Introduction to Generative AI and LLMs

Dunstan Matekenya, PhD
Data Scientist
The World Bank Group
Washington DC

Expected Learning Outcomes

Key Concepts that Participants Should Be Able to Explain, Understand and Appreciate

01 Understand Key Concepts In Artificial Intelligence

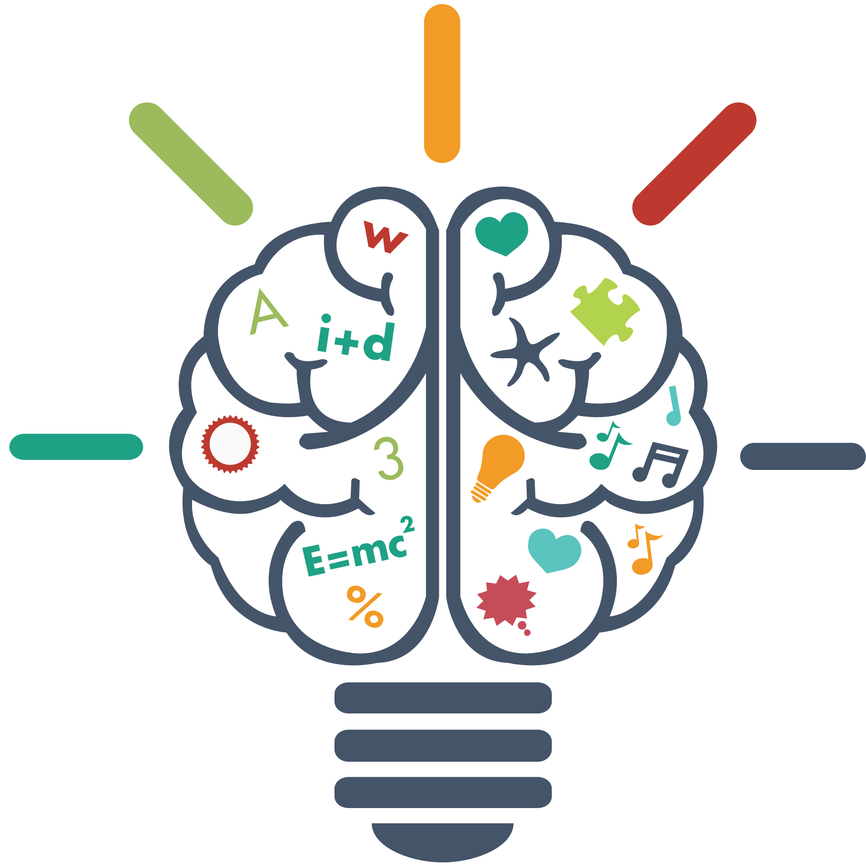
Generative AI vs. predictive AI; how are AI models built; how do you use AI; what are some of the major use cases of AI; what are important questions to ask about AI systems

02 Understand Essential Concepts In Large Language Models (LLMs)

language modelling and the main task involved; the role of language models in the broader NLP space (tasks); how language models are created; architecture behind modern LLMs; main characteristics of LLMs; how to evaluate LLMs; word embedding, vector database and their role in LLM apps; LLM main capabilities ; Different metrics for different LLM tasks; LLM challenges and biases; different ways to adapt and customize LLMs; key factors for comparing different LLMs

03 Accessing LLMs and Developing LLM Based Projects

How to use Hugging Face to access LLMs and why; main considerations when selecting LLM model for a project; the main steps in an LLM based project; the role of frameworks such as LangChain in LLM app development; what are the best practices in deploying LLM projects



Summary of Contents

1. Introducing Generative AI (Gen AI)

1. What is Gen AI?
2. Gen AI vs. Traditional ML
3. Evolution of Gen AI
4. Overview of Gen AI Capabilities
5. Important Characteristics of AI Models

What is Generative AI

- **Generative AI (Gen AI) refers to a class of Machine Learning (ML) models which can generate new content**
- **Given a prompt or instruction as a text, image or another media, a Gen AI system will produce a response**

Text Content

- Generate content from scratch (e.g., write essay)
- Summarize existing content
- Answer questions based on documents

Images

- Create images of different types
- Query images

Videos

- Create videos given text prompt
- For example, movie scripts

Audios

- Generate audio given text prompt
- For example, music or synthetic voices

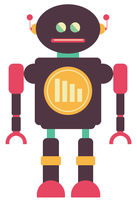
Code

- Create code in different programming languages (e.g., Python,

How Does Gen AI Work

Prompt

Give me a course outline to teach the basics of Gen AI to members of parliament



Model

Meta AI

Meta on WhatsApp or Facebook

Chat-GPT

Gemini Advanced

Google Gemini

Response

Course Outline:

I. Introduction to Artificial Intelligence (AI)

- Definition of AI
- Brief history and evolution of AI
- Key concepts: machine learning, deep learning, natural language processing, etc.
- Real-world applications of AI in various sectors (e.g., healthcare, finance, transportation)

II. AI Ethics and Governance

- Importance of ethical considerations in AI development and deployment
- Ethical dilemmas: bias, privacy, transparency, accountability
- Regulatory frameworks and international standards for AI governance
- Case studies of AI ethics and governance failures and successes

III. AI and Policy Making

- How AI can inform evidence-based policymaking
- Leveraging AI for efficient resource allocati and decision-making

Detailed Generative AI Capabilities

Detailed Description of Gen AI Capabilities

Text Generation

- Content Creation. Writing articles, stories, essays, and reports.
- Conversational AI. Powering chatbots and virtual assistants to engage in human-like conversations.
- Language Translation. Translating text between different languages.
- Summarization. Condensing long documents into shorter summaries.
- Text Completion. Predicting and completing sentences or paragraphs based on a given prompt.

Image Generation and Manipulation

- Text-to-Image generation. Create image based on textual description or based on abstract ideas or concepts
- Art and design creation. Creating original artwork, designs, and illustrations.
- Photo Editing. Enhancing or altering photos, including colorization and style transfer.
- Image enhancement. Enhancing the resolution of images to add more detail. Denoise images.
- Image Synthesis. Generating realistic images from textual descriptions or sketches (e.g., DALL-E).
- Background Replacement
- Colorization

Video Generation and Editing

- Text-to-Video Generation. Description based and concept based video generation.
- Animation. Creating animated sequences from scripts or storyboards.
- Video editing. Cutting and splicing and color correction.
- Deepfakes. Generating realistic videos by swapping faces or altering speech.
- Video Summarization. Creating condensed versions of longer videos.

Detailed Generative AI Capabilities-cont'd

Detailed Description of Gen AI Capabilities

Code Generation

- Code completion. Predicts or completes particial code, useful in IDE
- Code generation from natural language. Converts comments and natural language descriptions into functional code.
- Code translation. Translates code between different programming languages.
- Documentation generation. Automatically generates documentation for code, including comments and usage examples.
- Debugging and Error Fixing. Identifies bugs and suggests fixes

Data Generation and Simulation

- Synthetic data generation-image, text, audio and video data. Generate data for ML model training and other uses.
- Synthetic data generation-tabular data. Generate synthetic data for statistical and other tabular datasets.
- SiPredicts or completes particial code, useful in IDE

Audio Generation and Processing

- Music Composition.Creating original music tracks in various genres.
- Speech to text (STT). Convert speech to text
- Speech Synthesis: Generating natural-sounding human speech (e.g., text-to-speech systems).
- Sound Effects. Producing custom sound effects for media production.

Main Components of a Gen AI System

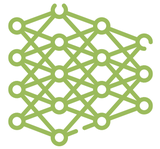
What Makes Up an AI System

Collect and Process data

Openly available data from the internet (images, audios, videos, books etc) or proprietary datasets.

Data

Model Architecture



Model design and architectures

LLM (e.g., GPT-4, BERT), image generation models (e.g., StableDiffusion) or speech models (e.g., Whisper)

Gen AI

Model training, evaluation and validation

The process of using data to train the model, evaluate it and ensure its performing as expected

