

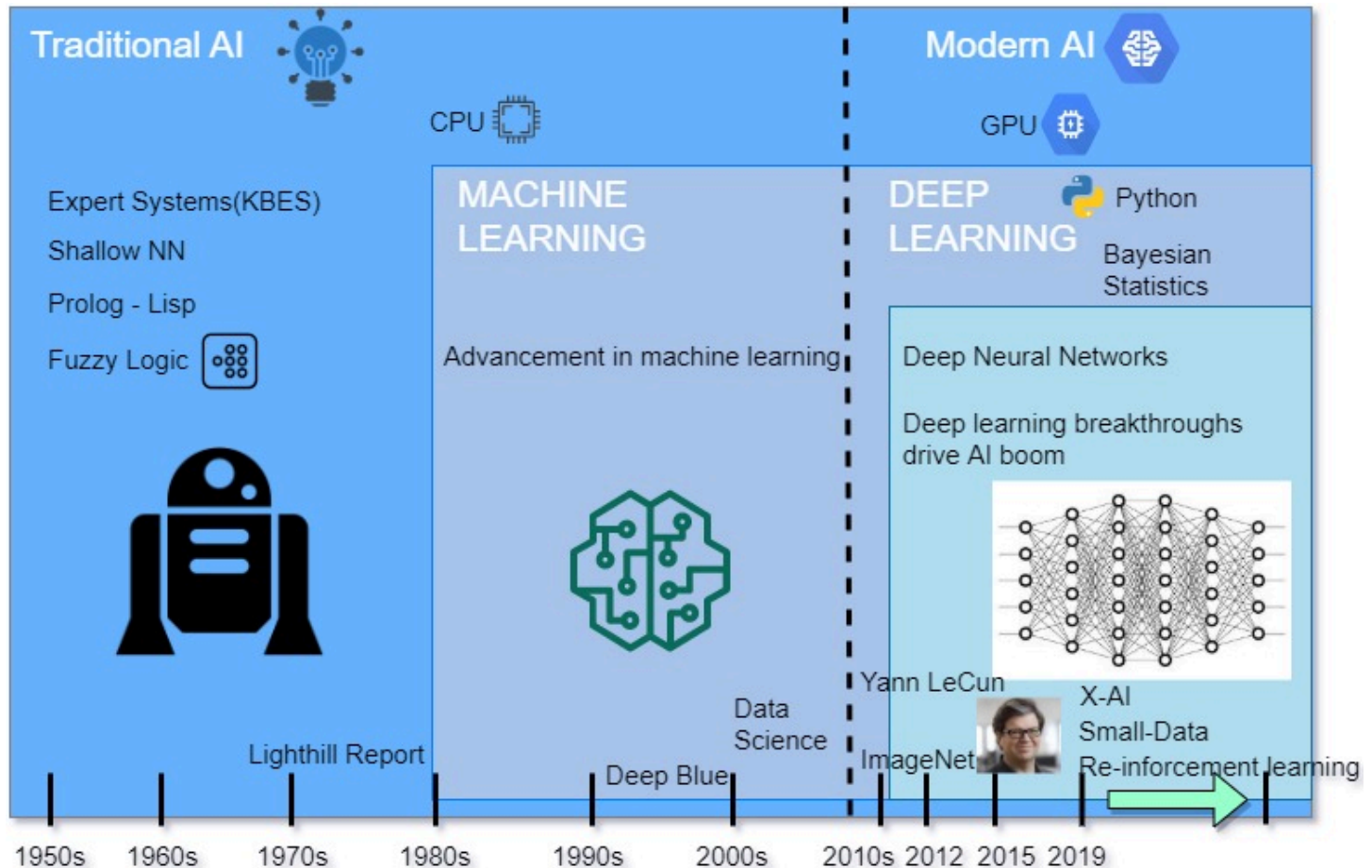
AI Explained: Making Sense of Ai (Essay)

by Diego Pacheco

<https://diegopacheco.github.io/>

What AI can do?

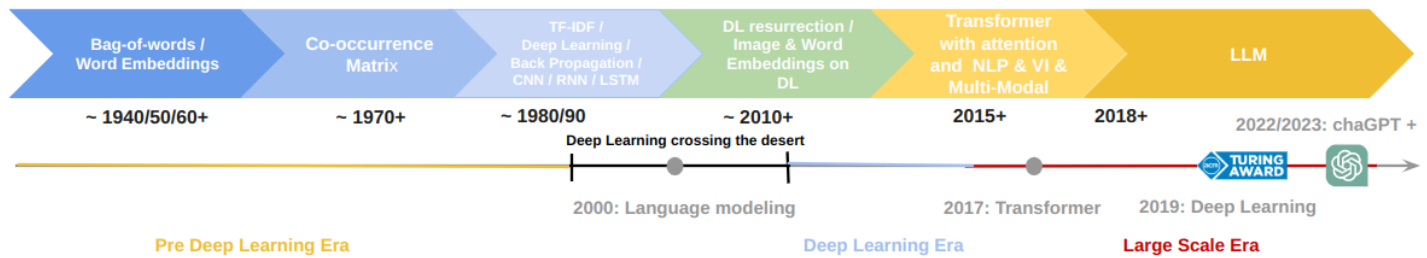
Traditional AI:



Source: <https://towardsdatascience.com/traditional-ai-vs-modern-ai-5117b469a0c9>

1. **Classification:** Classify data into predefined categories.
2. **Regression:** Predict continuous values.
3. **Clustering:** Group similar data points.
4. **Decision-making:** Make decisions based on rules and logic.
5. **Optimization:** Find the best solution among options.
6. **Natural Language Processing (NLP):** Understand and generate human language (e.g., chatbots, sentiment analysis).
7. **Computer Vision:** Interpret and understand visual data (e.g., image recognition, object detection).
8. **Robotics:** Control and interact with physical devices.
9. **Expert Systems:** Mimic human expertise in specific domains.
10. **Predictive Maintenance:** Predict equipment failures and schedule maintenance.

Generative AI (GenAI):



Source: <https://medium.com/@glegoux/history-of-the-generative-ai-aa1aa7c63f3c>

1. **Text Generation:** Create new text, such as articles, stories, or conversations.
2. **Image Generation:** Create new images, such as photos, artwork, or designs.
3. **Music Generation:** Compose music, melodies, or sound effects.
4. **Video Generation:** Create new videos, such as animations or clips.
5. **Data Generation:** Create synthetic data for training or testing.
6. **Style Transfer:** Transfer styles between images, music, or text.
7. **Image-to-Image Translation:** Translate images from one domain to another.
8. **Text-to-Image Synthesis:** Generate images from text descriptions.
9. **Dialogue Generation:** Engage in conversation, responding to user input.
10. **Creative Writing:** Generate creative writing, such as poetry or short stories.

Limits of GenAI:



1. **Lack of common sense:** GenAI models often struggle with real-world nuance and common sense.
2. **Contextual understanding:** While GenAI excels in specific contexts, it can fail to generalize to new situations.
3. **Bias and stereotypes:** If trained on biased data, GenAI models can perpetuate and amplify existing biases.
4. **Evaluation metrics:** Assessing GenAI's performance can be challenging due to the subjective nature of creativity and quality.
5. **Training data quality:** GenAI is only as good as the data it's trained on; poor-quality data can lead to suboptimal results.
6. **Mode collapse:** GenAI models may produce limited variations or repetitive output.

