, LLMs are poised to revolutionize the way we interact with information — and the way we learn, create and work.

Limitations of LLMs



While LLMs offer impressive capabilities in language understanding and generation, they also have limitations that can hinder their effectiveness in real-world applications. These limitations include:

- **Missing context.** LLMs may struggle to understand the broader context of a given task, leading to inaccurate or irrelevant outputs.
- **Non-tailored outputs.** LLMs may generate generic responses that don't address the specific needs or requirements of the user or task.
- **Limited specialized vocabulary.** LLMs trained on general datasets may lack the specialized vocabulary needed for specific domains or tasks.
- **Hallucination.** LLMs can sometimes invent information or generate outputs that are biased, inaccurate or misleading.

LLM Hallucinations

Here is a Hallucination Leaderboard as per Vectara's Hallucination Evaluation Model. GPT-4 and GPT-4 Turbo came out on top with the highest accuracy rate (97%) and lowest hallucination rate (3%) of any of the tested models.

Model	Accuracy	Hallucination Rate	Answer Rate	Average Summary Length (Words)
GPT 4	97.0 %	3.0 %	100.0 %	81.1
GPT 4 Turbo	97.0 %	3.0 %	100.0 %	94.3
GPT 3.5 Turbo	96.5 %	3.5 %	99.6 %	84.1
Llama 2 70B	94.9 %	5.1 %	99.9 %	84.9
Llama 2 7B	94.4 %	5.6 %	99.6 %	119.9
Llama 2 13B	94.1 %	5.9 %	99.8 %	82.1
Cohere-Chat	92.5 %	7.5 %	98.0 %	74.4
Cohere	91.5 %	8.5 %	99.8 %	59.8
Anthropic Claude 2	91.5 %	8.5 %	99.3 %	87.5
Google Palm 2 (beta)	91.4 %	8.6 %	99.8 %	86.6
Mistral 7B	90.6 %	9.4 %	98.7 %	96.1
Google Palm 2 Chat (beta)	90.0 %	10.0 %	100.0 %	66.2
Google Palm 2	87.9 %	12.1 %	92.4 %	36.2
Google Palm 2 Chat	72.8 %	27.2 %	88.8 %	221.1

Image credits: ***Vectara GitHub***

We can divide these hallucination types into three main categories:

- **Factual inaccuracies**: This type of hallucination occurs when a language model presents information that is not true or correct, but is framed as if factual. This includes dates, events, statistics or statements that are verifiably false. It can happen due to various reasons including misinterpretation of input data, low quality data and training methodologies, reliance on outdated or incorrect sources or the blending of information from different contexts that leads to an inaccurate output.

- **Generated quotations or sources**: This occurs when a language model fabricates quotes or citations. It might generate a statement and incorrectly attribute it to a real person, or create a fictitious source that does not exist at all. This is problematic because it leads to misinformation, falsely attributed statements and confusion.

- **Logical inconsistencies**: This includes generating responses that are internally inconsistent or logically flawed. After generating a response for a user query, the LLM can contradict itself in further responses. It occurs when a model makes a series of statements that, when taken together, are incoherent or conflicting — challenging the credibility of the model's outputs and confusing users who rely on its consistency.

In all of these cases, the language model is not intentionally misleading but is exhibiting its own limitations from various reasons that might include its training data, quality of data, knowledge cut-off date, poor fine-tuning, etc.

LLM hallucination mitigation strategies

Researchers are developing various approaches to make sure the response generated by LLMs is accurate. Some strategies need human intervention like reinforcement learning through human feedback (RLHF); others need fresh and custom data to train the model, known as fine-tuning. Retrieval augmented generation (RAG) is where an external knowledge source is provided to the LLM.

To address these limitations, several techniques have been developed:

1. Fine-tuning

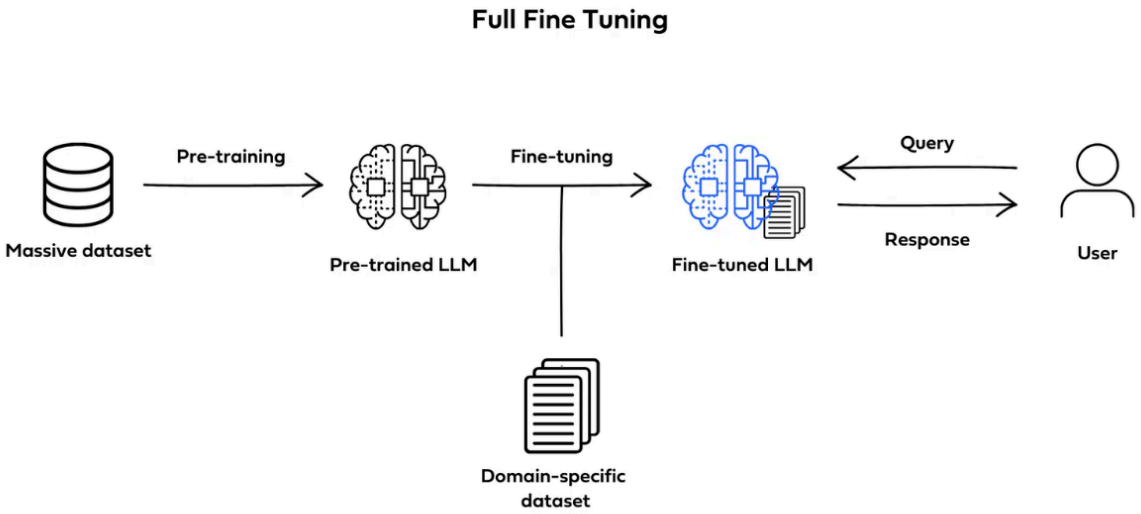


Image credits: Deci AI

Fine-tuning involves further training an LLM on a specific dataset related to the desired task. This helps the model learn the relevant domain vocabulary and nuances, leading to more accurate and tailored outputs.

Example: Fine-tuning an LLM on legal documents can improve its ability to accurately answer legal questions.

2. Prompt engineering

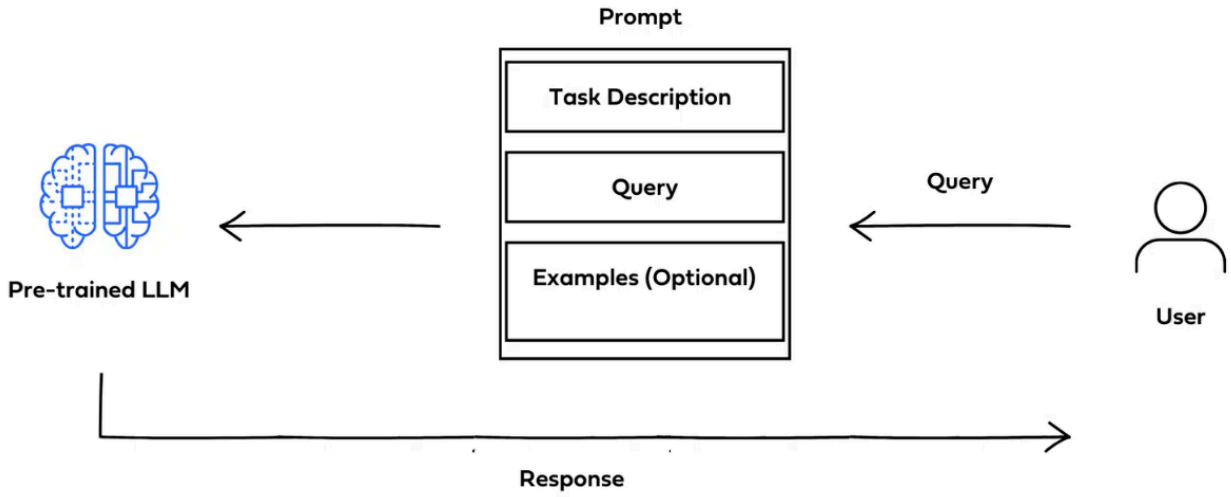


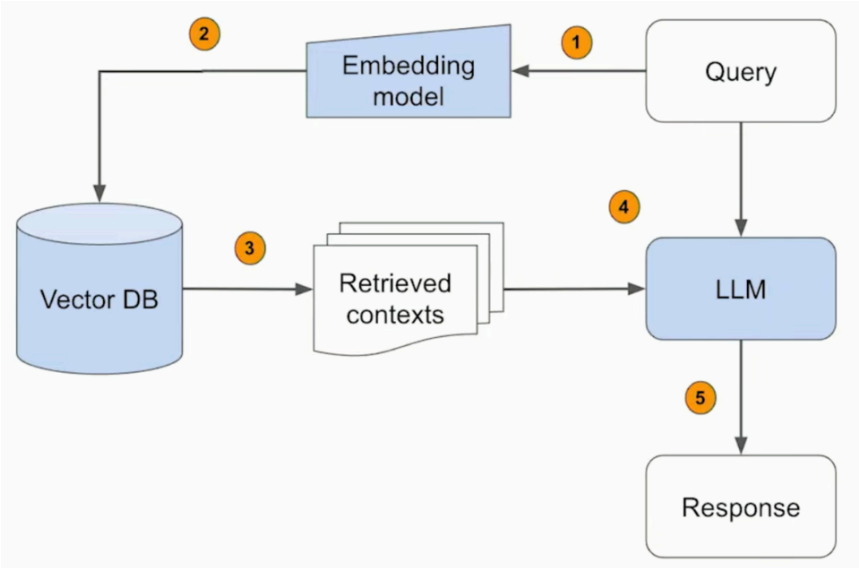
Image credits: Deci AI

Prompt engineering involves carefully crafting the prompts provided to the LLM. By providing specific instructions and context through the prompt, users can guide the LLM toward desired outputs.