Model Dev

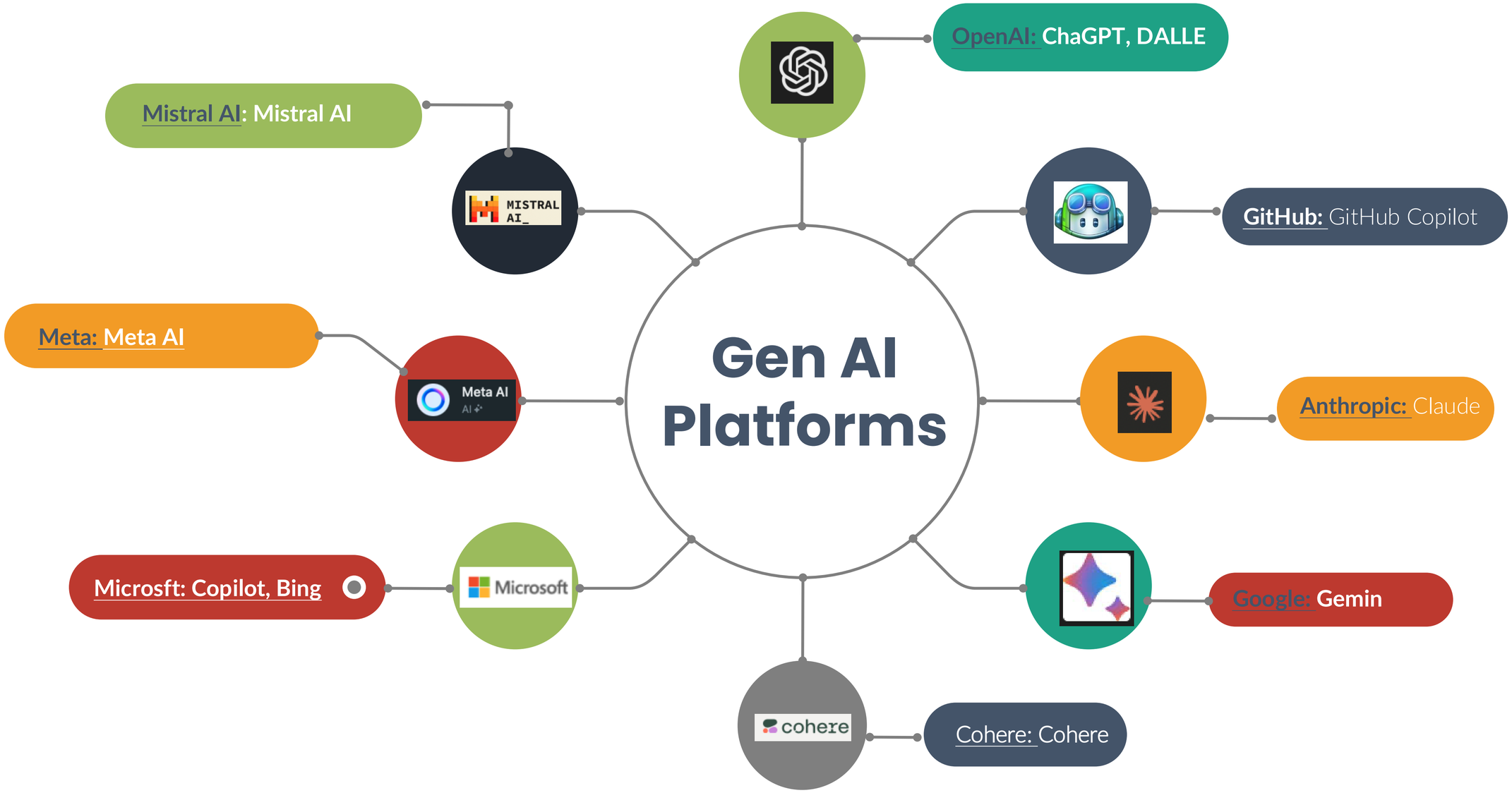
Deployment and Apps

APIs and interfaces

Model is deployed via APIs or integrated into applications, enabling end-users to interact with the generative AI system (e.g., chatGPT, Meta AI on WhatsApp, Google Gemeni or company specific platform

Major Gen AI Platforms

Popular Gen AI, LLM Platforms, Tools and Companies



Factual Accuracy, Correctness and Currency of Gen AI Outputs Depend on Several Factors

Training Data

The accuracy of AI responses depends heavily on the data the model was trained on. If the training data contains inaccuracies or biases, the AI might reflect those in its outputs.

Hallucinations, toxicity and other errors

Hallucinations is when an AI models presents results as factually correct when they are demonstrably inaccurate or wrong. Models have other errors such as failure to perform calculations correctly, presenting historical facts inaccurately and more. For toxicity, the model provides which maybe offensive to some groups

Information currency, real-time information

AI models have a cutoff date for the information they were trained on. Thus, they do not provide real-time information

Prompt specificity

More specific prompts can help guide the AI to produce more accurate and relevant information. Vague or ambiguous prompts might lead to less accurate responses.

Randomness in generated outputs

All Gen AI models are stochastic in nature as their core capability is to predict the next token based on probabilities. As such, give the same prompt, an AI systm can provide a slightly different response. For ChatGPT, there is a parameter called temperature which controles the level of randomness and creativity in ChatGPT responses.

Additional Factors to Consider when Implementing Gen AI Apps in an Organization,

Cost

Open Source platforms vs commercial; compute costs for inference; compute costs for fine-tuning;

Data sovereignty and privacy

When users interact with the Gen AI system, where does the data go? Is the data being used to train the models

Biases, toxicity and other errors

Depending on who the end users of the AI system are, its important to check and understand if the system is prone to discriminate other groups or produce toxic content which maybe inappropriate for certain groups

Explore Meta AI and Chat-GPT Image Generators

Prompt- Create image an African woman carrying a bucket of water

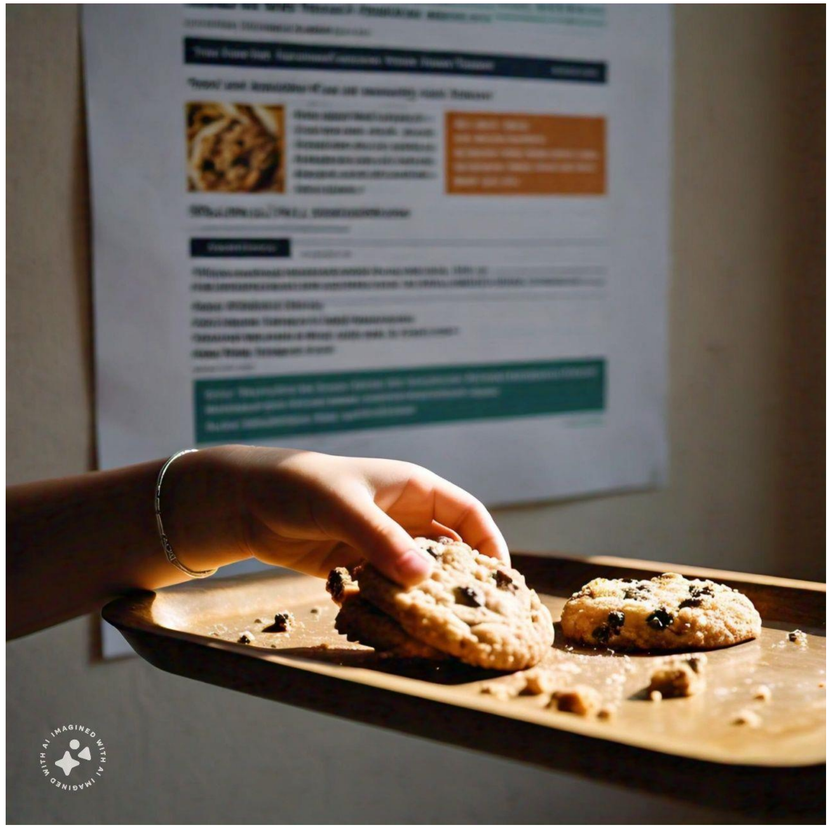
ChatGPT 4o (DALLE 3)

Meta AI on WhatsApp

Explore Meta AI and Chat-GPT Image Generators

Prompt-I need an image to use as a poster for a report about survey results on children health and nutrition

ChatGPT 4o (DALLE 3)



Meta AI on WhatsApp

How Are Gen AI Systems Being Used in Industries

A summary of different ways AI can be used

Content Generation

1. Generate text content (e.g., product advertisements, product descriptions, and personalized recommendations)
2. Generate video, image, arts and ither creative content. For example, help movie makers
3. Draft/write essays, emails, novels
4. Translation across different languages

Chatbots and Virtual Assistants

1. Democratize access to information by enabling use of natural language to access complicated technical docs. For example, a chatbot which can allow farmers to ask questions from agriculture document)
2. Improve access to services and service delivery. For example, a chatbot to answer questions about taxes

Prediction, automation and system optimization

1. In medicine, AI can be used to help doctors understand medical imagery better (e,g., TB diagnosis)
2. In pharmaceuticals, AI can assist in the discovery and design of new drugs by generating novel molecular structures.
3. In finance, telecom and other sectors AI models can help with targeting products (recommendation) to the right customers
4. In robotics, AI models can help in training robots

How Are Gen AI Systems Being Used in Industries-cont'd

A summary of different ways AI can be used

Logical Reasoning Engines

1. LLMs can perform natural language understanding (NLU) tasks
2. For example, sentiment analysis of product reviews or other textual information can be achieved with LLMs
3. This NLU gives LLMs the ability to take a vague instruction and break it down into smaller tasks using techniques such as chain of reasoning

Text Retrieval and Search

1. Search through large databases or the Internet to locate relevant information based on user-defined queries.
2. LLMs can be combined with traditional search algorithms to provide users with better search experience

Language Translation

1. Text translation across multiple languages is easier and more accurate with LLMs
2. Majority of pre-trained LLMs can be fine-tuned to perform machine translation (MT) task very wll

Artificial Intelligence(AI)

A simplified look at major areas

GENERATIVE AI

Creating new and original content (e.g., images, text) by learning from existing data patterns using LLMs

- **Text** — chatGPT
- **Images** — DALLE-2
- **Videos** — Sora
- **Audio** — Sora

PREDICTIVE AI

Build models using historical data and make predictions

- A commercial bank using models to decide whether to give a loan

Generative AI Vs. Predictive AI

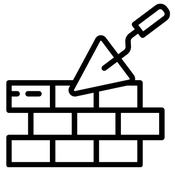
Key Differences and Similarities

Generative (Gen AI)	Vs.	Predictive AI
• Typical Gen AI model tasks include text generation, image generation and more.		• Predictive AI models can also be thought of as discriminative AI as they only discriminate possible output classes (e.g., dog vs. cats) or make predictions (e.g., predict house price or stock price).
• Often have large foundation models which are trained as general purpose models. For example, LLMs are general purpose next token prediction models which can be fine-tuned to perform more specialized tasks (e.g., translation, conversation).		• Mostly trained on a narrow specific tasks. For example, a model to predict whether a customer applying for a loan will likely pay back. Or predictions meant to recommend products to a customer.
• Trained on massive amounts of data, sometimes multi-modal (text, audio, images)		• Trained on relatively smaller datasets and usually with single mode (e.g., customer records)
• Have creative capabilities		• No creative capabilities
• Consists of multiple family of both predictive and generative AI models working together		• Often only a single model. If multiple models (ensemble), they are all predictive models

Evolution of Gen AI

How Did We Get Here?