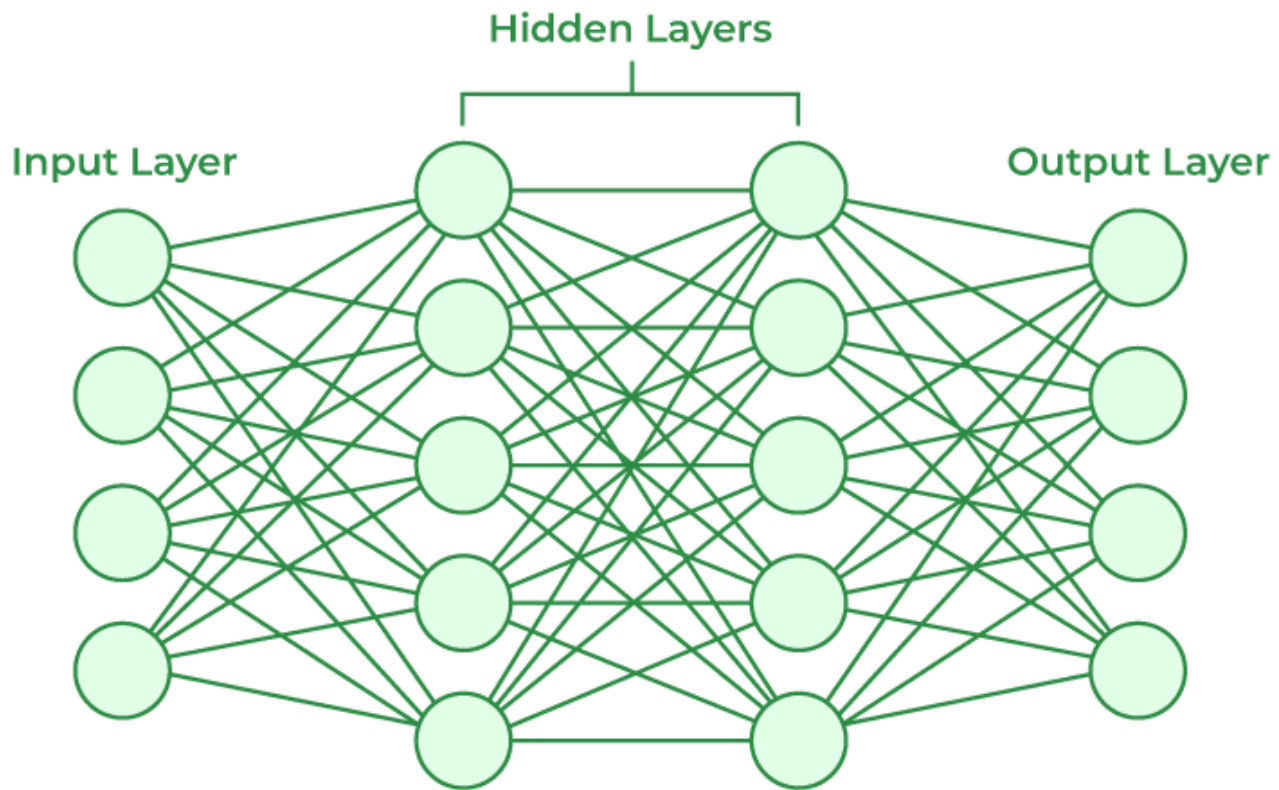
7. Lack of transparency: GenAI models can be difficult to interpret and understand, making it hard to identify errors or biases.

8. Ethical considerations: GenAI raises concerns around authorship, ownership, and potential misuse (e.g., deepfakes).

9. Scalability: Training and deploying GenAI models can be computationally intensive and require significant resources.

10. Human oversight: GenAI still requires human oversight and editing to ensure quality and accuracy.

Model Design



Source: <https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/>

Overfitting:

- occurs when a model is too closely aligned to a limited set of data points, causing it to be useful only for the initial data set and not for any other data sets.
- a function fits too closely or exactly to its training data, resulting in a model that can't make accurate predictions or conclusions from any data other than the training data.
- caused by low bias and high variance, complex models, or training data that contains too much noise.
- can be tackled by using k-fold cross-validation, regularization techniques, training with sufficient data, or ensembling techniques.

Underfitting:

- occurs when a model is too simple to capture data complexities, causing it to be unable to learn the training data effectively and resulting in poor performance on both the training and testing data.
- caused by high bias and low variance, simple models, or inadequate features used to train the model.

- can be tackled by increasing model complexity, increasing the number of features, removing noise from the data, or increasing the duration of training.

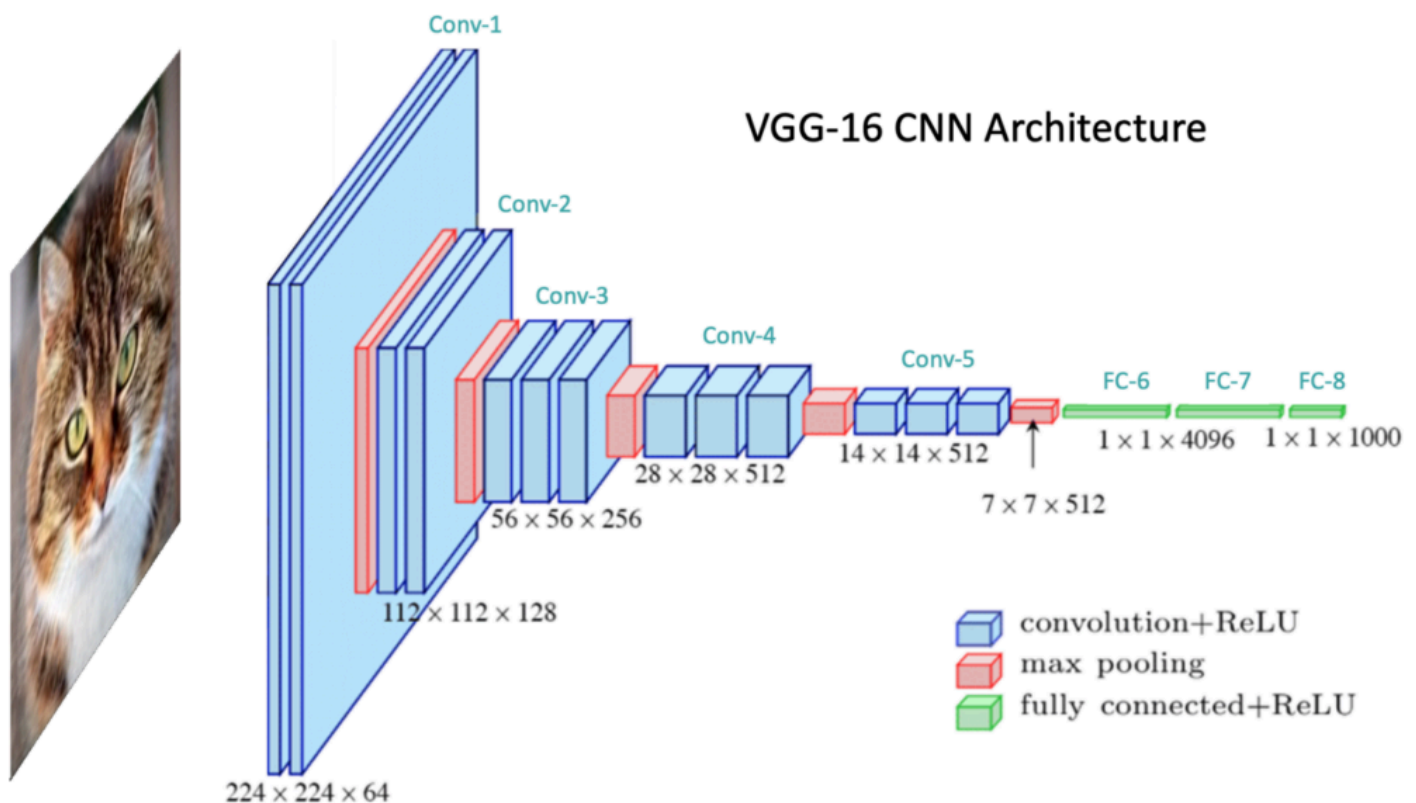
Common Mistakes with Models

Here are some common mistakes to avoid when defining a neural network model:

1. **Overfitting:** Model is too complex, performing well on training data but poorly on test data.
2. **Underfitting:** Model is too simple, failing to capture underlying patterns in data.
3. **Poor data preprocessing:** Failing to normalize, standardize, or transform data appropriately.
4. **Inadequate regularization:** Not using techniques like dropout, L1/L2 regularization to prevent overfitting.
5. **Incorrect activation functions:** Using inappropriate activation functions for specific layers or tasks.
6. **Insufficient training data:** Using too little data to train the model, leading to overfitting or underfitting.
7. **Inadequate hyperparameter tuning:** Failing to optimize hyperparameters, leading to suboptimal model performance.
8. **Ignoring bias-variance tradeoff:** Failing to balance model complexity and generalization.
9. **Not using transfer learning:** Failing to leverage pre-trained models for related tasks.
10. **Poor model evaluation:** Using inappropriate metrics or not evaluating on multiple datasets.
11. **Not handling class imbalance:** Failing to address class imbalance in classification tasks.
12. **Not using early stopping:** Failing to stop training when model performance plateaus.
13. **Using too many layers:** Adding unnecessary layers, leading to overfitting or vanishing gradients.
14. **Not using batch normalization:** Failing to normalize inputs to each layer.
15. **Not using appropriate optimization algorithms:** Using inappropriate optimizers for specific tasks.

By being aware of these common mistakes, you can design and train more effective neural network models.

Types of Neural Networks



Feedforward Neural Networks: A simple artificial neural network architecture in which data moves from input to output in a single direction. It has input, hidden, and output layers; feedback loops are absent. Its straightforward architecture makes it appropriate for a number of applications, such as regression and pattern recognition.

Multilayer Perceptron (MLP): A type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers, and an output layer. It uses nonlinear activation functions.

Convolutional Neural Network (CNN): A specialized artificial neural network designed for image processing. It employs convolutional layers to automatically learn hierarchical features from input images, enabling effective image recognition and classification.

Recurrent Neural Network (RNN): An artificial neural network type intended for sequential data processing. It is appropriate for applications where contextual dependencies are critical, such as time series prediction and natural language processing, since it makes use of feedback loops, which enable information to survive within the network.

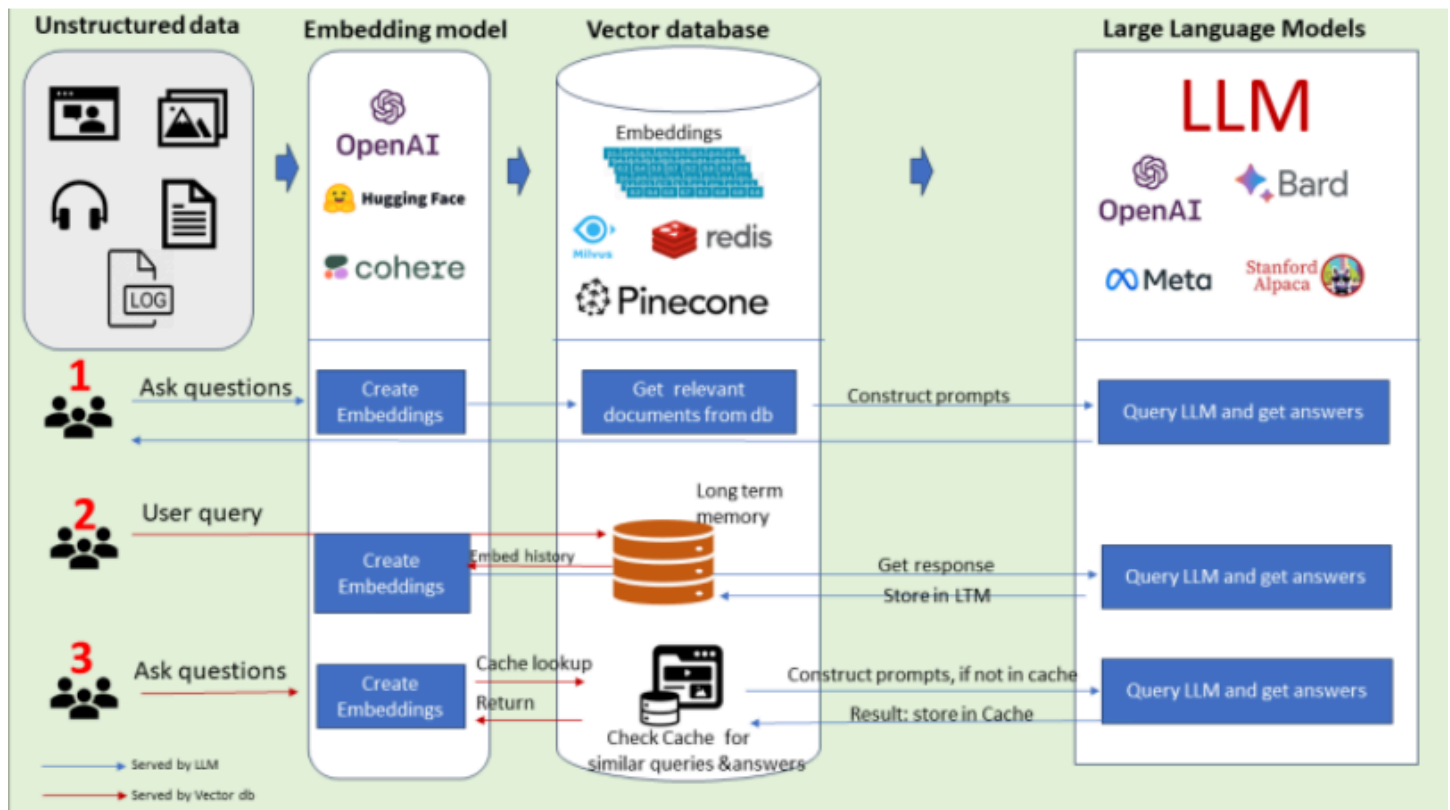
Long Short-Term Memory (LSTM): A type of RNN that is designed to overcome the vanishing gradient problem in training RNNs. It uses memory cells and gates to selectively read, write, and erase information.

Radial Basis Function Network: Comprises an input vector, an output layer with one node for each category, a layer of RBF neurons, and a layer of RBF neurons. The classification process involves comparing the input to examples from the training set, where each neuron has a prototype stored.

Sequence-to-Sequence Model: Two Recurrent Neural Networks create a sequence-to-sequence model. In this case, a decoder processes the output while an encoder processes the input. Working simultaneously, the encoder and decoder can use the same parameter or a different one.