Example: A prompt asking the LLM to "summarize the following news article in 50 words" will likely result in a more concise and informative output than simply providing the article itself.

3. Retrieval Augmented Generation (RAG)

RAG combines retrieval-based and generative approaches. It first retrieves relevant documents from a large corpus based on the user's query. Then, it uses the retrieved documents as input to generate a new and tailored response.



Let's understand the RAG workflow.

- **Query.** It starts with a query, which is the input or question you're asking the model.

- **Embedding model**. This query is then processed by an embedding model which transforms the query into a vector. This process involves converting the text into a numerical form, capturing semantic meaning in a high-dimensional space.

- **Vector database.** The query vector is used to retrieve relevant contexts from a large database of documents. These documents have also been converted into vectors in the same high-dimensional space.

- **Retrieved contexts.** The model retrieves the most relevant documents (contexts) based on how close their vectors are to the query vector. This is often done using nearest neighbor search algorithms.

- **LLM.**The retrieved documents are then provided to the LLM, along with the original query.

- **Response.** The LLM generates a response based on both the original query, and the additional context provided by the retrieved documents.

Example: A RAG system can search for relevant scientific papers based on a researcher's query, using those papers to generate a new and informative summary.

These techniques offer valuable solutions to overcome LLM limitations and unlock their full potential. By applying these techniques, we can build LLMs that are more context-aware, generate tailored outputs, possess specialized vocabulary and provide accurate and reliable information.

Additionally, other promising approaches are actively being explored

- **Bias detection and mitigation techniques.** These techniques aim to identify and address biases present in LLM training data and outputs.
- **Explainability methods.** These methods aim to make LLM decision-making processes more transparent and understandable.
- **Human-in-the-loop systems.** These systems involve human oversight and intervention to ensure the quality and reliability of LLM outputs.

By continuously improving and refining these techniques, we can work toward building more robust and trustworthy LLMs that benefit society in a responsible, ethical manner.

RAG chunking strategies

When implementing RAG in production, it is important to understand how chunking strategies help to avoid failures.

The Document is a chunk of information. It does not mean it is the whole text file, a PDF, a Word document, a presentation, a book, an image, etc.

All it means is that it has some information. This document is often generated from a large number of files and information that your firm has.

The process of generating documents/chunks from whatever input files your firm has been working on is often technically referred to as Chunking.

Chunking allows you to convert your information into “right” size pieces so that the pipeline can be most efficient, productive, accurate etc.

In technical terms, “Chunking” refers to segmenting the large corpus of documents into smaller, more manageable pieces that can be efficiently retrieved and processed by the model.

1: Naive Chunking
All it means is that at regular intervals of the length of the text, we split the text. Here is an example of how it is done and how the output looks like. we are splitting the text at each 200 characters disregarding if it is splitting the text in the middle of the word or a sentence etc.

2: Semantic Chunking
When we care about the sentence construction as a construct and split only when the sentences are separate as a strategy, it is called semantic chunking. Libraries like NLTK and Spacy are powerful enough to understand complex sentences and break them into appropriate parts.

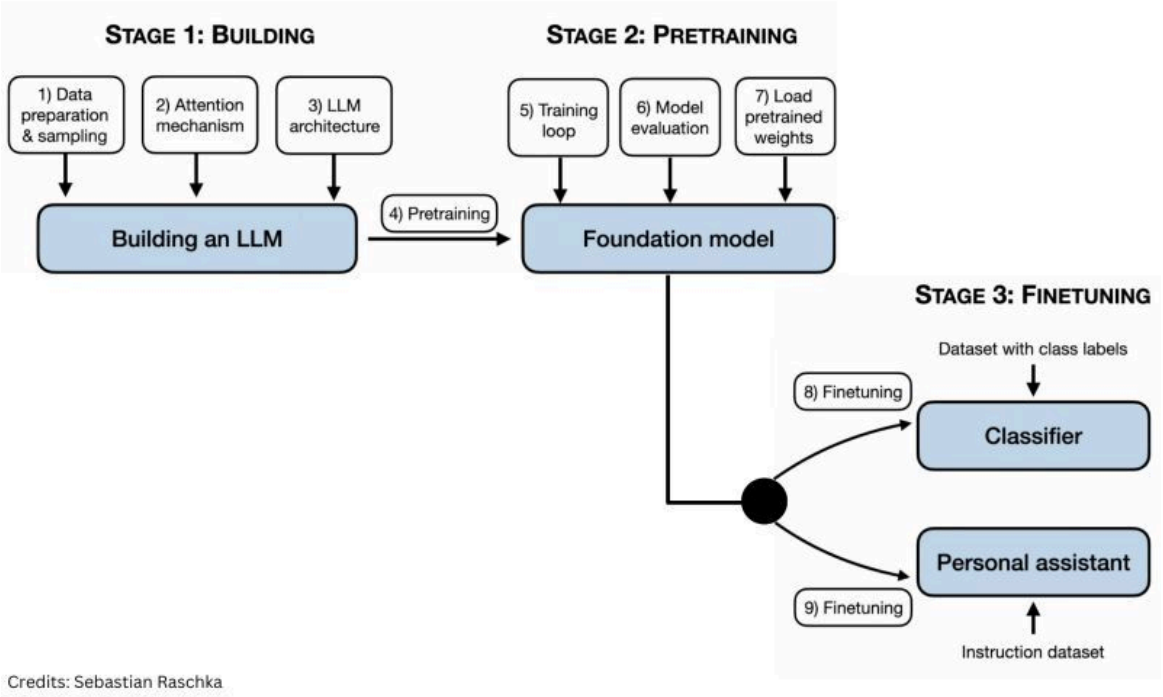
3. Compound Semantic Chunking

When each sentence is too small. Some documents need to have a longer length but not too long a modification of Semantic chunking can be used. This way the chunk is grown by concatenating the sentences until an appropriate threshold for the size is met.

4: Recursive Chunking
When dealing with documents that use certain clues to the newlines or sentence separation like multiple spaces or multiple new line characters or even multiple HTML tags, often it is useful to split the text by such indicator. This strategy helps in niche domains as this is the characteristic of the language that has been developed by the users as a standard practice.

Know in-depth about these chunking strategies with this hands-on tutorial.

Developing a Large Language Model



What goes into developing an LLM? Let's understand.

LLMs are the backbone of our GenAI applications and it is very important to understand what goes into creating these LLMs.

Just to give you an idea, here is a very basic setup and it involves 3 stages.

Here are the different stages of building an LLM.
Stage 1: Building
Stage 2: Pre-training
Stage 3: Finetuning

 Building Stage:
⦿ Data Preparation: Involves collecting and preparing datasets.
⦿ Model Architecture: Implementing the attention mechanism and overall architecture

 Pre-Training Stage:
⦿ Training Loop: Using a large dataset to train the model to predict the next word in a sentence.
⦿ Foundation Models: The pre-training stage creates a base model for further fine-tuning.

 Fine-Tuning Stage:
⦿ Classification Tasks: Adapting the model for specific tasks like text categorization and spam detection.
⦿ Instruction Fine-Tuning: Creating personal assistants or chatbots using instruction datasets.

Modern LLMs are trained on vast datasets, with a trend toward increasing the size for better performance.

The above explained process is just the tip of the iceberg but its a very complex process that goes into building an LLM. It takes hours to explain this but just know that developing an LLM involves gathering massive text datasets, using self-supervised techniques to pretrain on that data, scaling the model to have billions of parameters, leveraging immense computational resources for training, evaluating capabilities through benchmarks, fine-tuning for specific tasks, and implementing safety constraints.

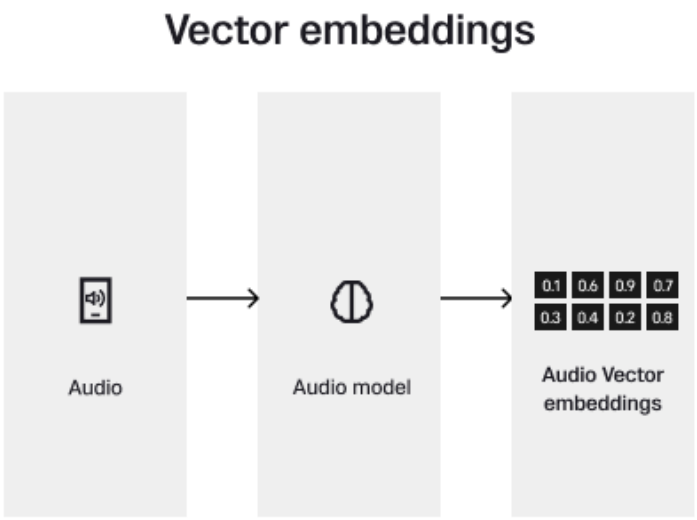
Know more about building an LLM in this video by Sebastian Raschka.