1950s-1960s | 1980-2000 | 2000-2010 | 2010 - 2017 | 2017-Present

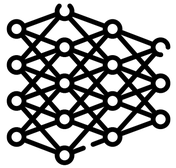


Early Foundations

- Introduction of the term Artificial Intelligence at Dartmouth College
- First NLP computer program is created

Classical ML

- Mostly rule based ML and expert systems
- Advanced ML techniques such as NN, LR, KNN and other emerge
- Some companies start using ML at scale

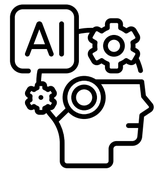


Emergence of Deep Learning

- The power of deep neural networks for image recognition is demonstrated by Jeff Hinton
- Google begins work on deep learning

Breakthroughs in NLP, Image Recognition

- Success of convolutional neural networks (CNN) in image recognition.
- Breakthroughs in NLP using RNN, LSTMS
- Introduction of Generative Adversarial Networks (GANS)
- OpenAI is founded



LLMs and Gen AI

- Introduction of BERT improving various NLP tasks
- Release of transformer model in 2017
- Introduction of vector embeddings and word to vector
- Open AI release GPT-2 in 2019

2.Essential Concepts in LLMs

1. The Language Modelling Task
2. LLMs Development-Transformer Architecture
3. Categorizing LLMs
4. Summary of LLM capabilities

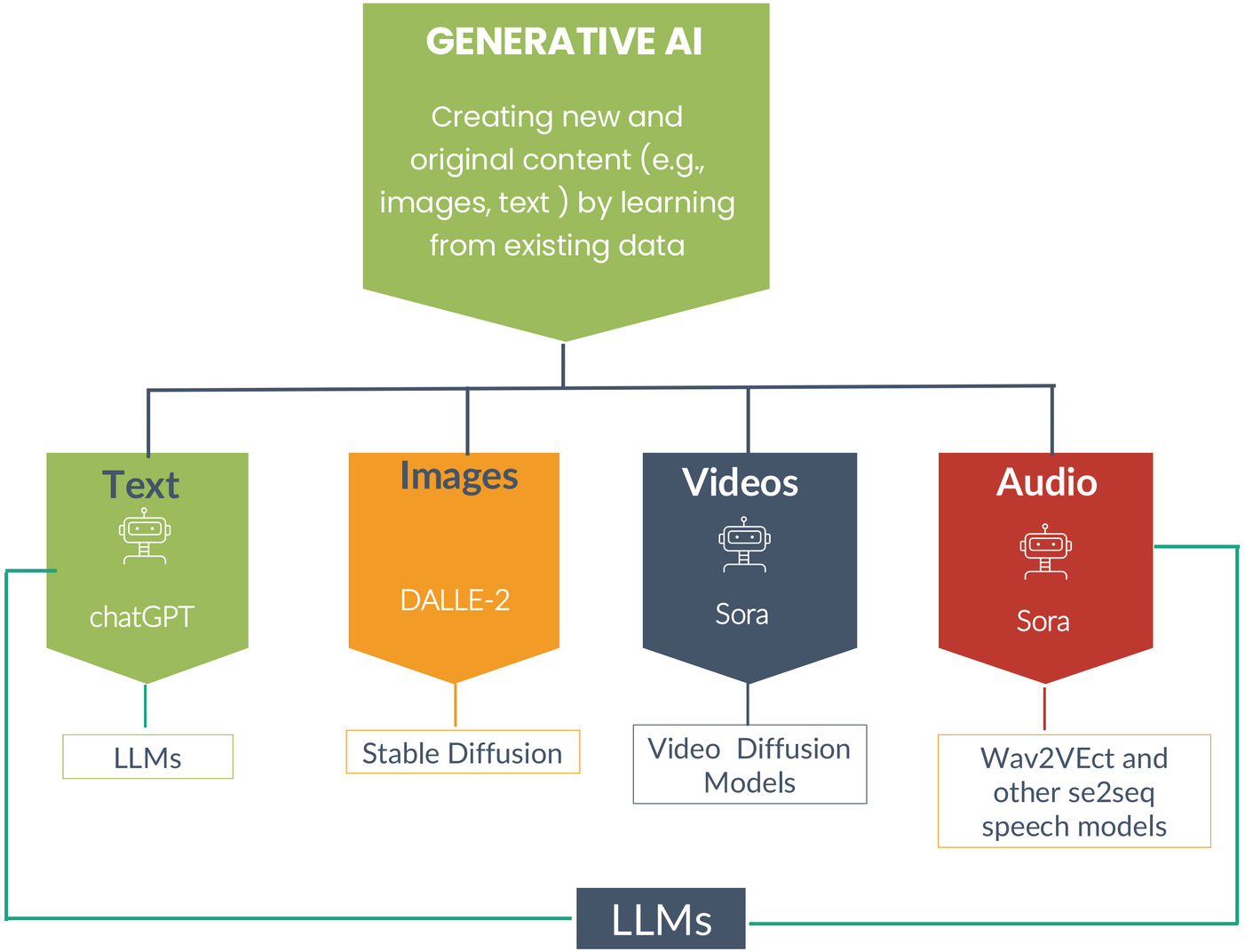
Key Terms in LLMs

Common Terms and Concepts

Language Model	A language model is a ML model that aims to predict and generate plausible language. Autocomplete is a language model, for example.
Parameters	After the process of model training, model parameters or weights are what the model learns from data and are used during inference
Architecture	A neural network architecture refers to a specific design of a neural network (e.g., RNN, LSTM
Transformer	A type of neural network architecture designed to process sequential data non-sequentially and is the backbone of LLMs
Encoder	In the transformer architecture, an encoder is responsible for processing input data and converting it into a form that can be effectively used by the decoder

Inference	The process of taking inputs (e.g., user prompt) and passing them into the model to obtain outputs. During this process, the model uses model parameters
Parameters	After the process of model training, model parameters or weights are what the model learns from data and are used during inference
Architecture	A neural network architecture refers to a specific design of a neural network. In NLP, popular architectures include RNN, LSTM
Transformer	A type of neural network architecture designed to process sequential data non-sequentially and is the backbone of LLMs
Encoder	In the transformer architecture, an encoder is responsible for processing input data and converting it into a form that can be effectively used by the decoder (in tasks like machine translation)

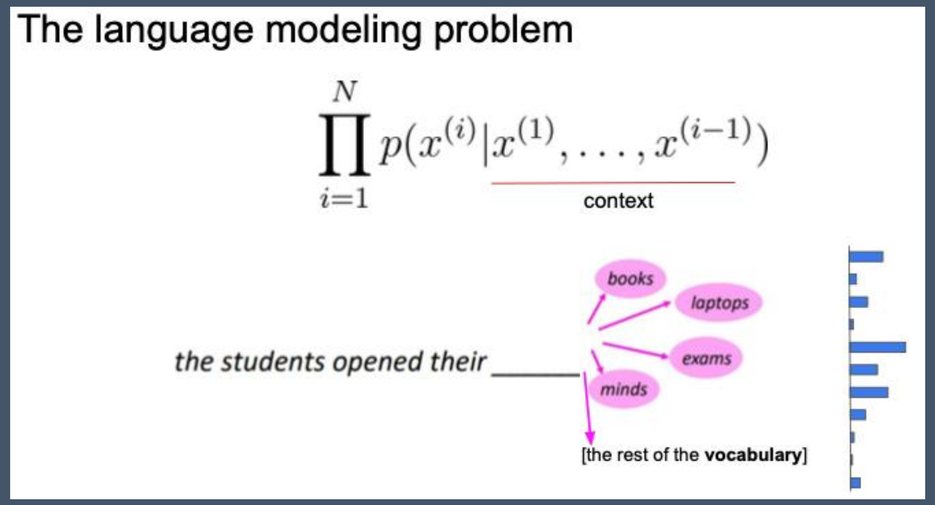
Why Are We Focusing on LLMs?



- There are four main categories of generative models- those which generate text, those which generate images, those which generate audios and finally those generating videos
- These models are significantly different in their architectures due to differing tasks and input data for training. However, all of them are based on deep learning.
- The most prominent and capable Gen AI is that of text generation (language generation and understanding). This is why we are focusing on LLMs rather than the computer vision models (image generation) or other models.
- Also, LLMs are important because they do act as a unifying block across all the four model categories. For example, to generate video from text, the model needs an LLM to understand user intent. Similarly, LLMs are useful in Speech Systems (e.g., Speech to text)

What is a Language Model (LM)?

A language model is a machine learning model that aims to predict and generate plausible language. For example, autocomplete which we use in smartphones is an example of a language model because it predicts what's the next plausible word based on what you have typed.



- An LM can also be thought of a statistical model that can predict the probability distribution of a sequence of words in a sentence.
- A language model answers the question: What is *p(text)*?
- **The text in *p(text)* is often called a token. A token can be a single character or a sequence of characters forming a word, a sentence or even a document.**
- **Therefore, an LM estimating the probability of a token or sequence of tokens occurring within a longer sequence of tokens**
- **For simplicity, we can think of a token as a single a word**

What is a Language Model (LM)- An Example

Given the sentence below, a LM will estimate probabilities of potential words based on the vocabulary

Target Sentence

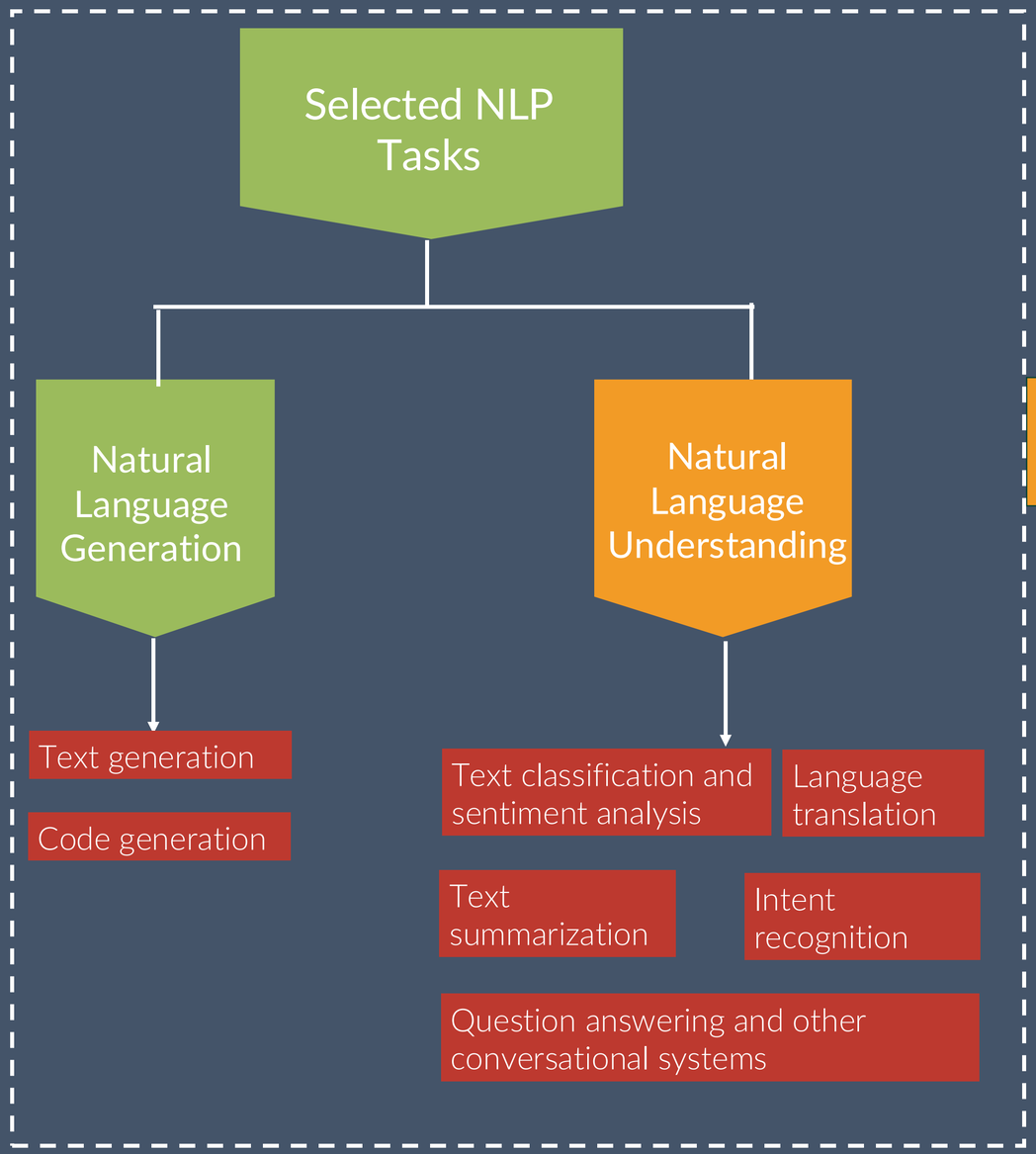
The students opened their .....