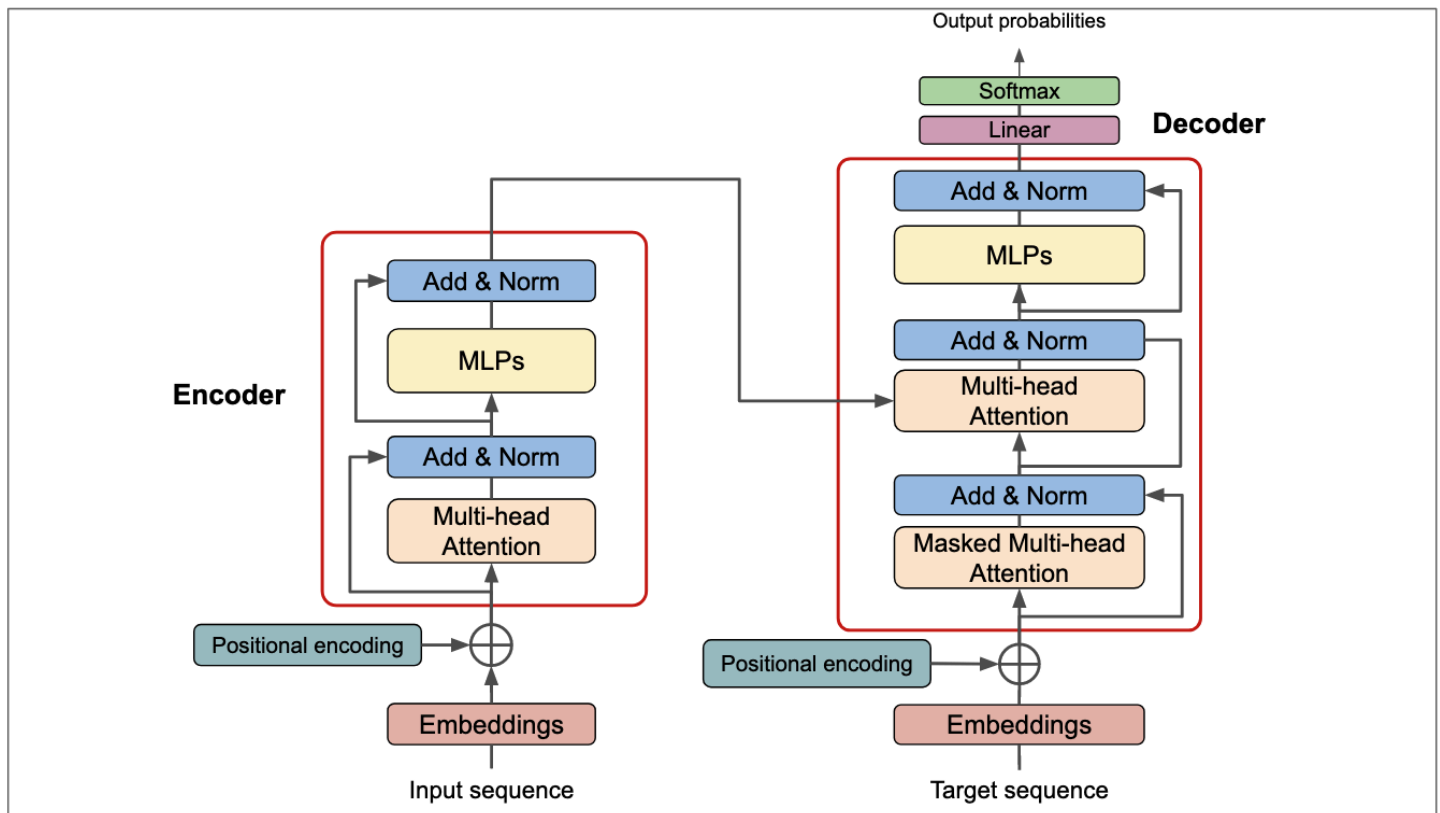
Modular Neural Network: Consists of several distinct networks that each carry out a specific task. Throughout the calculation process, there isn't much communication or interaction between the various networks. They each contribute separately to the outcome.

Gen AI



Source: <https://www.linkedin.com/pulse/3-ways-vector-databases-take-your-llm-use-cases-next-level-mishra/>

Transformers Architecture



Source: <https://deepprevious.github.io/posts/001-transformer/>

- Follows an encoder-decoder structure
- Does not rely on recurrence and convolutions in order to generate an output

The Encoder

- Consists of a stack of $N = 6$ identical layers
- Each layer is composed of two sublayers
- The first sublayer implements a multi-head self-attention mechanism
- The second sublayer is a fully connected feed-forward network consisting of two linear transformations with Rectified Linear Unit (ReLU) activation in between

The Decoder

- Consists of a stack of $N = 6$ identical layers
- Each layer is composed of three sublayers
- The first sublayer receives the previous output of the decoder stack, augments it with positional information, and implements multi-head self-attention over it
- The second layer implements a multi-head self-attention mechanism similar to the one implemented in the first sublayer of the encoder
- The third layer implements a fully connected feed-forward network, similar to the one implemented in the second sublayer of the encoder

Sum Up: The Transformer Model

- Each word forming an input sequence is transformed into a d_{model} -dimensional embedding vector
- Each embedding vector representing an input word is augmented by summing it (element-wise) to a positional encoding vector of the same d_{model} length, hence introducing positional information into the input
- The augmented embedding vectors are fed into the encoder block consisting of the two sublayers explained above
- The decoder receives as input its own predicted output word at time-step, $t - 1$

- The input to the decoder is also augmented by positional encoding in the same manner done on the encoder side
- The augmented decoder input is fed into the three sublayers comprising the decoder block explained above
- Masking is applied in the first sublayer in order to stop the decoder from attending to the succeeding words
- At the second sublayer, the decoder also receives the output of the encoder, which now allows the decoder to attend to all the words in the input sequence
- The output of the decoder finally passes through a fully connected layer, followed by a softmax layer, to generate a prediction for the next word of the output sequence

Types of Transformers

Aspect	BERT	GPT
Model Type	Transformer-based Encoder	Transformer-based Decoder
Directionality	Bidirectional	Unidirectional (Left-to-Right)
Layers	6 or 12 encoder layers	12 decoder Layers
Context Window	Fixed-length context	can vary depending on the length of the input text
Parameters	Over 100 million to 340 million parameters	117 million
Use Cases	NLP tasks like classification, NER, QA	Text generation, completion, conversation
Pre-training Objective	Masked Language Model (MLM)	Autoregressive Language Modeling
Fine-tuning	Task-specific layers added on top of BERT	Few-shot or one-shot task adaptation
Input Format	WordPiece or subword tokenization	Byte Pair Encoding (BPE) or similar
Contextual Embeddings	Used for both left and right context	Used for left context only

Source: <https://medium.com/@thirupathi.thangavel/bert-vs-gpt-comparison-a93037913cdc>

Autotransformers: used for both step-up and step-down applications, autotransformers contain only one winding, with a portion of the coil serving as both the primary and secondary winding

- Industrial control transformers: change supply voltages for electromagnetic devices such as contractors, solenoids, relays and timers

Isolation transformers: a transformer with a separation/barrier between the primary and secondary windings

Power transformers: a transformer that is powering up an application, rather than powering down or stepping the voltage down

Toroidal transformers: donut-shaped transformers that save space compared to E-I cores, and may reduce external magnetic field

Polyphase transformer: a transformer for polyphase systems, multiple single-phase transformers can be used, or all phases can be connected to a single polyphase transformer

Grounding transformer: lets three wire (delta) polyphase system supplies accommodate phase to neutral loads by providing a return path for current to a neutral