Vector databases

A vector database is a kind of database that is designed to store, index and retrieve data points with multiple dimensions, commonly referred to as vectors. Unlike databases that handle data (like numbers and strings) organized in tables, vector databases are specifically designed for managing data represented in multi-dimensional vector space. This makes them highly suitable for AI and machine learning applications, where data often takes the form of vectors like image embeddings, text embeddings or other types of feature vectors.

These databases utilize indexing and search algorithms to conduct similarity searches, enabling them to rapidly identify the most similar vectors within a dataset. This capability is crucial for tasks like recommendation systems, image and voice recognition — as well as natural language processing, since efficiently understanding and processing high dimensional data plays a vital role. Consequently, vector databases represent an advancement in database technology tailored to meet the requirements of AI applications that rely heavily on vast amounts of data.

Vector Embeddings



When we talk about vector databases, we should definitely know what vector embeddings are — how data eventually gets stored in a vector database. Vector embeddings are dense representations of objects (including words, images or user profiles) in a continuous vector space. Each object is represented by a point (or vector) in this space, where the distance and direction between points capture semantic or contextual relationships between the objects. For example in NLP, similar words are mapped close together in the embedding space.

The significance of vector embeddings lies in their ability to capture the essence of data in a form that computer algorithms can efficiently process. By translating high-dimensional data into a lower-dimensional space, embeddings make it possible to perform complex computations more efficiently. They also help in uncovering relationships and patterns in the data that might not be apparent in the original space.

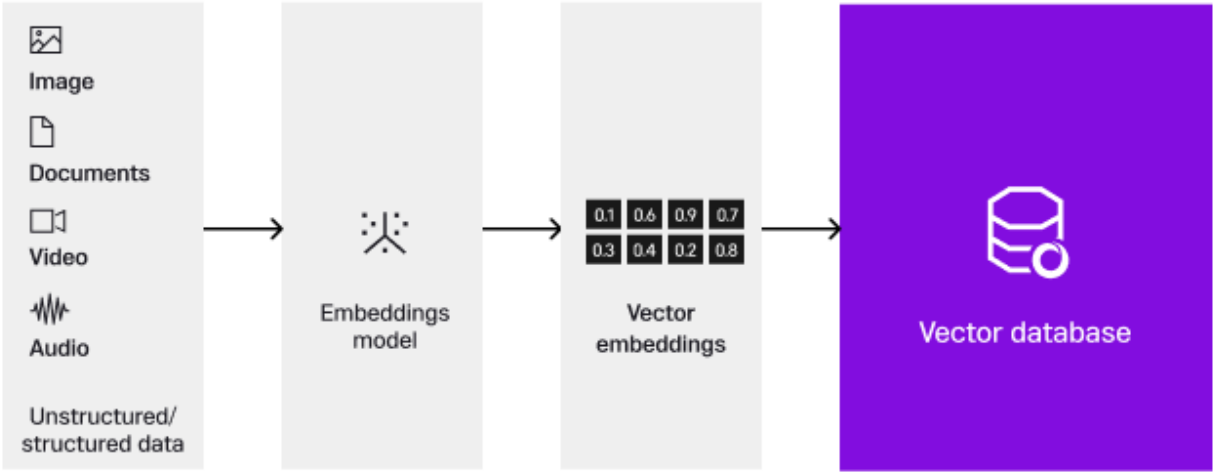
Types of vector embeddings

Generative AI applications are built using vector embeddings and the data source can range from text, audio, video and any type of structured and unstructured data. We can broadly divide these embeddings into four major categories:

- **Word embeddings**: These are the most common types of embeddings, used to represent words in NLP. Popular models include Word2Vec, GloVe and FastText.
- **Sentence and document embeddings**: These capture the semantic meaning of sentences and documents. Techniques like BERT and Doc2Vec are examples.
- **Graph embeddings**: These are used to represent nodes and edges of graphs in vector space, facilitating tasks like link prediction and node classification.
- **Image embeddings**: Generated by deep learning models, these represent images in a compact vector form, useful for tasks like image recognition and classification.

Here is my complete article on vector embeddings.

How vector databases work



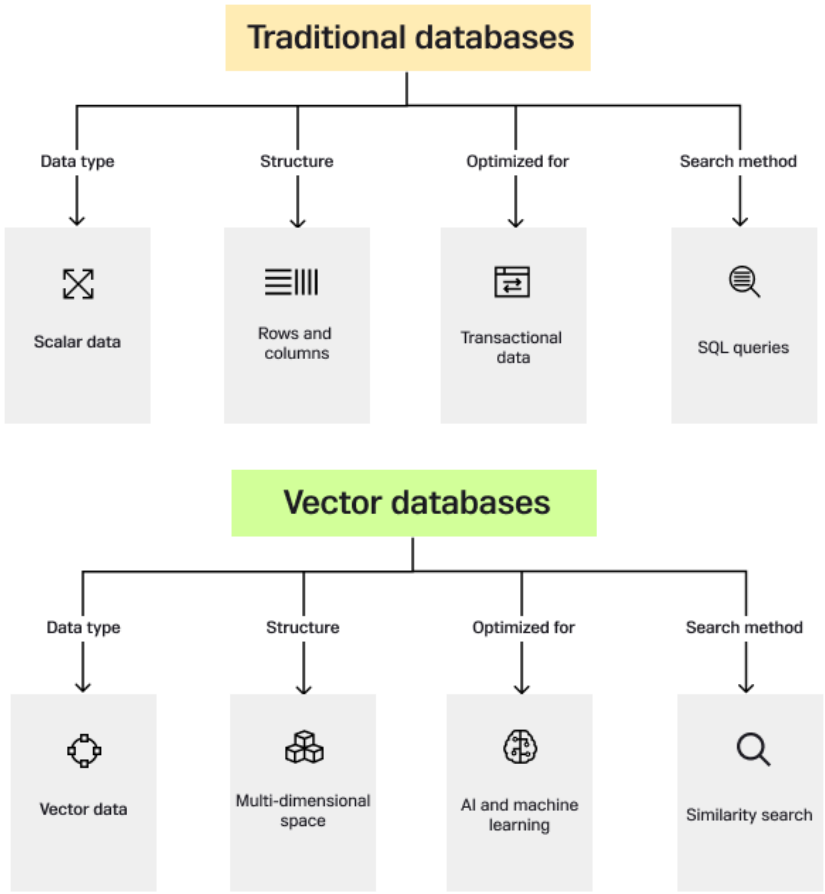
When a user query is initiated, various types of raw data including images, documents, videos and audio can be generated. All of this, which can be either unstructured or structured, is first processed through an embedding model. This model is often a complex neural network, translating data into high-dimensional numerical vectors and effectively encoding the data's characteristics into vector embeddings — which are then stored into a vector database.

When retrieval is required, the vector database performs operations (like similarity searches) to find and retrieve the vectors most similar to the query, efficiently handling complex queries and delivering relevant results to the user. This entire process enables the rapid and accurate management of vast, varied data types in applications that require high-speed search and retrieval functions.

Interested in learning more about vector databases? Check out this hands-on tutorial taking a deep dive into vector databases.

Traditional databases vs. vector databases

Let’s explore the difference between a vector database and a traditional database.



Vector databases represent a significant departure from traditional databases in their approach to data organization and retrieval. Traditional databases are structured to handle discrete, scalar data types like numbers and strings, organizing them in rows and columns.

This structure is ideal for transactional data but less efficient for the complex, high-dimensional data typically used in AI and machine learning. In contrast, vector databases are designed to store and manage vector data — arrays of numbers that represent points in a multi-dimensional space.

This makes them inherently suited for tasks involving similarity search where the goal is to find the closest data points in a high-dimensional space — a common requirement in AI applications like image and voice recognition, recommendation systems and natural language processing. By leveraging indexing and search algorithms optimized for high-dimensional vector spaces, vector databases offer a more efficient and effective way to handle the kind of data that is increasingly prevalent in the age of advanced AI and machine learning.

Want to learn more about vector databases and building AI applications? Check out Vector Databases & AI Applications for Dummies, SingleStore Special Edition.

The vector database landscape

Let's look at five approaches for persisting and retrieving vector data:

➤ Pure vector databases like Pinecone

➤ Full-text search databases like ElasticSearch

➤ Vector libraries like Faiss and Annoy

➤ Vector-capable NoSQL databases like MongoDB®, Cosmos DB and Cassandra

➤ Vector-capable SQL databases like SingleStore or PostgreSQL

In pure vector databases, data is organized and indexed based on the vector representation of objects or data points.