Probabilities Generated By LM

Word Counter	Words in Vocabulary	Probability
1	Books	0.6
2	Mouth	0.1
3	Laptops	0.2
4	Minds	0.004
	.....	0.000
	.....	
10,0000	Cars	0.0001

- An LM estimates probabilities based on what it learned in a corpus of data during training phase
- Thus, the LM can be thought doing next token prediction
- At each timestep, sample a token from the language model's new probability distribution over next tokens.

What is the Role of a Language Model (LM) in NLP?



LMs play a crucial role in all of these tasks

1. A judge of grammatical correctness (e.g., should prefer “The boy runs.” to “The boy run.”)

2. A judge of semantic plausibility (e.g., should prefer “The woman spoke.” to “The sandwich spoke.”

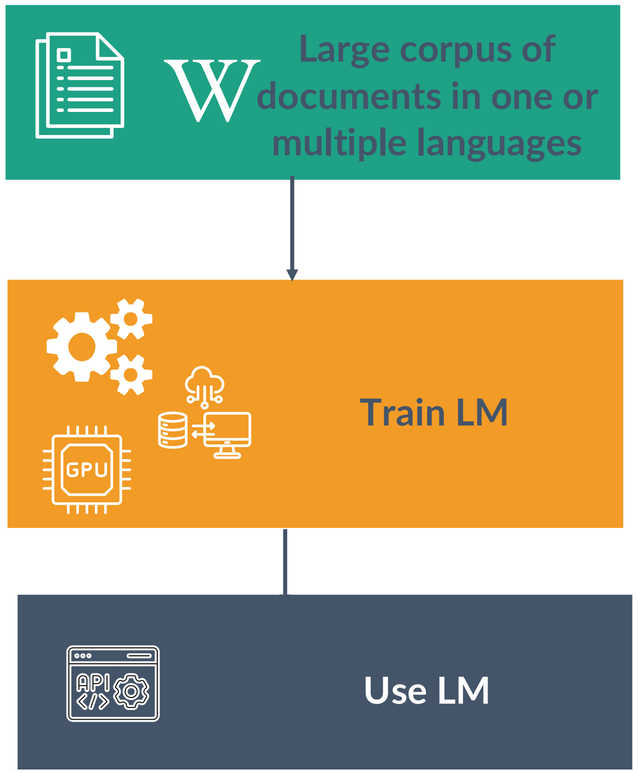
3. An enforcer of stylistic consistency (e.g., should prefer “Hello, how are you this evening? Fine, thanks, how are you?” to “Hello, how are you this evening? Has your house ever been burgled?”

3. A repository of knowledge (?)
   - e.g., “Barack Obama was the 44th President of the United States”

How Are Language Models Created?

Developing LM

The process of developing a LM is essentially an ML process of learning from massive dataset (in NLP called corpus)



Types of LM

1. N-gram Models: These are simple probabilistic models that use the conditional probability of a word given the previous $n-1$ words. For example, a bigram model uses the previous word to predict the next word.

2. Neural Network-Based Models: These models use neural networks to capture more complex patterns in the data. Examples include:
    1. Recurrent Neural Networks (RNNs): These are designed to handle sequences of data by maintaining a hidden state that captures information about previous words.
    2. Long Short-Term Memory (LSTM) Networks: A type of RNN designed to address the vanishing gradient problem, making it better at capturing long-range dependencies.
    3. Transformers: The most advanced type of language models, which use attention mechanisms to weigh the importance of different words in a sequence, allowing for more parallel processing and better handling of long-range dependencies.

Introducing Large Language Models

Core Features and Characteristics of LLMs

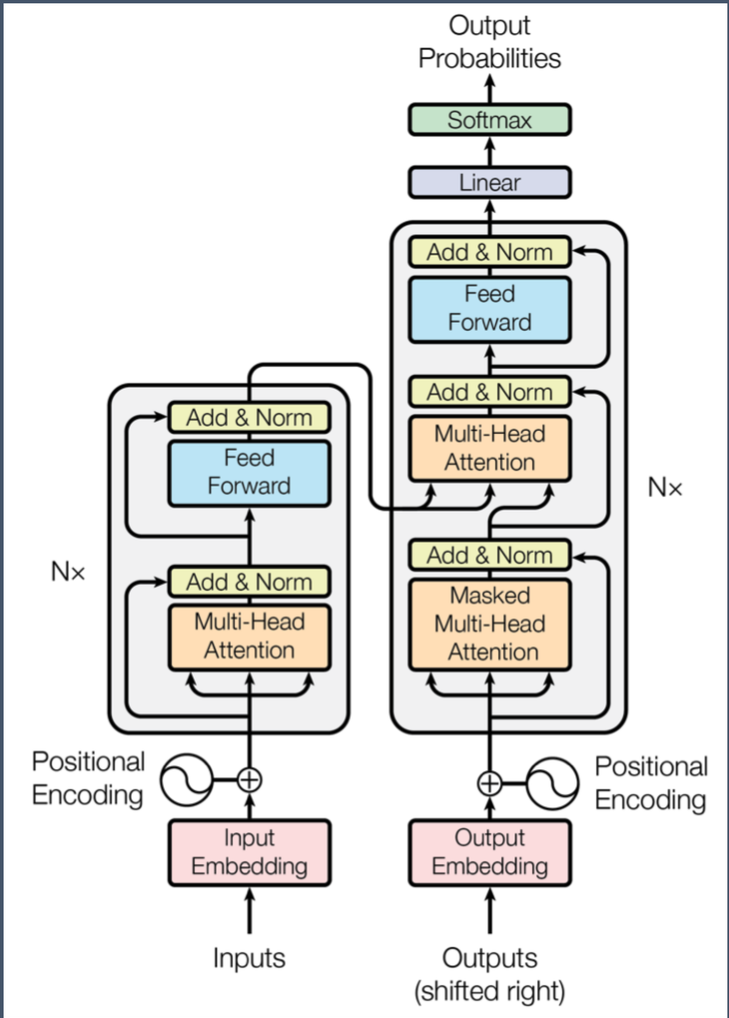
Main Features

- All LLMs are neural network-based models. More specifically, deep learning.
- LLMs capture complex patterns in text data
- The neural network architecture for modern LLMs is called Transformers
- The transformer architecture for language modeling was introduced in 2017. The core idea in transformers is what's called attention.
- Full Transformers consist of an encoder and a decoder.
- An encoder converts input text into an intermediate representation, and a decoder converts that intermediate representation into useful text
- Transformers rely heavily on a concept called self-attention. The self part of self-attention refers to the "egocentric" focus of each token in a corpus. Effectively, on behalf of each token of input, self-attention asks, "How much does every other token of input matter to me?
- Transformer-based LLMs can process data in parallel, making them more efficient and scalable compared to sequential models like RNNs.
- The definition of large is not very clear. In the past, models such as BERT with only 110m parameters have been described as large while other LLMs (e.g., PaLM2) has 340B parameters

Characteristics and Capabilities

- **Parameters:** LLMs have a vast number of parameters (ranging from millions to hundreds of billions), which allows them to learn and model complex patterns in language data.
- Training Data: They are trained on extensive datasets that include diverse text sources, enabling them to capture a wide range of linguistic nuances and knowledge.
- Pre-training: LLMs are initially pre-trained on large corpora using unsupervised learning to learn general language patterns and representations.
- Fine-tuning: They can be fine-tuned on specific tasks or domains using smaller, task-specific datasets, which helps them adapt to particular applications.
- Text Generation: LLMs can generate fluent and contextually appropriate text based on given prompts, making them useful for creative writing, dialogue systems, and content creation.
- Continuations: They can produce plausible continuations of text, maintaining coherence and relevance to the initial input.
- Versatility: LLMs can perform a wide range of NLP tasks, including text classification, translation, summarization, and question answering, often with minimal task-specific modifications.
- Transfer Learning: Knowledge learned during pre-training can be transferred to new tasks with fine-tuning, demonstrating their adaptability
- Unified Framework: Some LLMs, like T5, are designed to handle multiple tasks within a single framework by framing all tasks as text-to-text transformations.
- Task Agnostic: They can generalize across different tasks, reducing the need for separate models for each specific task.