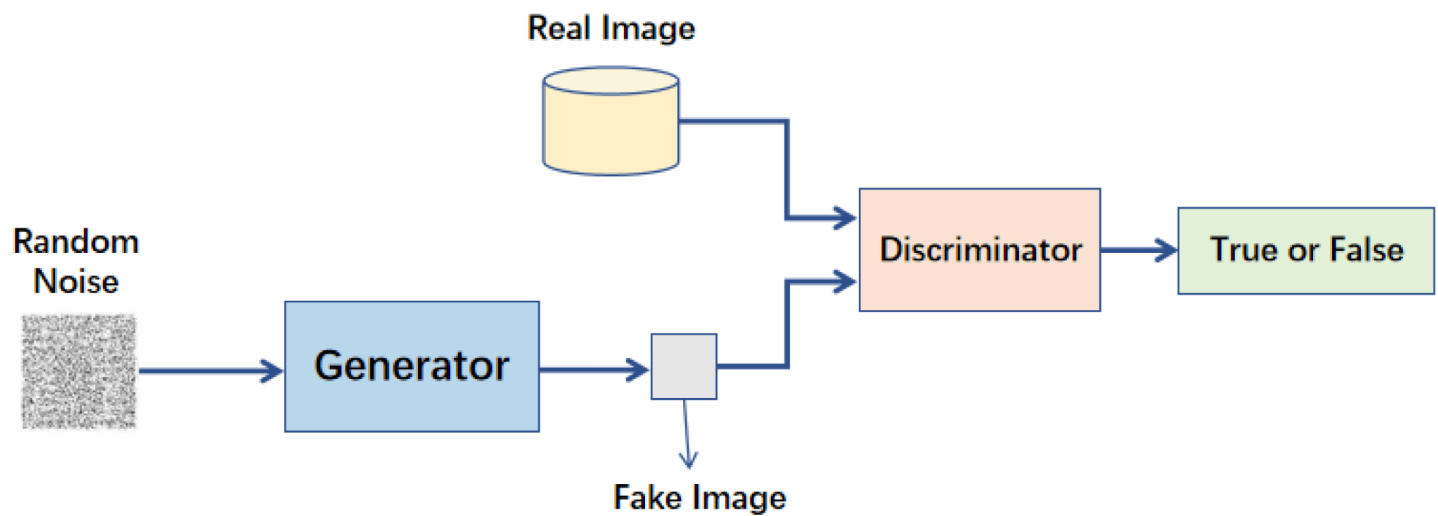
Phase-shifting transformer: a specialized type of transformer that can be configured to adjust the phase relationship between input and output

Variable-frequency transformer: a specialized three-phase power transformer that allows the phase relationship between the input and output windings to be continuously adjusted by rotating one half

Leakage transformer: a transformer with a significantly higher leakage inductance than other transformers, sometimes increased by a magnetic bypass or shunt in its core between primary and secondary, which is sometimes adjustable with a set screw

Resonant transformer: a transformer in which one or both windings has a capacitor across it and functions as a tuned circuit

Adversarial Networks



Source: <https://www.mdpi.com/2078-2489/14/10/575>

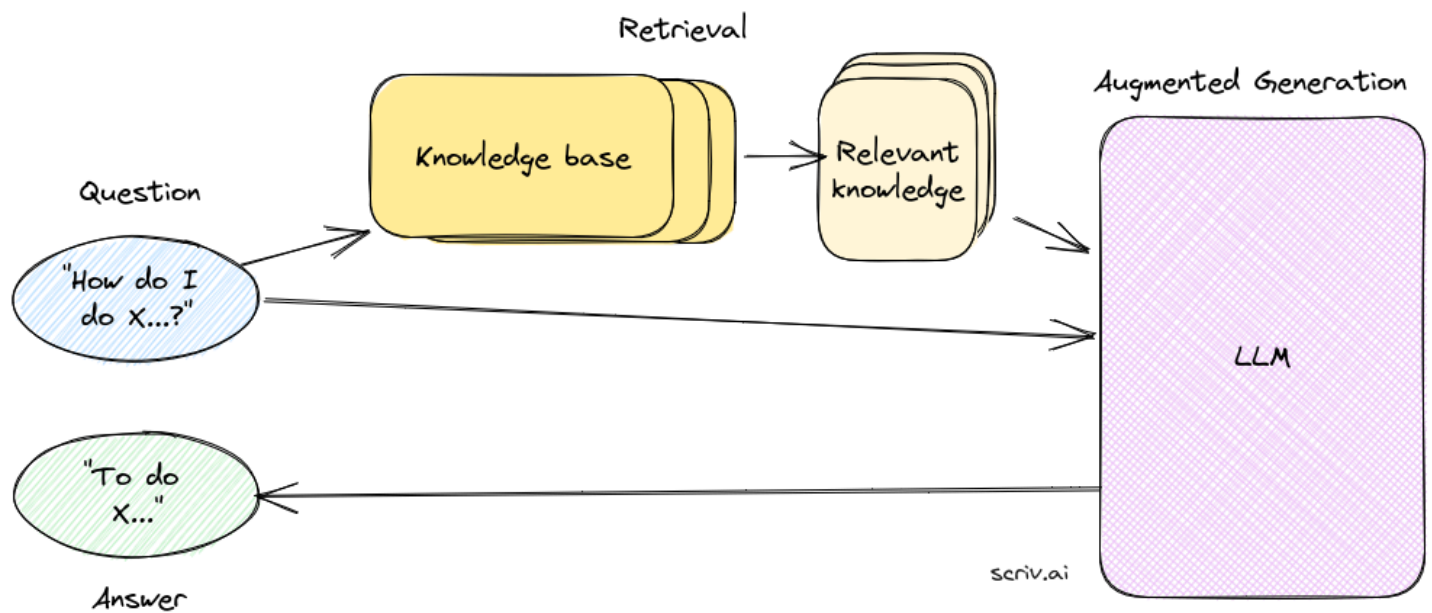
Efficient generative adversarial networks: These networks use linear additive-attention transformers. They are built upon a novel transformer block named Ladaformer, which reduces computational complexity and overcomes training instabilities often associated with transformer GANs.

Generative Adversarial Networks (GANs): These networks combine a generator and a discriminator. The generator creates content, while the discriminator identifies authentic versus counterfeit images.

Transformer-based Generative Adversarial Networks: These networks are used for image generation, image-to-image translation, video synthesis, and other applications. They have shown tremendous performance improvement for several problems in computer vision.

GANsformers: These networks combine GANs and transformers. They use a transformer to provide an attentional reference so the generator can increase the use of context to enhance content.

RAG



Source: <https://blog.roboflow.com/what-is-retrieval-augmented-generation/>

Retrieval-Augmented Generation (RAG) is a technique used to improve the accuracy and reliability of generative AI models by combining internal knowledge with facts fetched from external sources ¹. Here are some key points and examples about RAG in generative AI ^{1 2}:

Combining Internal and External Resources: RAG links generative AI services to external resources, especially ones rich in the latest technical details, to provide more accurate and up-to-date information.

Building User Trust: RAG gives models sources they can cite, like footnotes in a research paper, so users can check any claims, which builds trust and reduces the possibility a model will make a wrong guess.

Easy Implementation: Developers can implement RAG with as few as five lines of code, making it faster and less expensive than retraining a model with additional datasets.

Applications: RAG has various applications, such as:

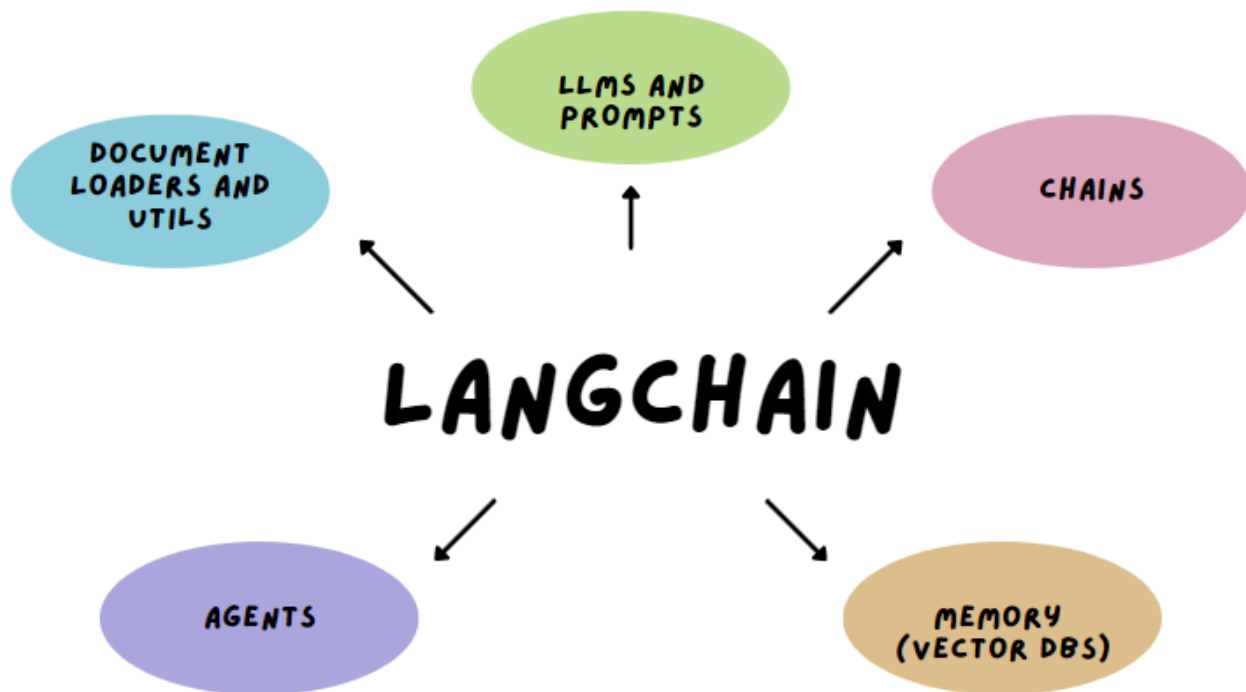
A generative AI model supplemented with a medical index could be a great assistant for a doctor or nurse.