These vectors can be numerical representations of various data types including images, text documents, audio files or any other form of structured or unstructured data.

SingleStore database. SingleStore provides a simpler, more powerful approach to handling vector data. It allows you to store and query vector data alongside traditional structured data, providing a unified platform for various types of queries and analysis. As a distributed SQL database, SingleStore is also highly performant, highly available and can scale out to adapt to growing data sets.

How vector databases search and retrieve data

Unlike traditional relational databases, vector databases handle data in a fundamentally different way. They store and retrieve information based on vector embeddings, which capture the semantic meaning and relationships between data points in a high-dimensional space. This enables a powerful approach called semantic search, where the focus shifts from exact keyword matches to finding similar data based on its underlying meaning and context.

Here's how vector databases search and retrieve data

1. Data representation

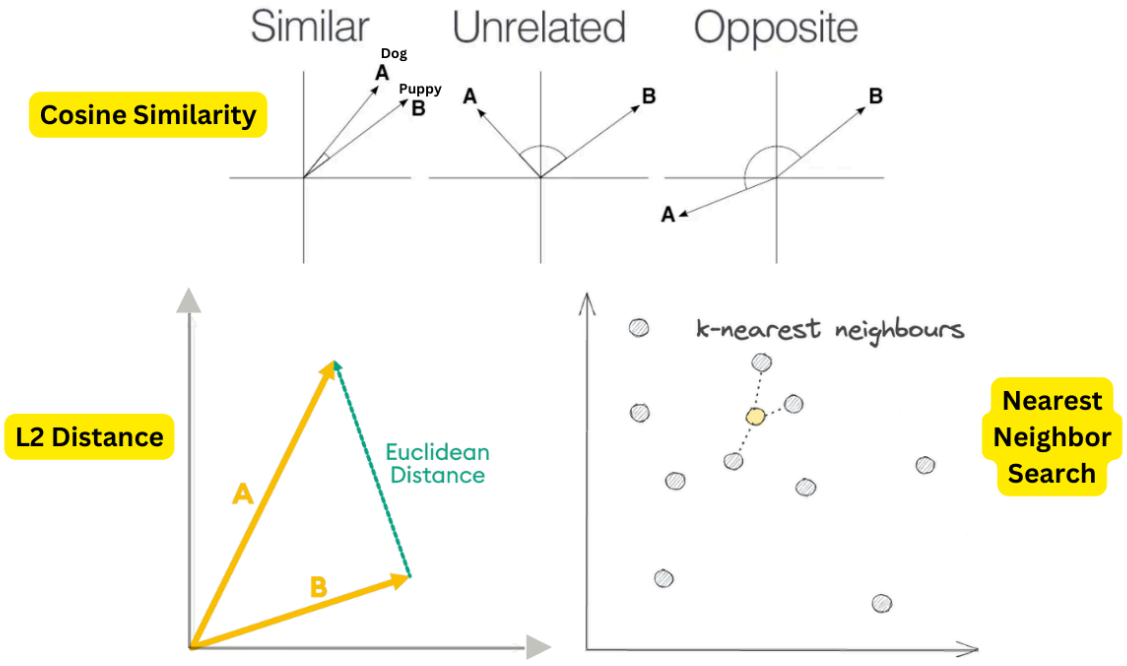
Each data point is converted into a high-dimensional vector, capturing its key features and characteristics. This process involves complex algorithms like word2vec or GloVe, which analyze textual data and map them to numerical vectors.

These vectors encode the semantic meaning of the data, allowing the database to understand the relationships between different data points.

2. Indexing and search

The vectors are then indexed in a specialized data structure optimized for efficient search. This index allows the database to quickly find vectors that are similar to a query vector.
The query vector can be generated in the same way as a data point vector, representing the desired search criteria or the user's intent.

3. Similarity search techniques



Several methods can be used to search for similar vectors within the database:

- **Cosine similarity.** This popular method measures the angle between two vectors. Smaller angles indicate higher similarity, as the vectors point in the same direction in the high-dimensional space.
- **L2 distance.** This method calculates the Euclidean distance between two vectors. Smaller distances indicate higher similarity, as the vectors are closer together in the space.
- **Nearest neighbor search algorithms.** These algorithms search for the k nearest neighbors of the query vector. This allows retrieving the most similar data points even if they don't perfectly match the query.

4. Retrieval and ranking
Once similar vectors are identified, the database retrieves the corresponding data points and ranks them based on their similarity score.

This allows users to access relevant information — even if it doesn't use the exact keywords they provided in their query.

Benefits of semantic search

- **Improved relevance.** Semantic search retrieves data based on its actual meaning, leading to more relevant and accurate results.
- **Understanding intent.** Vector databases can understand the user's intent behind the query, leading to more context-aware results.
- **Handling synonyms and variations.** The system can handle synonyms, variations and different phrasings, leading to a more user-friendly experience.
- **Efficient search.** Vector databases can efficiently retrieve similar data points, even for large datasets.

Other retrieval approaches

- **Metadata filtering.** Vector databases can combine semantic search with user-defined metadata filters, further refining the retrieved data.
- **Hybrid approaches.** Combining vector search with traditional keyword search can leverage the strengths of both techniques.
- **Multimodal search.** Integrating search with other modalities like images or audio can enhance the retrieval process.

The emergence of vector databases marks a significant shift in data search and retrieval. Their ability to understand semantic meaning and relationships opens doors for new applications across diverse fields. By utilizing these powerful tools, we can unlock more insightful information — and discover its full potential.

Some notable generative AI frameworks + tools

The rapid advancement of generative AI has spurred the development of various frameworks designed to facilitate the creation and application of this exciting technology. These frameworks provide the necessary infrastructure and tools for developers and researchers to explore the possibilities of generative AI. Here are a few examples:

LangChain

Developed by Harrison Chase and debuted in October 2022, LangChain serves as an open-source platform designed for constructing sturdy applications powered by LLMs, including chatbots like ChatGPT and various tailor-made applications.

LangChain seeks to equip data engineers with an all-encompassing toolkit for utilizing LLMs in diverse use cases, including chatbots, automated question answering, text summarization and beyond.

LangChain is composed of six key modules:

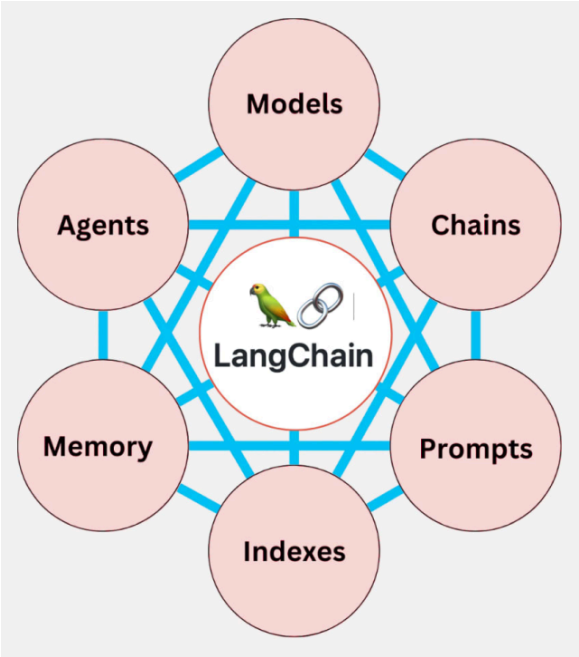
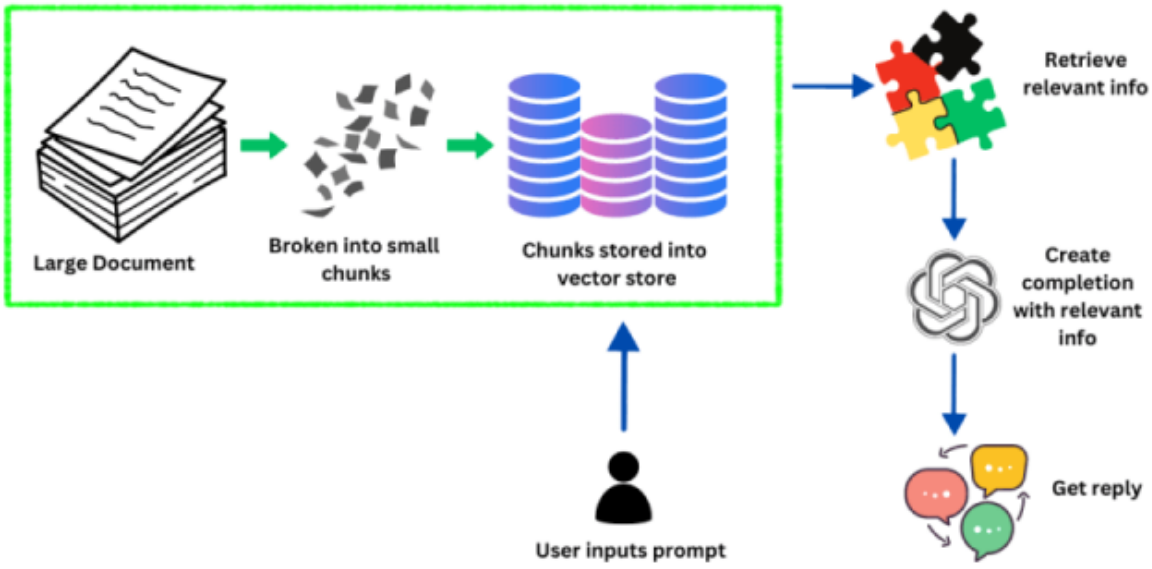