All About the Transformer Architecture



The original Transformer Architecture

- Two main stacks: encoder and decoder

- Each layer: attention mechanisms and
- feed-forward computations

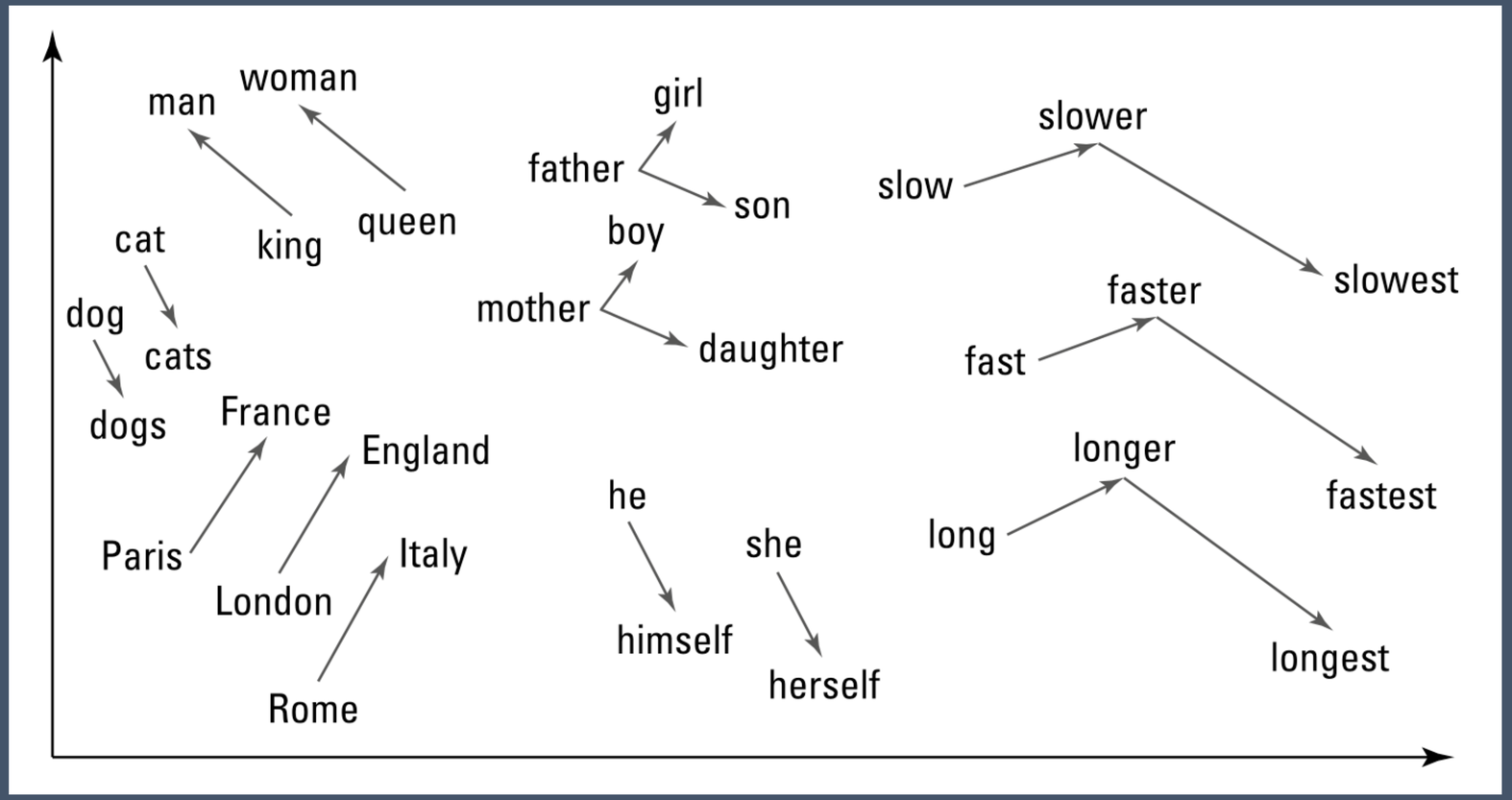
- Capture complex semantic patterns and
- Dependencies

- No recurrence nor convolutions

- It was originally designed and intended for language translation task
- However, tranformers are used o train general purpose LLMs which can perform various tasks
  - Translation
  - Summarization
  - Question-answering

Word Representation in LLMs

A word embedding is a representation of a word. Typically, the representation is a real-valued vector that encodes the meaning of the word in such a way that the words that are closer in the vector space are expected to be similar in meaning.

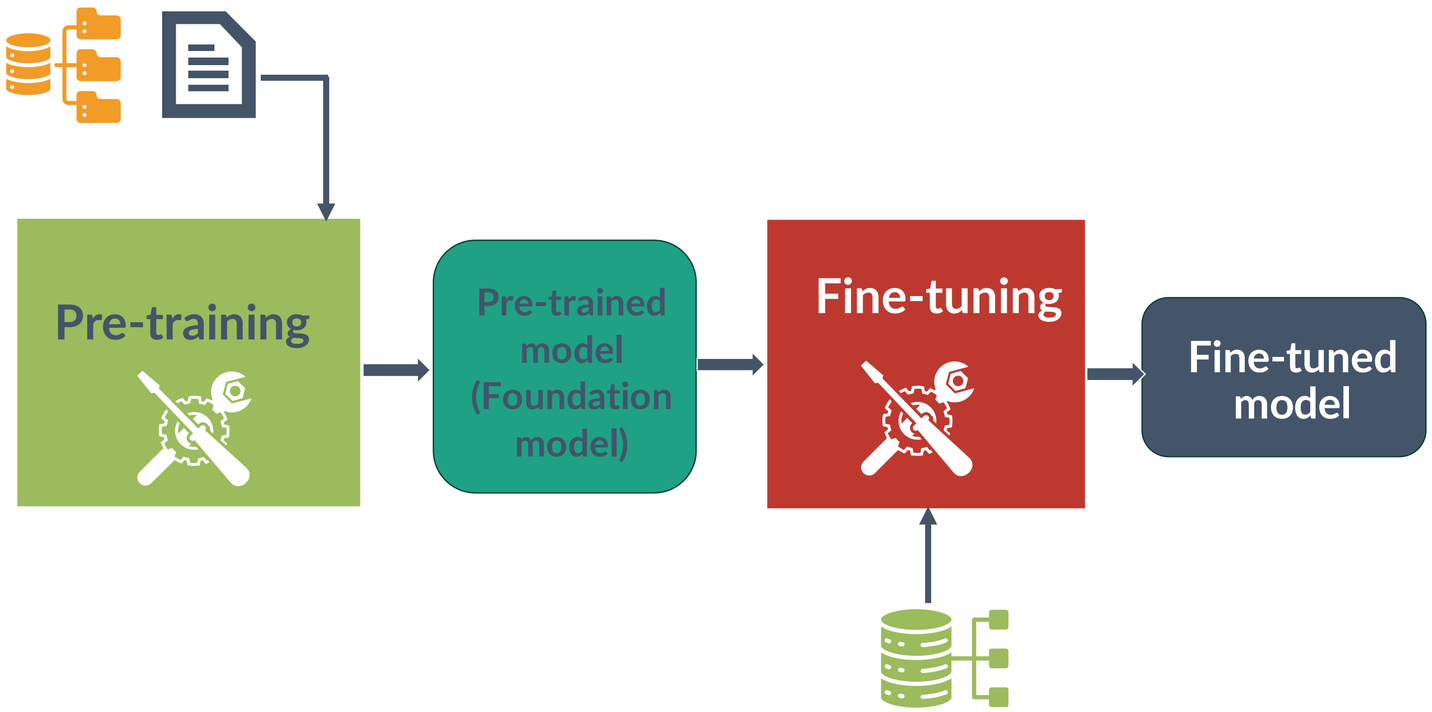


- Words are represented as dense, continuous vectors rather than discrete symbols.

- Each word is associated with a unique vector of fixed dimensionality (e.g., 100, 300).
- The position of word vectors in the embedding space reflects semantic similarities and relationships.
- Words with similar meanings are located close to each other in the vector space (e.g., "king" and "queen", "apple" and "orange").
- Embeddings reduce the high dimensionality of one-hot encoded vectors (where each dimension corresponds to a unique word in the vocabulary) to a lower-dimensional space. This reduction makes computations more efficient and captures more generalizable patterns.
- Popular methods for generating word embeddings
  - Word2Vec
  - GloVe (Global Vectors for Word Representation)
  - FastText
- The encoder part of transformer models can also learn word embeddings
  - Use contextual embeddings, where the representation of a word varies depending on its context within a sentence.
  - Capture more nuanced meanings by considering the entire sentence or passage, enhancing the understanding of polysemous words (words with multiple meanings).

A word can be identified by the group of words that are used with it frequently. This is the basis for creating vector embeddings

LLMs Development



Main Steps

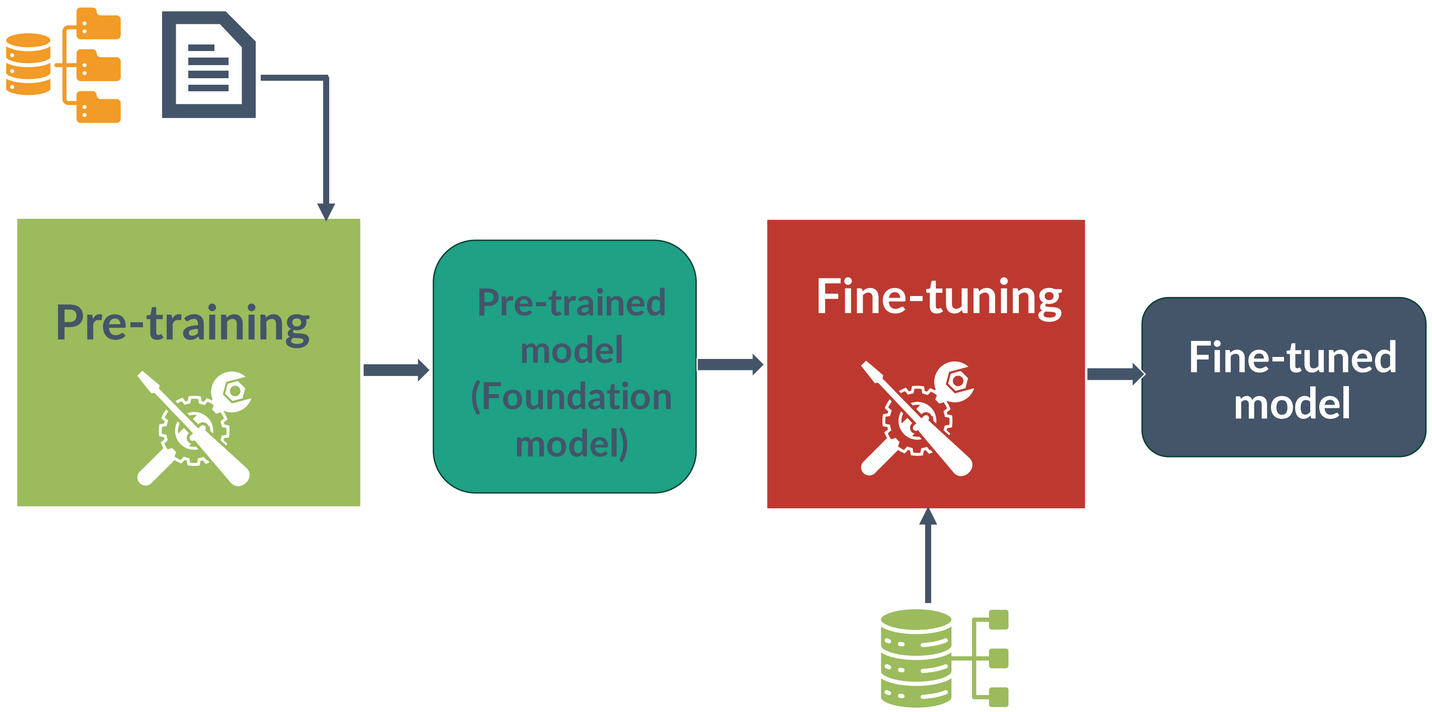
1. Data ingestion and preparation
    1. Gather Data: Collect a large and diverse dataset of text from various sources, ensuring it is representative of the language and tasks the model will handle.
    2. Clean Data: Remove noise, duplicates, and irrelevant information. Normalize text by converting to lowercase, removing special characters, etc.
    3. Tokenization: Break down text into tokens (words, subwords, or characters) that the model can process.