Financial analysts would benefit from an assistant linked to market data.

Almost any business can turn its technical or policy manuals, videos or logs into resources called knowledge bases that can enhance LLMs.

Companies Adopting RAG: Companies including AWS, IBM, Glean, Google, Microsoft, NVIDIA, Oracle and Pinecone are adopting RAG due to its potential to enhance generative AI capabilities.

LangChain



Source: <https://shurutech.com/wp-content/uploads/2023/06/LangChain-Components.webp>

LangChain is a framework for developing applications powered by large language models (LLMs) ^{1 2}. LangChain simplifies every stage of the LLM application lifecycle: development, productionization, and deployment ². Here are some of the key points about LangChain ^{1 3}:

LangChain Libraries: Python and JavaScript libraries that contain interfaces and integrations for a myriad of components, a basic run time for combining these components into chains and agents, and off-the-shelf implementations of chains and agents.

LangChain Templates: a collection of easily deployable reference architectures for a wide variety of tasks.

LangServe: a library for deploying LangChain chains as a REST API.

LangSmith: a developer platform that lets you debug, test, evaluate, and monitor chains built on any LLM framework and seamlessly integrates with LangChain.

Components: composable tools and integrations for working with language models.

Off-the-shelf chains: built-in assemblages of components for accomplishing higher-level tasks.

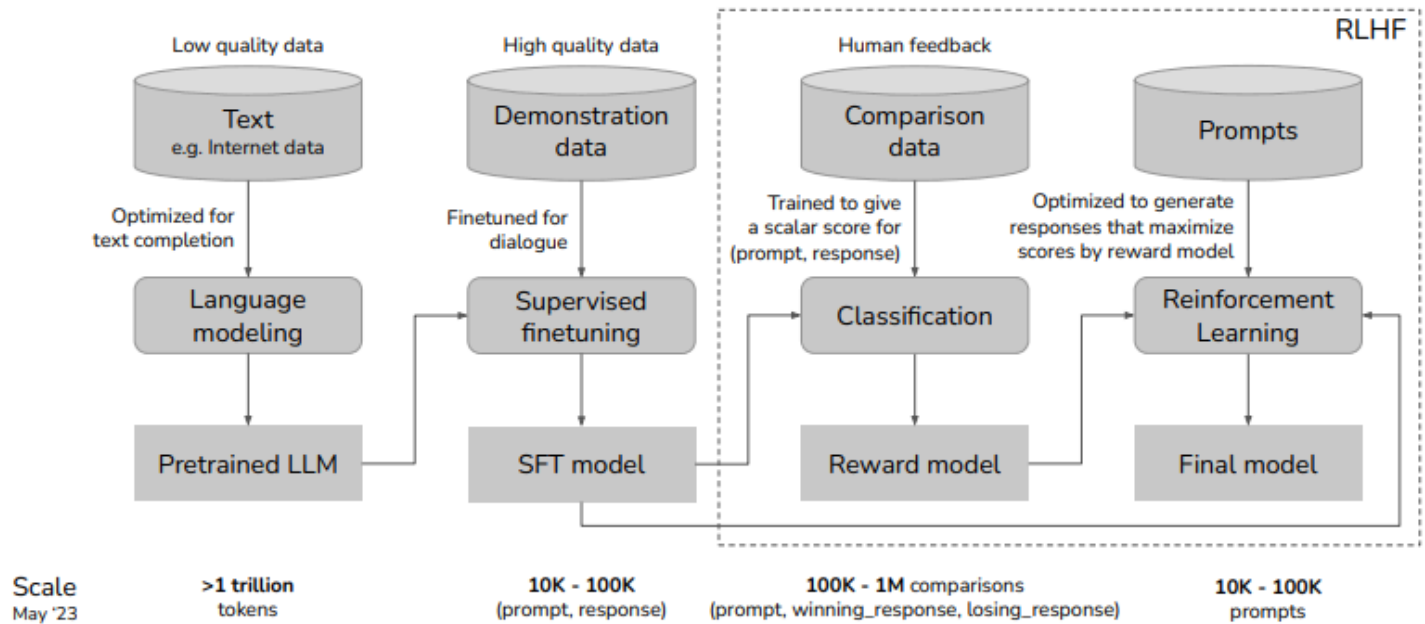
LangChain Expression Language (LCEL): a declarative way to compose chains. LCEL was designed from day 1 to support putting prototypes in production, with no code changes, from the simplest “prompt + LLM” chain to the most complex chains.

Modules: LangChain provides standard, extendable interfaces and integrations for the following modules: Model I/O, Interface with language models, Retrieval, Interface with application-specific data, Agents, Let models choose which tools to use given high-level directives.

Use cases: LangChain provides walkthroughs and techniques for common end-to-end use cases, such as Document question answering, RAG Agents, etc.

Integrations: LangChain is part of a rich ecosystem of tools that integrate with the framework and build on top of it.

Training LLMs



Source:

<https://www.analyticsvidhya.com/blog/2023/07/beginners-guide-to-build-large-language-models-from-scratch/>

The time it takes to train a large language model (LLM) depends on a variety of factors, including the model architecture, training dynamics and methods for optimizing training performance ¹. Here are some other factors that can influence the training time ^{2 3}:

Computational power: LLM models and data's sheer volume and scale require substantial computational power.

Data collection and preparation: The initial step involves seeking out and compiling a training dataset.

Model configuration: Transformer deep learning frameworks are commonly used for Natural Language Processing (NLP) applications.

Infrastructure: LLMs are trained on huge text corpora, typically at least 1000 GB in size.

Model parallelism: Distributes parts of the model over multiple Graphics Processing Units (GPUs).

Fine-tuning: Once the training is done, the model is evaluated using a testing dataset to gauge its performance.

For example, training GPT-3 (175 billion parameters) with training complexity in the order of 3.14×10^{23} , will take 314,000 seconds of Exaflop machine's compute cycles to complete training with 300 billion tokens. 300,000 seconds is approximately 3.5 days ³.