Image credits: ByteByteGo

1. **LLMs.** LangChain serves as a standard interface that allows for interactions with a wide range of LLMs.
2. **Prompt construction.** LangChain offers a variety of classes and functions designed to simplify the process of creating and handling prompts.
3. **Conversational memory.** LangChain incorporates memory modules that enable the management and alteration of past chat conversations, a key feature for chatbots that need to recall previous interactions.
4. **Intelligent agents.** LangChain equips agents with a comprehensive toolkit. These agents can choose which tools to utilize based on user input.
5. **Indexes.** Indexes in LangChain are methods for organizing documents in a manner that facilitates effective interaction with LLMs.
6. **Chain.** While using a single LLM may be sufficient for simpler tasks, LangChain provides a standard interface and some commonly used implementations for chaining LLMs together for more complex applications — either among themselves or with other specialized modules.



LangChain is equipped with memory capabilities, integrations with vector databases, tools to connect with external data sources, logic and APIs. This makes LangChain a powerful framework for building LLM-powered applications.

You can install LangChain using the following pip command.

```
pip install langchain
```

You can find out more — and try a hands-on tutorial — in our beginner's guide to LangChain.

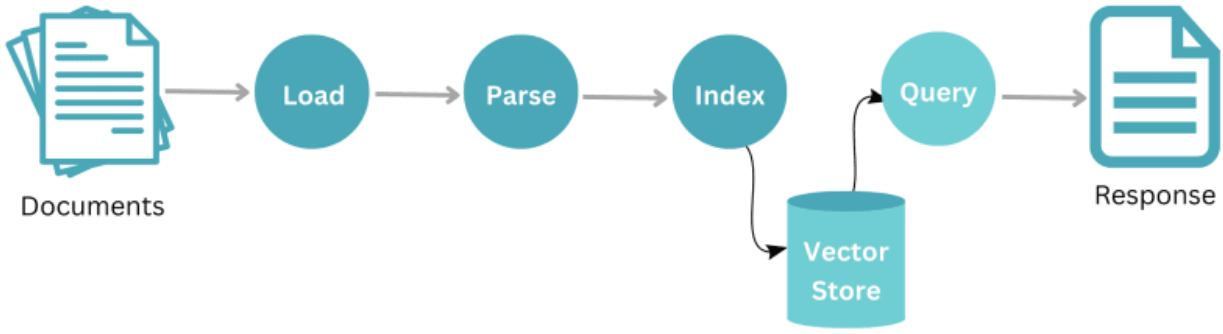
LlamaIndex

LlamaIndex is an advanced orchestration framework designed to amplify the capabilities of LLMs like GPT-4. While LLMs are inherently powerful, having been trained on vast public datasets, they often lack the means to interact with private or domain-specific data. LlamaIndex bridges this gap, offering a structured way to ingest, organize and harness various data sources — including APIs, databases and PDFs.

By indexing this data into formats optimized for LLMs, LlamaIndex facilitates natural language querying, enabling users to seamlessly converse with their private data without the need to retrain the models. This framework is versatile, catering to both novices with a high-level API for quick setup, and experts seeking in-depth customization through lower-level APIs. In essence, LlamaIndex unlocks the full potential of LLMs, making them more accessible and applicable to individualized data needs.

How LlamaIndex works

LlamaIndex serves as a bridge, connecting the powerful capabilities of LLMs with diverse data sources, thereby unlocking a new realm of applications that can leverage the synergy between custom data and advanced language models. By offering tools for data ingestion, indexing and a natural language query interface, LlamaIndex empowers developers and businesses to build robust, data-augmented applications that significantly enhance decision-making and user engagement.



LlamaIndex operates through a systematic workflow that starts with a set of documents. Initially, these documents undergo a load process where they are imported into the system. Post loading, the data is parsed to analyze and structure the content in a comprehensible manner. Once parsed, the information is then indexed for optimal retrieval and storage.

This indexed data is securely stored in a central repository labeled "store". When a user or system wishes to retrieve specific information from this data store, they can initiate a query. In response to the query, the relevant data is extracted and delivered as a response, which might be a set of relevant documents or specific information drawn from them. The entire process showcases how LlamaIndex efficiently manages and retrieves data, ensuring quick and accurate responses to user queries.

Here is a hands-on tutorial on LlamaIndex: An Absolute Beginner's Guide to LlamaIndex.

Hugging Face

Hugging Face is a multifaceted platform that plays a crucial role in the landscape of artificial intelligence, particularly in the field of natural language processing (NLP) and generative AI. It encompasses various elements that work together to empower users to explore, build, and share AI applications.
Here's a breakdown of its key aspects:

1. Model Hub:

- Hugging Face houses a massive repository of pre-trained models for diverse NLP tasks, including text classification, question answering, translation, and text generation.
- These models are trained on large datasets and can be fine-tuned for specific requirements, making them readily usable for various purposes.
- This eliminates the need for users to train models from scratch, saving time and resources.

2. Datasets:

- Alongside the model library, Hugging Face provides access to a vast collection of datasets for NLP tasks.

- These datasets cover various domains and languages, offering valuable resources for training and fine-tuning models.
- Users can also contribute their own datasets, enriching the platform's data resources and fostering community collaboration.

3. Model Training & Fine-tuning Tools:

- Hugging Face offers tools and functionalities for training and fine-tuning existing models on specific datasets and tasks.
- This allows users to tailor models to their specific needs, improving their performance and accuracy in targeted applications.
- The platform provides flexible options for training, including local training on personal machines or cloud-based solutions for larger models.

4. Application Building:

- Hugging Face facilitates the development of AI applications by integrating seamlessly with popular programming libraries like TensorFlow and PyTorch.
- This allows developers to build chatbots, content generation tools, and other AI-powered applications utilizing pre-trained models.
- Numerous application templates and tutorials are available to guide users and accelerate the development process.

5. Community & Collaboration:

- Hugging Face boasts a vibrant community of developers, researchers, and AI enthusiasts.
- The platform fosters collaboration through features like model sharing, code repositories, and discussion forums.
- This collaborative environment facilitates knowledge sharing, accelerates innovation, and drives the advancement of NLP and generative AI technologies.

Hugging Face goes beyond simply being a model repository. It serves as a comprehensive platform encompassing models, datasets, tools, and a thriving community, empowering users to explore, build, and share AI applications with ease. This makes it a valuable asset for individuals and organizations looking to leverage the power of AI in their endeavors.

Here is a hands-on tutorial on Hugging Face: [Hugging Face 101]

Haystack

Haystack can be classified as an end-to-end framework for building applications powered by various NLP technologies, including but not limited to generative AI. While it doesn't directly focus on building generative models from scratch, it provides a robust platform for:

1. Retrieval-Augmented Generation (RAG):
Haystack excels at combining retrieval-based and generative approaches for search and content creation. It allows integrating various retrieval techniques, including vector search and traditional keyword search, to retrieve relevant documents for further processing. These documents then serve as input for generative models, resulting in more focused and contextually relevant outputs.

2. Diverse NLP Components:
Haystack offers a comprehensive set of tools and components for various NLP tasks, including document preprocessing, text summarization, question answering, and named entity recognition. This allows for building complex pipelines that combine multiple NLP techniques to achieve specific goals.

3. Flexibility and Open-source:
Haystack is an open-source framework built on top of popular NLP libraries like Transformers and Elasticsearch. This allows for customization and integration with existing tools and workflows, making it adaptable to diverse needs.

4. Scalability and Performance:
Haystack is designed to handle large datasets and workloads efficiently. It integrates with powerful vector databases like Pinecone and Milvus, enabling fast and accurate search and retrieval even with millions of documents.

5. Generative AI Integration:
Haystack seamlessly integrates with popular generative models like GPT-3 and BART. This allows users to leverage the power of these models for tasks like text generation, summarization, and translation within their applications built on Haystack.

While Haystack's focus isn't solely on generative AI, it provides a robust foundation for building applications that leverage this technology. Its combined strengths in retrieval, diverse NLP components, flexibility, and scalability make it a valuable framework for developers and researchers to explore the potential of generative AI in various applications.

The Rise of Small Language Models

Small language models (SLMs) are a type of artificial intelligence (AI) that are trained on a smaller dataset of text and code compared to their larger counterparts, known as large language models (LLMs).