2. Model architecture design
    1. Select Architecture: Decide on the architecture of the model, typically based on the transformer model (e.g., BERT, GPT, T5).
    2. Design Model: Define the number of layers, attention heads, hidden units, and other hyperparameters.

3. Pre-training
    1. The output of pre-training is often a general purpose model
4. Fine-tuning
    1. Task-Specific Datasets: Collect and prepare smaller datasets for specific downstream tasks (e.g., sentiment analysis, named entity recognition).
    2. Fine-tuning Process: Fine-tune the pre-trained model on these datasets, allowing it to adapt to specific tasks while leveraging the general language understanding from pre-training.

5. Model evaluation
6. Deployment, monitoring and maintenance

Understanding LLM Model Categories



Categorizing LLMs

1. **Foundation models(FM)**. These are pre-trained models that serve as a base for various downstream tasks. They are typically trained on broad and diverse datasets, capturing a wide array of language patterns and knowledge. Examples include BERT, GPT-3, and T5.
    1. **Task-specific** LLMs such as Meta's Code Llama specialize in unique, highly targeted tasks like generating software code.
    2. **Domain-specific** LLMs apply gen AI technology to specific subjects and industries. For example, NVIDIA's BioBERT, which has been trained on biomedical text, helps research- ers understand scientific literature and extract information from medical documents.

2. **General-purpose characteristics of FM**. Many foundation models are designed to be general-purpose in nature, meaning they can be fine-tuned or adapted to perform well on a wide variety of NLP tasks, such as text classification, question answering, text generation, machine translation and more.
3. A foundation model can often be multimodal, meaning it handles both text and other media such as images.
4. Fine-tuned models are specialized models adapted from FM. This involves applying additional training using transfer learning on task-specific or domain-specific datasets helps adapt the model to particular applications.
    1. Task specific fine-tuning. This takes an FM and fine-tunes to perform a specific task. For example, you can fine-tune an FM to become better at sentiment analysis
    2. Domain specific. Similarly, you can fine-tune an FM to your domain of choice.

Summary of LLM Capabilities

1. **LLM core capability.** LLMs are sophisticated predictive models that anticipate the next word in a sequence based on the context provided to them as part of a process referred to as a completion. Alternatively, LLMs perform what's called next token prediction.

2. **Making LLMs better with RLHF**. General purpose pre-trained LLMs do not have the capability to chat or provide the kind of responses that chat-GPT does. Instead, after pretraining, OpenAI does instruction-tuning using a techniques such as called Reinforcement Learning with Human Feedback (RLHF) to create a model which is able to chat and respond to questions in a form amenable to humans.

3. **The role of prompting and context.** Carefully constructed prompts help these models deliver tailored content, yielding better completions. LLM performance is influenced not only by the training data but also by the context provided by these user inputs — the prompts.

TO DO- COMPLETE THIS SLIDE

3. Harnessing Pre-trained LLMs

1. The LLMs Landscape
2. Vector Embeddings and other Concepts
3. Accessing LLMs with the HuggingFace Platform
4. Evaluating LLMs

Characterizing LLMs

We can review, explore and compare LLMs based on the following Key characteristics

1. Proprietary vs. open source. This is important because it determines how you can use the model, cost of using the model, whether you modify and own it etc.
    1. Open source LLMs are models whose code and, often, pretrained weights are made publicly available, allowing anyone to use, modify, and distribute them. Thus, as a user, you can host these open models on your own infrastructure
    2. Commercial LLMs are the ones which are owned by corporations and one needs to pay to access and use them. The model weights are not freely accessible, these models can only be used with the providers API

2. General purpose vs. specialized in the domain space. Whether the model is specialized on a specific domain (e.g., legal, medical etc.) or its general purpose.
3. General purpose vs. specialized in the task space. For example, whether the model is specialized at handling code rather than language. Or the model is fine-tuned to do question answering or sentiment analysis.
4. Size (number of parameters). This is important because it determines several things such as whether its easy to fine-tune or use the model. Note that most commercial companies do not fully share number of parameters for their proprietary models.
5. Context window. This is given as number of tokens but it can be roughly translated to words or number of pages This determines how much text you can put in the prompt. Thus, the larger the context window the better.
6. Multimodal. This is determining whether you can process other data types other than text. For example, some model can accept images, audios or videos as inputs

The Latest LLMs on the Market

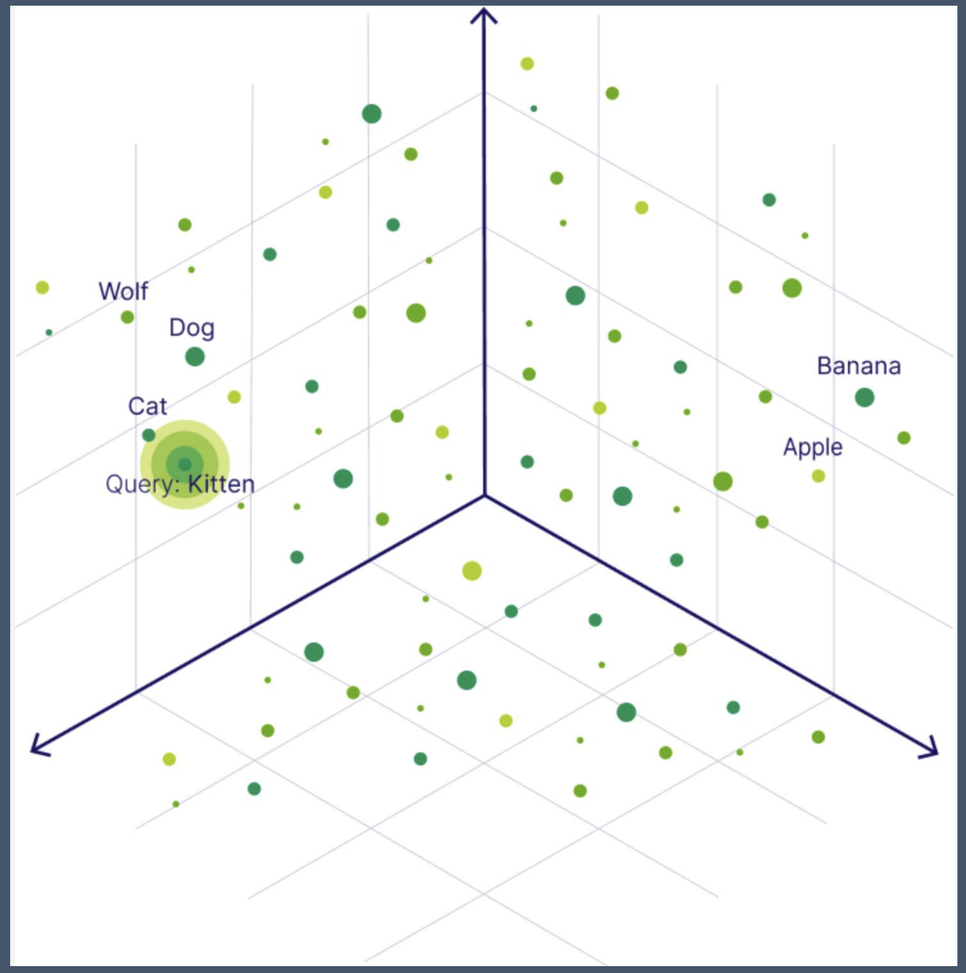
Model name (Company)	Open or proprietary	Domain Specific	Task	Params	Context window
GPT family (Open AI)	Proprietary	General purpose	General	175B-Trillions	8,192, 32, 768 & 128, 000
Llama family (Meta AI)	Open.	General purpose .	General	8B – 70B	8,000
Claude v1 (Anthropic)	Proprietary.	General purpose.	General	~175B - ~500B	100, 000
Falcon(Tech. Innovation. Inst	Open	General purpose	General.	7B, 40B, 180B	2, 048
Cohere Command (Cohere)	Proprietary.	General purpose.	General.	104B	128, 000
Code Llama(Meta)	Open	Programming code generation	Generating code	7B, 13B, 34B	100, 000.
Gemini (Google)	Proprietary	General purpose	General	175B- Trillions.	Up to 1 million
Mistral (Mistral AI)	Open .	General purpose.	General	7B	32, 000

Selected Domain Specific LLMs

Domain-specific LLMs focus on a specific subject area or industry.

- BioBERT is trained on biomedical text, making it an excellent resource for understanding scientific literature and extracting information from medical documents.
- ClinicalBERT. ClinicalBERT is pre-trained on clinical notes and medical records, making it effective for tasks like clinical named entity recognition, relation extraction, and clinical outcome prediction.
- CodeBERT is a cybersecurity solution that has been trained to assist with IT security concerns such as vulnerability detection, code review, and software security analysis.
- SciBERT. A BERT-based model pre-trained on a large corpus of scientific papers from the domains of computer science and biomedicine. It aims to improve performance on tasks like scientific text classification, entity recognition, and question answering.
- FinBERT. FinBERT is optimized for understanding financial communications, such as earnings calls, financial reports, and market analysis. It can be used for sentiment analysis, entity recognition, and classification in financial documents.

Vector Databases



1. **Vector Database.** A vector database indexes, stores, and provides access to structured or unstructured data alongside its vector embeddings. It allows users to find and retrieve similar objects quickly at scale in production.