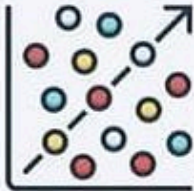
Traditional AI

SKILL<S/ASH>

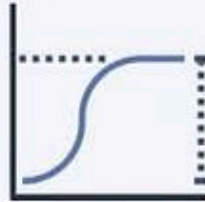


ESSENTIAL MACHINE LEARNING ALGORITHMS

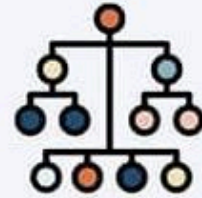
Linear
Regression



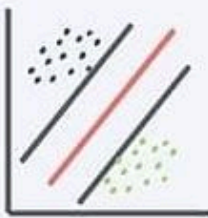
Logistic
Regression



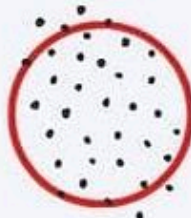
Decision Tree



SVM



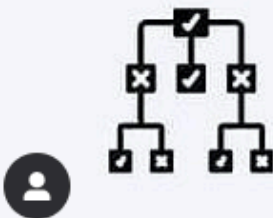
KNN



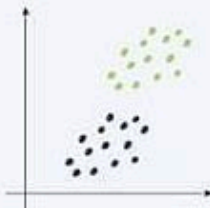
Dimensionality
Reduction



Random Forest



K-means

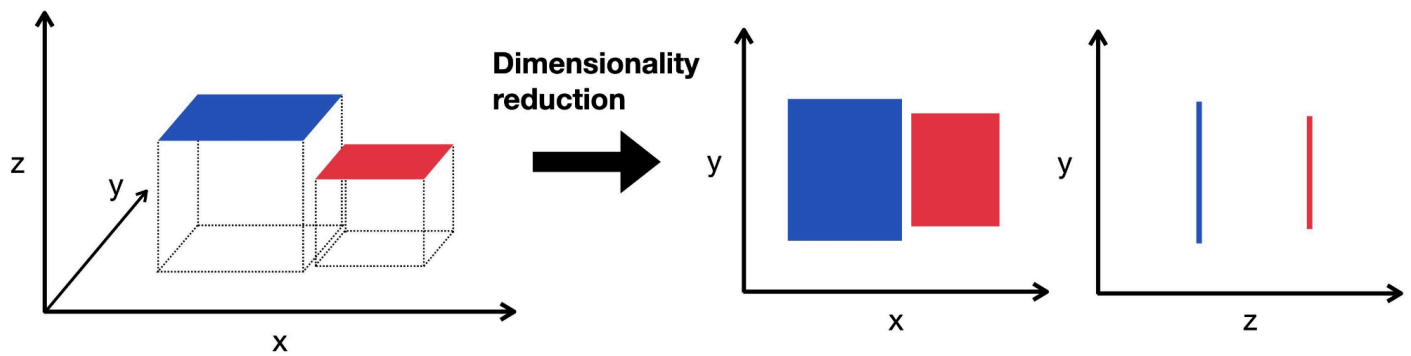


Naive Bayes



Source: <https://twitter.com/danieluriol/status/1443411036820885505>

Dimensionality reduction algorithms



Source: https://www.sc-best-practices.org/preprocessing_visualization/dimensionality_reduction.html

Dimensionality reduction algorithms are used to reduce the number of input variables in the dataset, and the differences between PCA, SVD, t-SNE, and other dimensionality reduction algorithms are as follows ^{1 2}:

Principal Component Analysis (PCA)

- PCA is a dimensionality reduction technique widely used in data analysis and machine learning.
- Its primary goal is to transform high-dimensional data into a lower-dimensional representation, capturing the most important information.
- PCA is a linear dimensionality reduction technique.
- It tries to preserve the global structure of the data.
- It does not involve hyperparameters.
- It gets highly affected by outliers.

Linear Discriminant Analysis (LDA)

- LDA serves as a technique for both dimensionality reduction and classification, aiming to optimize the distinction between various classes within a dataset.
- LDA is particularly prevalent in supervised learning scenarios where the classes of data points are predetermined.
- LDA computes "linear discriminants," determining the directions that serve as axes to maximize separation between multiple classes.
- LDA is a supervised algorithm that maximizes class separation.

Singular Value Decomposition (SVD)

- SVD is a factorization technique that decomposes a matrix into three matrices.
- It is used for dimensionality reduction, image compression, and data imputation.
- SVD is a more general form of PCA, and it can be used for non-square matrices.

t-distributed Stochastic Neighbor Embedding (t-SNE)

- t-SNE is a non-linear dimensionality reduction and data visualization technique.
- It embeds the points from a higher dimension to a lower dimension trying to preserve the neighborhood of that point.
- t-SNE tries to preserve the local structure of data by minimizing the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map.
- t-SNE is a non-linear dimensionality reduction technique.

Comparison of Dimensionality Reduction Algorithms

- PCA is a linear technique that tries to preserve the global structure of the data, while t-SNE is a non-linear technique that tries to preserve the local structure of the data.
- LDA is a supervised algorithm that maximizes class separation, while PCA and t-SNE are unsupervised algorithms.
- SVD is a more general form of PCA and can be used for non-square matrices.

When to Use Each Algorithm


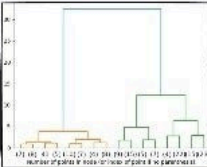
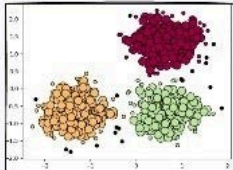
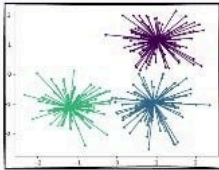
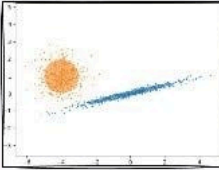
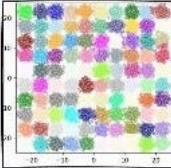
- PCA is suitable for high-dimensional data with linear relationships between variables.
- t-SNE is suitable for high-dimensional data with non-linear relationships between variables.
- LDA is suitable for classification tasks with a small number of classes.
- SVD is suitable for image compression and data imputation.

Clustering Algorithms

6 Types of Clustering Algorithms in Machine Learning



blog.DailyDoseofDS.com

Clustering Algorithm Type		Clustering Methodology	Algorithm(s)
	Centroid-based	Cluster points based on proximity to centroid	KMeans KMeans++ KMedoids
	Connectivity-based	Cluster points based on proximity between clusters	Hierarchical Clustering (Agglomerative and Divisive)
	Density-based	Cluster points based on their density instead of proximity	DBSCAN OPTICS HDBSCAN
	Graph-based	Cluster points based on graph distance	Affinity Propagation Spectral Clustering
	Distribution-based	Cluster points based on their likelihood of belonging to the same distribution.	Gaussian Mixture Models (GMMs)
	Compression-based	Transform data to a lower dimensional space and then perform clustering	BIRCH

Source: https://twitter.com/_avichawla/status/1770374547830641091

Hierarchical clustering: uses a tree-like structure to cluster data points.

Agglomerative clustering: a bottom-up approach that begins with each element as a separate cluster and merges them into successively larger clusters.

Divisive clustering: a top-down approach that begins with the whole set and divides it into successively smaller clusters.

Partitioning clustering: divides data into a specified number of clusters.

K-Means clustering: divides objects into clusters based on similarities and are dissimilar to the objects belonging to another cluster.

Fuzzy C-Means: similar to K-Means, but objects can belong to more than one cluster.