This results in several key differences between the two:

Size:

As the name suggests, SLMs are significantly smaller than LLMs. They typically have fewer than 1 billion parameters, while LLMs can have hundreds of billions or even trillions.

This smaller size makes them:

- **Faster**: SLMs require less computational power to run, making them ideal for deploying on devices with limited resources, such as smartphones and edge devices.
- **More efficient**: Training an SLM takes less time and energy compared to an LLM.
- **More affordable**: The development and deployment costs of SLMs are generally lower than those of LLMs.

Focus:

Instead of being general-purpose like LLMs, SLMs are often designed for specific tasks or domains. This allows them to achieve higher accuracy and performance in their niche areas.

Some examples of SLMs include:

- **Chatbots**: SLMs can power chatbots that provide customer service, answer questions, or even have conversations.
- **Text summarization**: SLMs can be used to automatically summarize long articles or documents.
- **Machine translation**: SLMs can be used to translate text from one language to another.
- **Code generation**: SLMs can be used to generate code based on natural language instructions.

Limitations:

While SLMs offer several advantages, they also have some limitations:

- **Knowledge base**: Due to their smaller training datasets, SLMs have a more limited knowledge base compared to LLMs. This can lead to less accurate or nuanced responses in situations that require broader context or understanding.
- **Generalizability**: SLMs trained for specific tasks may not perform well on other tasks, making them less adaptable than LLMs.

Here are some examples of small language models.

→ DistilBERT
→ Orca 2
→ Phi 2
→ BERT Mini

→ GPT-Neo
→ GPT-J
→ MobileBERT
→ T5-Small

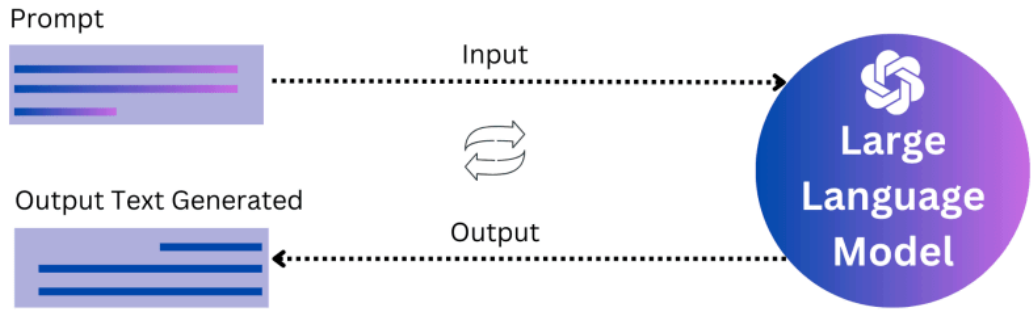
Here is an image depicting the key differences between small and large language models:

Aspect	Small Language Models	Large Language Models
Size	Can have less than 15 million parameters.	Can have hundreds of billions of parameters.
Computational Requirements	Can use mobile device processors.	Can require hundreds of GPU processors.
Performance	Can handle simple tasks.	Can handle complex, diverse tasks.
Deployment	Easier to deploy in resource-constrained environments.	Deployment often requires substantial infrastructure.
Training	Can be trained a week.	Training can take months.

Image credits: ***Techopedia***

Small language models offer a valuable alternative to large language models in situations where efficiency, affordability, and domain-specific performance are priorities. As research and development in this area continue, we can expect to see even more powerful and versatile SLMs emerge in the future.

Prompt Engineering



Prompt engineering is the art and science of crafting effective prompts to guide and control the outputs of Generative AI models. These prompts act as instructions, providing context and specific goals to the model, ultimately influencing the quality and direction of its outputs.

Key aspects of prompt engineering:

- **Crafting clear and concise instructions**: The prompt should clearly communicate the desired outcome to the model, avoiding ambiguity and unnecessary information.
- **Specifying context and background information**: Providing relevant context and background information helps the model understand the intended meaning of the prompt and generate more accurate and relevant outputs.
- **Utilizing different prompting techniques**: Various techniques like prefix tuning, temperature control, and prompt chaining can be used to fine-tune the model's behavior and achieve specific effects.
- **Iterative refinement**: Prompting is an iterative process. By analyzing the model's outputs and refining the prompts based on feedback, users can achieve optimal results.

Benefits of prompt engineering:

- **Improved quality and relevance of outputs**: Effective prompts can significantly enhance the quality and relevance of the model's outputs, ensuring they align with the desired goals.
- **Control over creative direction**: Users can guide the model's creative direction and influence the style, tone, and overall direction of the generated content.
- **Exploration of new possibilities**: Prompting opens doors for exploring new possibilities and applications of generative AI, pushing the boundaries of what these models can achieve.
- **Customization for specific tasks**: Prompts can be tailored to specific tasks and domains, allowing users to leverage generative AI effectively for various purposes.

Examples of prompt engineering applications:

- **Content generation**: Crafting prompts for generating creative text formats like poems, scripts, code, musical pieces, marketing copy, news articles, product descriptions, etc.
- **Question answering**: Formulating prompts to guide the model towards providing comprehensive and informative answers to open-ended and challenging questions.
- **Image generation**: Providing descriptive prompts to instruct the model on specific features and details to be included in the generated image.
- **Code generation**: Utilizing prompts to guide the model in generating code snippets or even entire programs based on specific functionalities and requirements.

Challenges of prompt engineering:

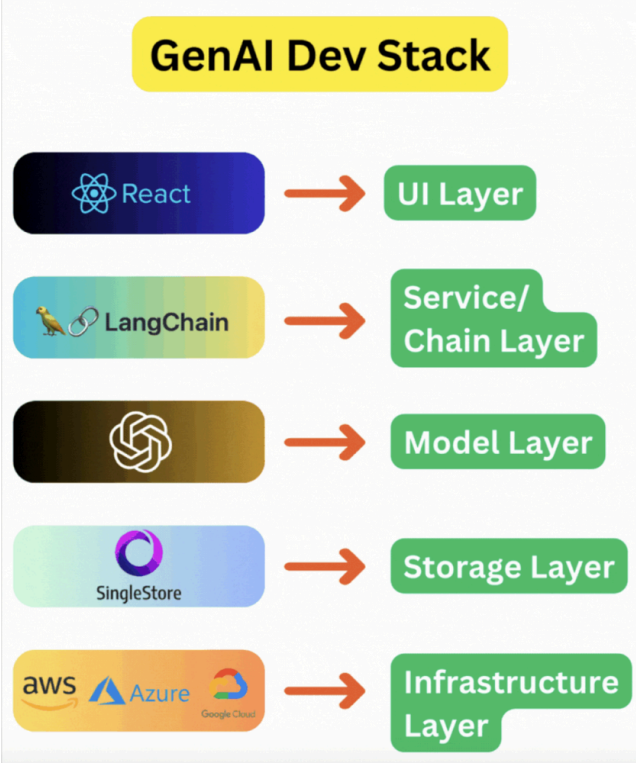
- **Bias and fairness**: Prompts can inherit biases present in the training data, leading to biased or unfair outputs. Careful crafting and evaluation of prompts are crucial to mitigate these risks.

- **Explainability and transparency**: Understanding how prompts influence the model's decision-making process can be challenging. Efforts are ongoing to develop methods for explaining and interpreting model outputs.
- **Overfitting and lack of creativity**: Overly specific prompts can restrict the model's creativity and lead to repetitive or unoriginal outputs. Striking a balance between guidance and freedom is essential for achieving optimal results.

Prompt engineering is a crucial skill for maximizing the potential of generative AI. By mastering this art, users can unlock new possibilities, generate creative and valuable outputs, and push the boundaries of artificial intelligence. As the field of generative AI continues to evolve, prompt engineering will play an increasingly important role in shaping the future of this transformative technology.

Generative AI Developer Stack

The concept refers to the set of technologies, frameworks, and tools used in developing applications that leverage Generative AI. This stack would typically include components for handling different aspects of application development, from the user interface to backend services, data management, and integration with AI models.



The image depicts a stack of technologies used in the GenAI Dev Stack, which is likely a development stack for a Generative Artificial Intelligence (AI) application. The stack is structured in layers, each representing a different aspect of the application's architecture:

 UI Layer: At the top, represented by "React", indicating that React (a JavaScript library for building user interfaces) is used for creating the user interface of the application.

 Service/Chain Layer: Below the UI layer, there's "LangChain", which is an open-source framework that helps developers create applications using large language models (LLMs).

 Model Layer: The next layer includes OpenAI's CatGPT and HuggingFace. Hugging Face is a machine learning and data science platform that helps users build, train, and deploy machine learning models.

 Storage Layer: The "SingleStore" in the following layer suggests that a database technology that unifies multiple data storage capabilities (like transactions, vector embeddings and real-time analytics) is used for data storage (vector database).