2. **Vector Embeddings.** Vector embeddings enables us to numerically represent unstructured data without losing its semantic meaning in so-called vector embeddings. A vector embedding is just a long list of numbers, each describing a feature of the data object. A popular example of vector embeddings are word embeddings which represent words in numeric format. These are called word vectors.

    - Vector embeddings numerically capture the semantic meaning of the objects in relation to other objects. Thus, similar objects are grouped together in the vector space, which means the closer two objects, the more similar they are
    - Vector embeddings are generated through a ML process and the model used to generate them (often transformers and other deep learning models) are called embedding models.
    - Its important to know and understand the model generating the vector embeddings you are working with.

3. **Vector Search. Vector embeddings allow us to find and retrieve similar objects from the vector database by searching for objects that are close to each other in the vector space, which is called vector search, similarity search, or semantic search.**
    1. **Semantic search, which is based on contextual meaning, offers a more human-like search experience by retrieving results that align with the user's intent. This capability is crucial for applications sensitive to typos or synonyms, as vector search can accurately match relevant results despite these variations.**

4. **Similarity Metrics in Vector Search. Numerical representation of data objects enables mathematical operations, like calculating the distance between two vectors to determine their similarity. Different distance metrics can be used: Euclidean distance, cosine similarity, dot product and more.**

5. **Vector indexing. Vector indexing is the process of organizing vector embeddings in a way that data can be retrieved efficiently.**

Vector Databases and LLMs

Vector Databases are an integra part of building LLM applications

1. Semantic Search and retrieval
    - Contextual Search: Vector databases store the vector embeddings generated by LLMs, allowing for semantic search. This means that search queries can retrieve results based on meaning rather than exact keyword matches.
    - Improved Accuracy: By using vector similarity measures (e.g., cosine similarity), searches can return more relevant results, accounting for synonyms, typos, and contextual relevance.

1. Multimodal data integration.

    - Cross-Modal Retrieval: Vector databases can store embeddings from different data types (text, images, audio) generated by multimodal LLMs. This allows for applications to search and retrieve relevant data across different modalities based on semantic similarity.
    - Unified Storage: They provide a unified storage solution for embeddings from various data sources, simplifying the data management process.

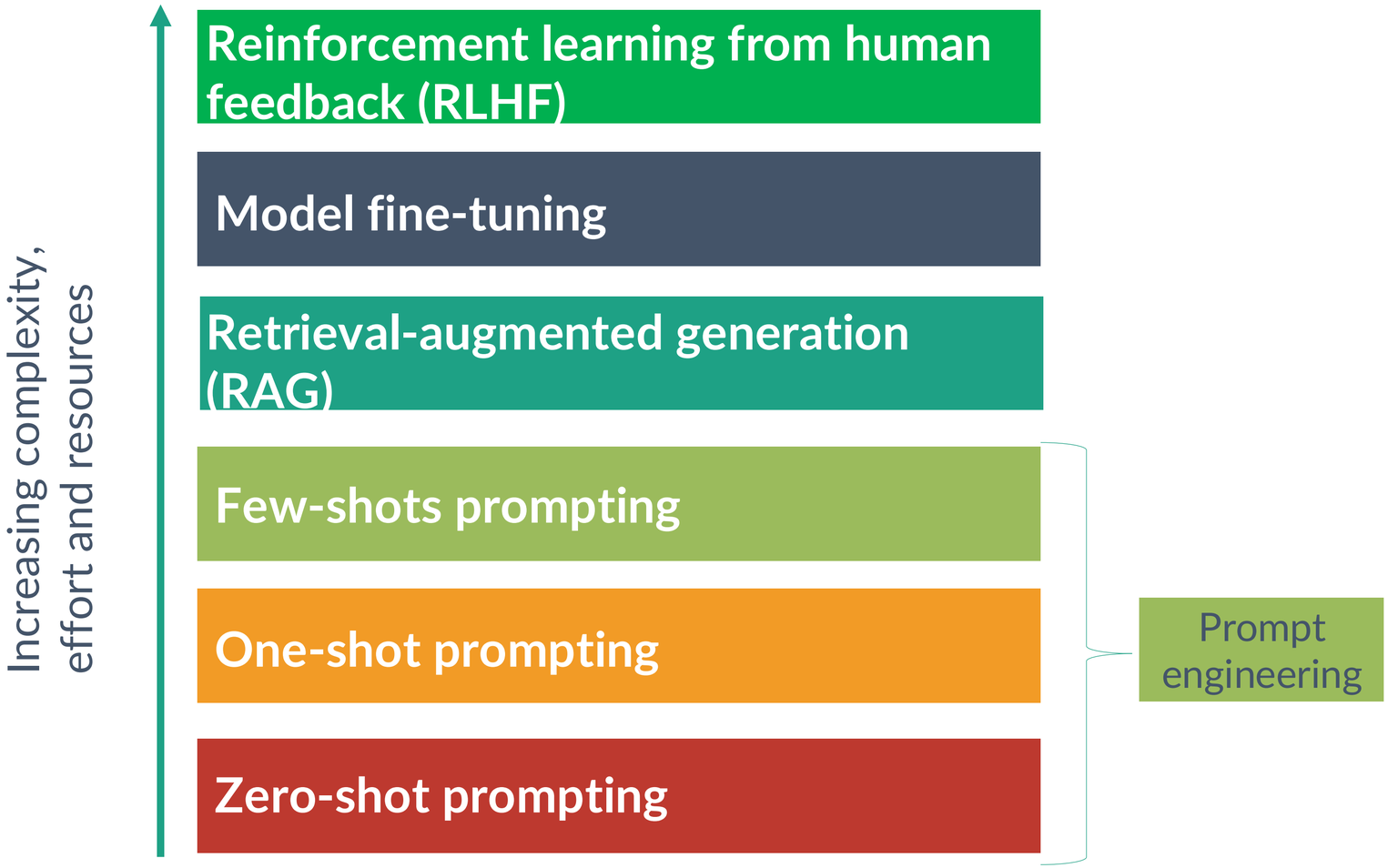
2. Content Generation and summarization.
    1. Efficient Retrieval for Summarization: Vector databases can quickly retrieve relevant information based on semantic similarity, aiding LLMs in generating accurate and comprehensive summaries.
    2. Content Matching: They help in matching content pieces for tasks like news aggregation, content curation, and plagiarism detection.

3. Contextual Understanding in Chatbots and Assistants
    1. Memory and Context: Chatbots and virtual assistants can store conversational context as embeddings. This helps maintain the flow of conversation over long interactions and provides more coherent and contextually appropriate responses.
    2. Question Answering: For applications like customer support, vector databases can store and retrieve answers to frequently asked questions based on the semantic similarity of user queries.

Adapting and Customizing LLMs

Main Approaches to Tailor LLM to Meet Specific Needs

Summary of Main Approaches



Why Adapt LLMs?

1. Tasks. To make them preform different task than what they were trained for. For example, some general purpose LLMs aren't fine-tuned for machine translation, they can be fine-tuned to do MT better.

2. Domain. Most LLMs are general purpose and do no perform well on data in a specific domain or organization specific information. They need to be adapted with specific domain data (e.g., medical)

3. Currency and retrieval. LLMs do not have up to date information. They can be adapted using RAG to retrieve and use updated documents and information.

4. Language. LLMs aren't trained on all languages. To have good performance on a language that they weren't trained on, an LLM can be fine-tuned.

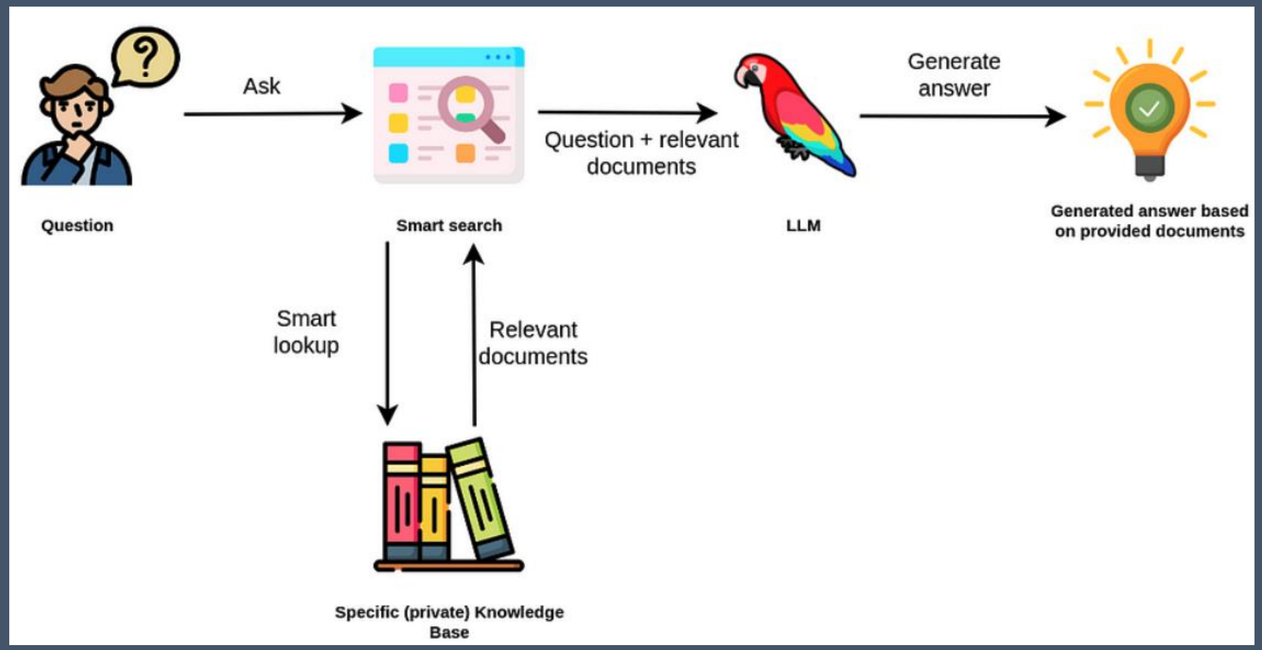
Adapting and Customizing LLMs

Prompt Engineering

- LLMs predict the next word in a sequence based on context, a process called completion. Carefully crafted prompts help deliver tailored content and improve completions. The performance of LLMs is influenced by both their training data and the user-provided prompts.
- Prompt engineering involves crafting inputs to shape a language model's output for desired results. By specifying important sections and desired word count, users can obtain targeted and relevant responses. Techniques used in prompt engineering include zero-shot, one-shot, and few-shot prompting
- Zero-shot prompting is the default; you simply issue a question and rely on the LLM’s pretrained information to answer it.
- One-shot prompting, you include an example of the desired output to help the model understand the desired output. For instance, assume you’re writing a travel brochure and you want the AI model to describe a vibrant public market in an exotic city. Even with limited exposure to this particular scenario, a model can generate a creative description that matches the tone of voice, vibrant use of adjectives, and structure that has already proven successful in your marketing content.
- Few-shot prompting takes it a step further by providing multiple examples to more clearly teach the LLM the desired output structure and language.