K-Means clustering is a popular algorithm for partitioning data into clusters based on similarities. It begins with an initial set of random centroids and iteratively updates the centroids and reassigns data points to clusters until convergence. The algorithm is sensitive to initial centroids and requires specifying the number of

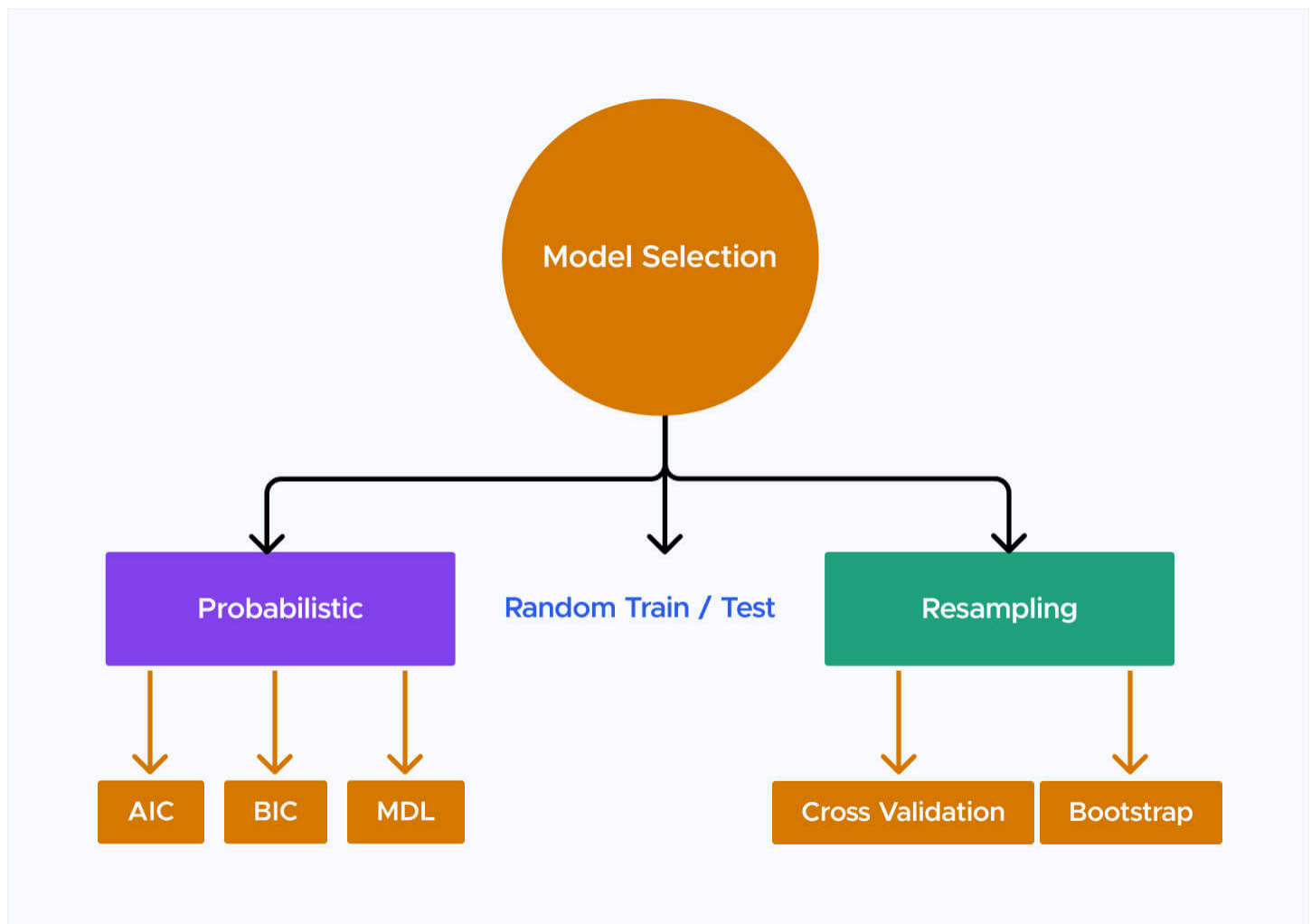
clusters, but it is efficient and scalable. K-Means is widely used in applications such as customer segmentation, image compression, and search engines.

The main difference between K-Means and Fuzzy C-Means is that K-Means assigns each data point to a single cluster, while Fuzzy C-Means allows data points to belong to multiple clusters with varying degrees of membership. This makes Fuzzy C-Means more suitable for datasets with overlapping clusters or noisy data. However, Fuzzy C-Means is computationally more expensive than K-Means and requires careful choice of parameters.

Hierarchical clustering, on the other hand, builds a tree-like structure of clusters, allowing for visualization and exploration of data at different levels of granularity. Agglomerative clustering starts with individual data points and merges them into clusters, while divisive clustering starts with the whole dataset and divides it into smaller clusters. Hierarchical clustering is useful for identifying nested patterns or relationships in data, but it can be computationally expensive for large datasets.

In summary, K-Means is a widely used algorithm for partitioning data into clusters, while Fuzzy C-Means is a variant that allows for fuzzy membership. Hierarchical clustering builds a tree-like structure of clusters, allowing for exploration and visualization of data at different levels of granularity. The choice of algorithm depends on the specific problem, dataset, and desired outcome.

Model Selection



Source: <https://censius.ai/blogs/machine-learning-model-selection-techniques>

Model selection algorithms are used to choose the best model for a given dataset and problem. Here are some comparisons of different model selection algorithms

Grid Search: This method performs an exhaustive search over a specified hyperparameter space, evaluating the model's performance at each point.

Random Search: This method samples the hyperparameter space randomly, evaluating the model's performance at each point.

Cross-Validation: This method splits the data into training and validation sets, evaluating the model's performance on the validation set.

K-Fold Cross-Validation: This method splits the data into K folds, training the model on K-1 folds and evaluating on the remaining fold.

Leave-One-Out Cross-Validation: This method trains the model on all but one instance, evaluating on that instance.

Bayesian Optimization: This method uses a probabilistic approach to search for the optimal hyperparameters.

Gradient-Based Optimization: This method uses gradient descent to search for the optimal hyperparameters.

Genetic Algorithm: This method uses an evolutionary approach to search for the optimal hyperparameters.

Simulated Annealing: This method uses a temperature schedule to control the exploration of the hyperparameter space.

Metrics used for model selection:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- R-Squared
- Accuracy
- F1 Score
- Area Under the ROC Curve (AUC)

Amount of Data

The amount of data needed for training AI models depends on various factors, and the notion that "more data is always better" is a common misconception. Here's a breakdown of when more or less data is beneficial:

More data is beneficial when

1. Model complexity is high: Complex models require more data to learn and generalize well.
2. Data variability is high: More data helps capture diverse patterns and relationships.
3. Noise and outliers are present: More data can help average out noise and reduce the impact of outliers.
4. Overfitting is a concern: More data can help prevent overfitting by providing a larger sample size.

Less data is beneficial when

1. Model complexity is low: Simple models can learn effectively with smaller datasets.
2. Data is highly curated and representative: Small, high-quality datasets can be more effective than large, noisy ones.
3. Labeling or annotation is expensive: In cases where labeling data is time-consuming or costly, smaller datasets may be more efficient.
4. Model interpretability is important: Smaller datasets can facilitate model interpretability and understanding.

When to use less data

1. Prototyping and initial exploration: Start with a smaller dataset to quickly test ideas and iterate.
2. Resource constraints: When computational resources or storage are limited, smaller datasets may be necessary.
3. Domain adaptation: When adapting a model to a new domain, a smaller dataset from the target domain can be more effective.

When to use more data

1. Large-scale industrial applications: In production environments, larger datasets can improve model performance and robustness.
2. Research and development: In research settings, larger datasets can help identify patterns and relationships that may not be apparent with smaller datasets.
3. High-stakes applications: In critical applications like healthcare or finance, larger datasets can help ensure model accuracy and reliability.

About This Experiment

- Using AI to Learn AI
- I used AI writing tools to write this essay.
- Mainly LLAMA2 and LLAMA 3.
- There was no single prompt that generated all this; it was a bunch of multiple prompts.
- Images and references are all from the web, not from AI writing.
- This is a work in progress; I will keep adding content.