 Infrastructure Layer: At the bottom of the stack, "AWS", "Azure", and "Google Cloud" indicate that the application's infrastructure is cloud-based, possibly utilizing services from all three major cloud providers for hosting, computing, and other infrastructure needs.

The significance of vector databases stems from their ability to handle complex, high-dimensional data while offering efficient querying and retrieval mechanisms. One such database you can use is the SingleStore database.

Using Generative AI Responsibly

Using Generative AI responsibly involves adhering to ethical guidelines and best practices to ensure that its applications are safe, fair, and beneficial. This includes respecting intellectual property rights by not generating copyrighted or trademarked material without permission. Privacy concerns should be addressed by not creating or sharing personal data without consent. It's crucial to avoid generating harmful or offensive content, including deepfakes or materials that could incite violence or discrimination.

Ensuring transparency about the use of AI-generated content is important to maintain trust and prevent misinformation. Additionally, creators should be mindful of perpetuating biases and strive to use AI in ways that promote diversity and inclusivity. Responsible use of Generative AI also involves staying informed about evolving ethical standards and legal regulations in this rapidly changing field.

Best Practices for Responsible Generative AI Use:

Data:

- **Use diverse and representative datasets**: To avoid bias, train AI models on data that reflects the real world in terms of demographics, backgrounds, and perspectives.

- **Minimize data collection and retention**: Only collect and retain data necessary for the intended purpose, and implement secure deletion policies for outdated data.
- **Protect privacy**: Obtain informed consent before using personal data and ensure its secure storage and handling.

Development and Deployment:

- **Design for transparency and explainability**: Make AI models and their outputs understandable to users, allowing them to assess potential biases or limitations.
- **Implement robust safeguards**: Build in filters and mechanisms to prevent the generation of harmful content like misinformation, deepfakes, or offensive material.
- **Conduct thorough testing and evaluation**: Rigorously test AI systems for bias, fairness, and safety before deployment, and continuously monitor their performance in real-world settings.

Content and User Interaction:

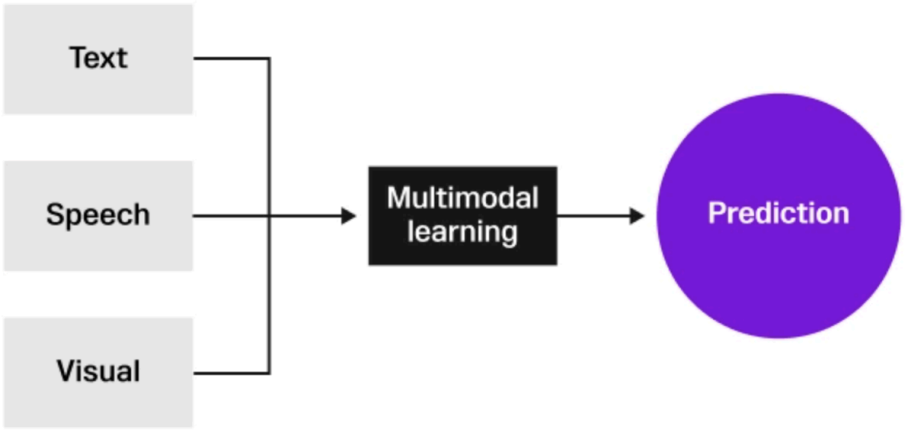
- **Clearly label AI-generated content**: Differentiate AI-generated content from human-created work, preventing confusion and potential misuse.
- **Promote user agency and control**: Provide users with options to modify, reject, or report AI-generated content that is inappropriate or harmful.
- **Combat misinformation and disinformation**: Implement fact-checking and verification mechanisms to ensure the accuracy and reliability of AI-generated content.

Societal Impact and Governance:

- **Engage in open dialogue and collaboration**: Foster discussion and debate about the ethical implications of Generative AI with diverse stakeholders, including the public, policymakers, and researchers.
- **Develop ethical guidelines and frameworks**: Establish clear principles and best practices for responsible AI development and use, and encourage their adoption by developers and organizations.
- **Promote public awareness and education**: Educate the public about Generative AI, its capabilities, limitations, and potential risks, fostering responsible use and informed decision-making.

Remember, responsible Generative AI use is an ongoing process. By continuously learning, adapting, and collaborating, we can ensure this powerful technology benefits everyone in a safe, fair, and ethical way.

Multimodal Models



In the context of machine learning and artificial intelligence, multimodal models are systems designed to understand, interpret or generate information from multiple types of data inputs or "modalities." These modalities can include text, images, audio, video and sometimes even other sensory data like touch or smell.
The key characteristic of multimodal models is their ability to process and correlate information across these different types of data, enabling them to perform tasks that would be challenging or impossible for models limited to a single modality.

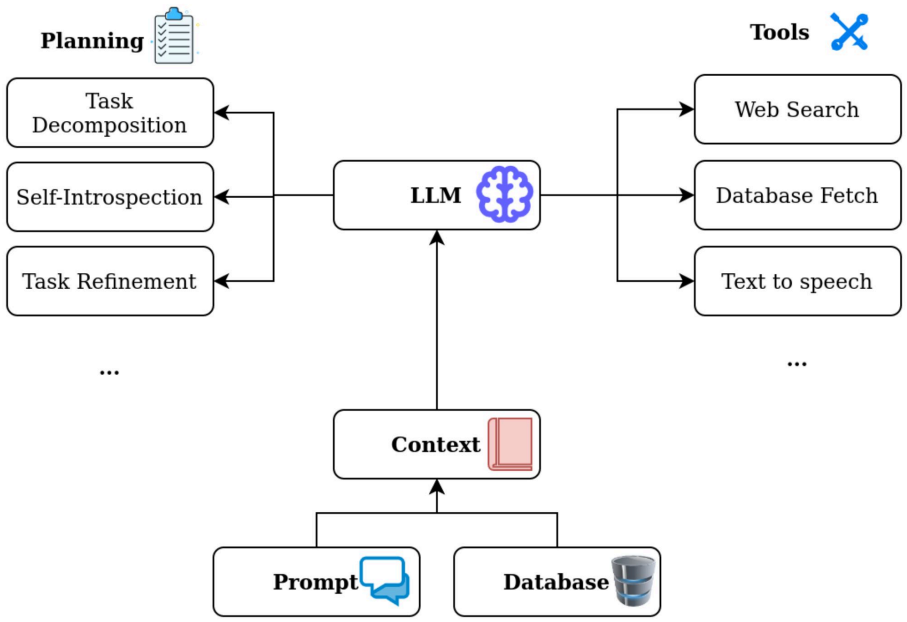
Context plays a vital role in LLMs when a user comes with any query. While traditional AI models often struggle with the limitations of single data types, adding context becomes a little challenging — resulting in an output with less accuracy. Multimodal models address this by considering data from multiple sources including text, images, audio and video. This broader perspective allows for more context and self-learning. And, this enhanced comprehension leads to improved accuracy, better decision-making and the ability to tackle tasks that were previously challenging.

Examples of multimodal models

Model	Year	Technical points	Function
Transformer	2017	Self-attention, positional Encoding and multi-Head Attention	Machine translation
ViT	2020	Patch-based Representation, linear Projection, transformer Encoder	Image classification and generation
BEiT	2021	Discrete visual embedding aggregation and MLM	Image Understanding and transfer learning
VisualChatGPT	2023	Invokes multiple VFMs, pre-trained LLMs and prompt management	Visual queries and instructions
MM-REACT	2023	Integration of ChatGPT and vision experts and textual prompt design	Visual understanding tasks
Frozen	2021	Few-Shot learning, utilize external knowledge and soft-prompting philosophy	VQA
BLIP-2	2023	Pre-trained image encoders, querying Transformer and frozen LLMs	Zero-shot image-to-text generation
LLaMA-Adapter	2023	Fine-tuning instruction-following, learnable adaption prompts and zero-initialized attention mechanism	Vision and language tasks
MiniGPT-4	2023	Frozen visual encoder with a frozen LLM, one projection layer and trained by image-text pairs	Identify humorous elements within images and create websites from handwritten drafts
LLaVA	2023	Instruction tuning LLMs and end-to-end trained LLMs	Visual and language understanding and multimodal chat abilities

Know more about [multimodal models in my article](#).

LLM-Powered Autonomous AI Agents



An LLM-based agent interacts with its environment through perception, sensing environmental data, and takes action based on the information, which may involve tools. In unimodal mode, the agent uses only text for both input and output.

In multi-modal mode, the agent can perceive using visual, auditory, and physical inputs and can execute embodied actions in the environment.

LLM-powered autonomous agents demonstrate these traits of autonomy:

➢ **Ability to plan**: These agents accept high-level instructions or goals in natural language and break them down into smaller tasks, plan their sequence of execution, evaluate their results, and refine them toward satisfying the instruction or goal as correctly as possible.

➢ **Ability to use tools**: These agents demonstrate the ability to understand software call syntax and semantics, select the software tools they need for any task, and run them by supplying syntactically and semantically sound parameters. These tools may be other LLM-based agents, non-LLM-based artificial intelligence agents, external application programming interfaces (APIs), retrievers from private data sources, or just simple functions that execute some data processing logic.

➢ **Ability to use contextual information**: Lastly, such agents can adapt their planning and actions based on contextual information present in the prompts (also called in-context learning) or in external data sources like vector databases (known as retrieval-augmented generation).

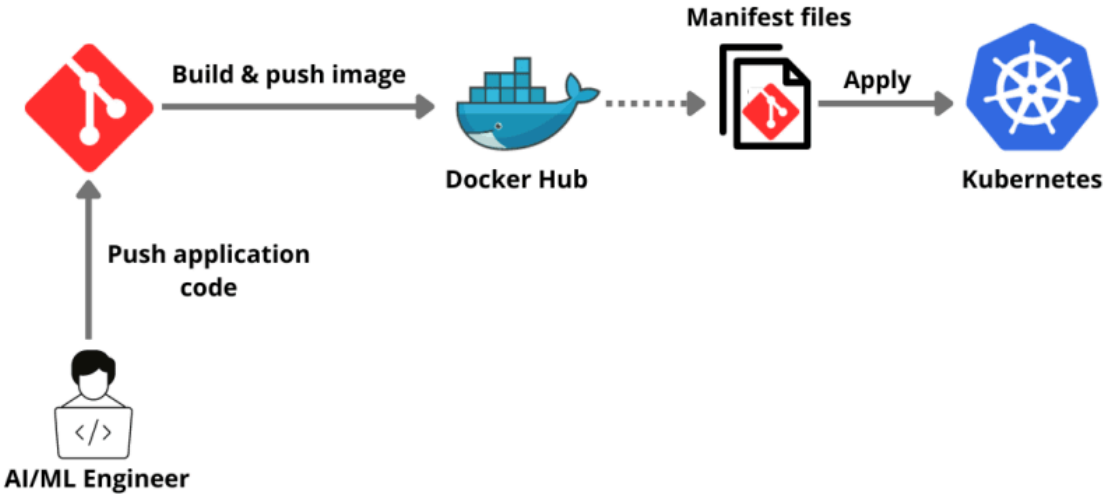
Know more about autonomous agents in this article.

Deploying GenAI Applications on Kubernetes: A Step-By-Step Guide!

Kubernetes, a powerful open-source platform, has emerged as a leading solution for managing and scaling containerized applications, ensuring they run seamlessly across various environments.

Why Deploy GenAI Applications On Kubernetes?

Deploying AI/ML applications on Kubernetes provides a robust solution for managing complex AI/ML workloads. One of the primary benefits is scalability. Kubernetes can automatically scale the infrastructure, accommodating varying workloads efficiently, ensuring that resources are allocated effectively based on demand. This auto-scaling feature is crucial for handling large computations involved in AI/ML tasks.



Additionally, Kubernetes supports multi-cloud and hybrid-cloud environments, offering flexibility and avoiding vendor lock-in. It provides a consistent and unified environment for development, testing, and deployment, enhancing the collaboration between data scientists and engineers. Kubernetes also ensures high availability and fault tolerance, automatically replacing or rescheduling containers that fail, ensuring the reliability and robustness of AI/ML applications. Furthermore, it simplifies many operational aspects, including updates and rollbacks, allowing teams to focus more on building AI/ML models rather than managing infrastructure.

A complete hands-on tutorial can be found here: Deploy Any AI/ML Application On Kubernetes: A Step-by-Step Guide!

Let's Build a Full-Stack AI Application in React

In the realm of real-time AI application development, few tools offer as much power and flexibility as the Elegance SDK. Pre-configured to harmoniously work with SingleStore, this SDK provides developers with a treasure trove of tools to craft impeccable full-stack applications infused with AI capabilities. In this article, we'll journey through the world of the Elegance SDK, and explore how to build a full-stack AI application using React.

What is the Elegance SDK?

The Elegance SDK is a pioneering toolkit designed to facilitate the swift construction of real-time AI applications. Its robust integrations with SingleStoreDB streamline the process, providing developers with a pre-configured environment ready for efficient application development. Notably, the Elegance SDK is a full-stack JavaScript toolkit, implying its capabilities stretch from back-end operations, using Node.js, to front-end endeavors — particularly with React.js.

Key features

- **Pre-configured SingleStoreDB connection**: Say goodbye to the often tedious process of setting up database connections. Elegance SDK provides a seamless connection experience with SingleStoreDB, ensuring your data flows without a hitch.
- **AI integrations**: Harness the power of AI with minimal hassle. From chat completions to content suggestions, the SDK offers tools that tap into the OpenAI API, imbuing your applications with intelligent functionalities.
- **Support for multiple databases**: Beyond SingleStoreDB, the SDK is versatile, offering support for MySQL and MongoDB using SingleStore Kai™.
- **File embeddings generation**: An intriguing feature, this allows developers to convert various file formats into actionable insights, paving the way for enhanced data analysis.
- **Full-stack tooling**: With ready-to-use Node.js controllers for the back end and React.js hooks for the front end, the SDK ensures a holistic development experience.

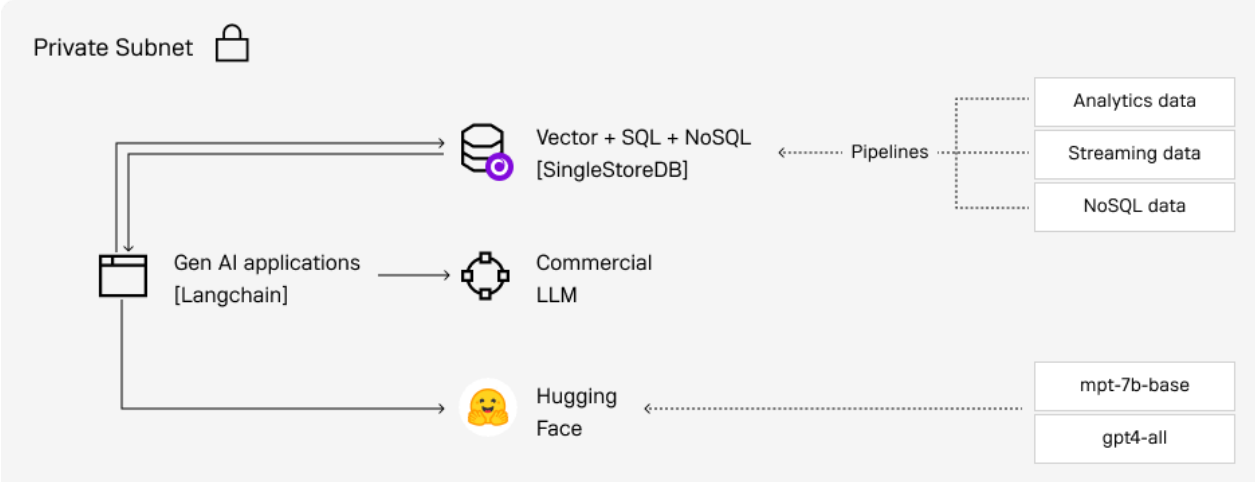
Here is the hands-on guide to build a full stack AI app: [Build a Full-Stack AI Application in React with the Elegance SDK]

SingleStore for Generative AI Applications

Generative AI applications require a robust and versatile database solution to effectively manage and utilize the massive amounts of data they generate. SingleStore Database, with its unique architecture and capabilities, emerges as a strong contender for this role.

Using SingleStoreDB as a full-context vector database

Experience faster vectors with exact nearest neighbor search and better compression with a simple architecture. SingleStoreDB is an enterprise-grade solution that powers fast ingestion with millisecond response times (no ETL), and enables complex joins with fresh data from transactions and other sources.



Notable Companies Using SingleStore

SIEMENS: Sentiment analysis on employee survey responses.

Siemens is driving sentiment analysis using vector capabilities within SingleStoreDB to analyze and gain deeper insights into the responses from company-wide HR surveys across 200,000 employees worldwide.

Nyris: Real-time image search and object recognition for search and analytics.

nyris.io, a leading AI-based visual search engine for retailers is powering its platform (including catalog search, visual search and analytics) using dot_product vector similarity capabilities in SingleStoreDB.

DirectlyApply: Catalog, customer 360 and personalization matching job seekers with roles.

DirectlyApply, a job discovery platform and vertical search engine that connects job seekers with employers, uses SingleStoreDB to store vector embeddings generated from job titles and run vector similarity search (using dot_product) to match embeddings with job openings and standard ISCO job titles.

Lumana: AI-driven video monitoring and surveillance for safety and security.

Lumana, a SaaS visual intelligence platform for real-time video monitoring and surveillance uses SingleStoreDB's vector functionality to perform image and video matching to monitor occupational safety, surveillance footage and more.

Know more about how SingleStore can help you build powerful Generative AI applications.

Sign up to SingleStore Database and claim your $600 worth free computing resources

Reach out for a conversation, we are happy to show the SingleStore demo and how to build resilient Gen AI applications with SingleStore Database.