Adapting and Customizing LLMs

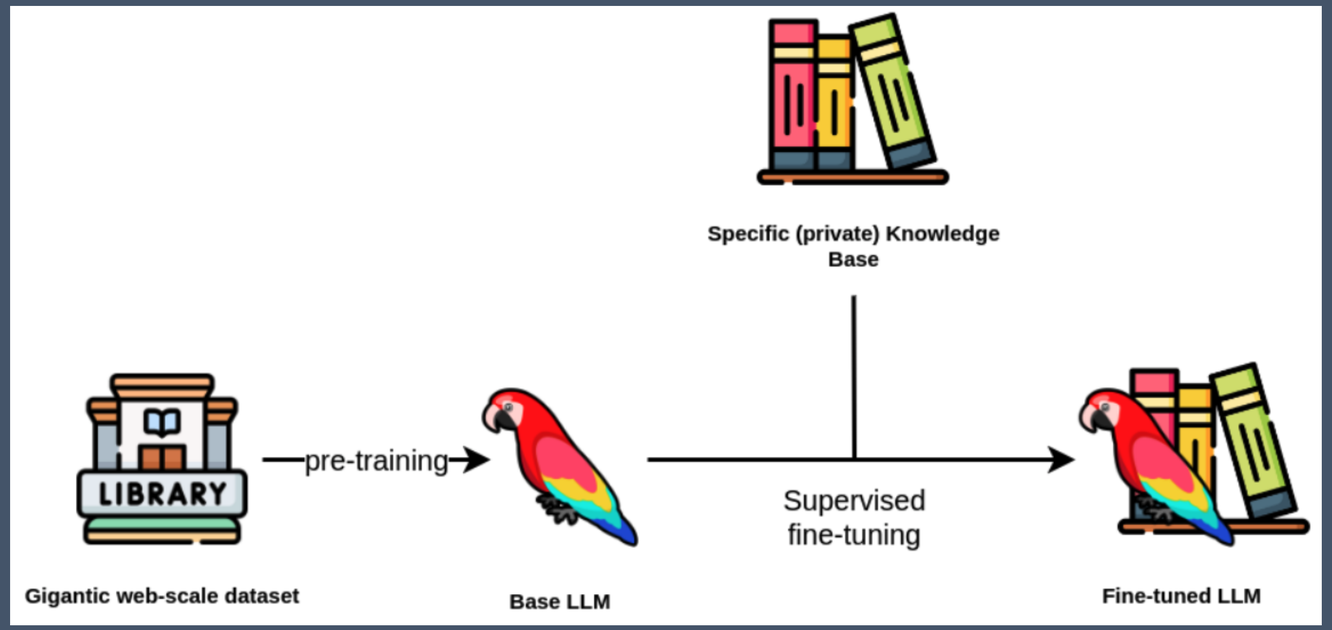
Retrieval Augmented Generation (RAG)



- RAG enhances content generation by combining a pretrained language model with a large-scale vector search system. It tackles common issues in generative AI, such as limited knowledge of niche subjects, unrecognized post-training facts and events, and lack of access to proprietary data.
- RAG accesses up-to-date information by retrieving relevant data stored as vectors (numerical representation of the data for fast retrieval), such as current news, to bring the model up to date with recent events or domain-specific content from a particular industry or market.
- You can enhance LLMs by granting them access to your data and documents, such as private wikis and knowledge bases. By retrieving this extra information, the models can generate more accurate and contextually relevant responses.
- For instance, if you published a new port on ongoing pandemic c such as COVID, you can use RAG to provide an LLM access to this document and thus enable users get answers based on uptodate and current information

Adapting and Customizing LLMs

Supervised Fine-tuning



- Fine-tuning enables you to adapt an LLM to particular task or to perform better on a custom dataset by updating the parameters of a pretrained model. These techniques empower users to shape LLMs according to their preferences and achieve better results in various applications

- Main Steps in Fine-tuning
  - Select the pretrained LLM that is most germane to the use case.
  - Identify or create data sets related to the use case to refine the LLM. This can includes examples of prompts as well as the data the model needs to answer or complete the prompt.
  - Train the model. You can adjust the learning rate, batch size, and other factors to improve outcomes. There are several methods available to make this process fast and efficient (e.g., Parameter Efficient Fine-tuning (PEFT), LoRA, QLoRA
  - Evaluate the fine-tuned LLM to verify results meet requirements.

- Fine-tune is resource intensive, requires more skilled machine learning engineers or data scientists, needs properly curated data. As such, it should be attempted after tying out the low resource and low effort adaptation approaches such as RAG and prompt engineering.

Evaluating LLMs

How to Ensure Outputs from LLMs or LLM based Apps Are Accurate

Why Evaluate LLMs?

1. **Accuracy.** Just like any other ML model, we have to ensure that the LLM's performance is accurate.
2. **Alignment.** Ensure that the outputs from the LLM match with what you expect. For example, in a QA system, you don't want the LLM to provide wrong, contradictory or even dangerous information.
3. **Reduce bias**. LLMs and other ML models can produce results which are discriminatory against other groups of people. Evaluating the model is the first to mitigate against this.
4. **Reduce hallucinations.** Ensure the LLM isn't producing false information and presenting it as facts.
5. **Reduce toxicity.** Toxicity refers to the generation of harmful, offensive, or inappropriate content. Evaluation helps to mitigate this behaviour.

How to Evaluate LLMs?

1. **More complicated to evaluate.** LLMs and LLM based apps are more complicated to evaluate compared to classical ML models because of the generative nature of outputs from LLMs
2. **Task specific.** LLM evaluation metrics, strategy and data will depend on the task at hand.
3. **Evaluation metric.** Select one or more evaluation metric to use to evaluate your LLM app. In most cases, you need more than one metric.
4. **Create evaluation data.** Evaluation always needs data which contains what are considered as accurate, correct or preferred responses/outputs. Although data often is human curated, nowadays, you can LLMs to generate evaluation data.
5. **Decide on evaluation strategy.** Setup experiments to perform evaluation.

More About Hallucinations

Truthfulness vs. Hallucinations

Hallucinations refers to LLMs generating text that contains false or nonsensical information but being presented confidently as if it were truthful

ADD HALLUCINATION EXAMPLES

Strategies to reduce LLM hallucinations:

1. Exposure to diverse and representative training data
2. Bias audits on model outputs + bias removal techniques
3. Fine-tune to specific use cases in sensitive applications
4. Prompt engineering: carefully crafting and refining prompts

LLM Tasks and Metrics

A summary of Key Metrics for Main LLM Tasks

Text/Document Classification

- Accuracy
- F1 score

Text Generation

- Perplexity
- BLEU score

Translation

- BLEU score
- METEOR

Question answering

- Exact Match (EM)
- BLEU/METEOR

Summarization

- ROUGUE score
- METEOR

- Metric limitations. Each metric provides its own insights but also has limitations.
- Multiple metrics. A combination of metrics including domain specific KPIs may work better than relying on a single metric.
- LLM capabilities. Be realistic: a pre-trained book genre classifier may not accurately identify sentiments.

Perplexity in Text Generation

```
model_name = "gpt2"
# ... Load model and tokenizer ...

prompt = "Latest research findings in Antarctica show"
prompt_ids = tokenizer.encode(test_sentence, return_tensors="pt")
output = model.generate(prompt_ids, max_length=17)
generated_text = tokenizer.decode(
  output[0], skip_special_tokens=True)
print(generated_text)
```

```
Latest research findings in Antarctica show that the ice sheet
is melting faster than previously thought.
```

```
perplexity = evaluate.load("perplexity", module_type="metric")
results = perplexity.compute(model_id='gpt2',
                             predictions=generated_text)
print(results['mean_perplexity'])
```

```
2867.6985334123883
```

- Perplexity is defined as the model's ability to predict the next word accurately and confidently
- Calculated upon model's output logits
- range [0, inf) : lower is better
- Example: generate a few tokens upon a prompt and calculate perplexity of sequence
- Likelihood that the model generated the sequence
- Average sequences' perplexity: mean

ROUGE Score in Text Summarization

ROUGE: Recall-Oriented Understudy for Gist Evaluation

```
rouge = evaluate.load("rouge")
predictions = ["""as we learn more about the frequency and
        size distribution of exoplanets ,we are discovering
        that terrestrial planets are exceedingly common."""]
references = ["""The more we learn about the frequency and
        size distribution of exoplantes, the more confident we
        are that they are exceedingly common."""]

results = rouge.compute(predictions=predictions,
                        references=references)
print(results)
```

{'rouge1': 0.7441860465116279, 'rouge2': 0.4878048780487805,
'rougeL': 0.6976744186046512,'rougeLsum': 0.6976744186046512}

- ROUGE is the overlap between generated a summary and reference summaries.

- The summaries can de defined as n-grams, word overlap, block/paragraph ….
- Predictions/generate summaries: LLM outputs
- References : human-provided summaries

- ROUGE scores:
  - rouge1 : unigram overlap
  - rouge2 : bigram overlap
  - rougeL : long overlapping subsequences

BLEU Score in Translation

BLEU: BiLingual Evaluation Understudy

```
import evaluate
from transformers import pipeline
bleu = evaluate.load("bleu")

translator = pipeline(
  "translation", model="Helsinki-NLP/opus-mt-es-en")
input = "Qué hermoso día"
references = [["What a gorgeous day",
               "What a beautiful day"]]
```

```
translated_outputs = translator(input)
translated_sentence = translated_outputs[0]['translation_text']
print("Translation: ", translated_sentence)
```

Translation: What a beautiful day.

```
results = bleu.compute(
  predictions=[translated_sentence], references=references)
print(results)
```

```
{'bleu': 0.7598356856515925,
'precisions': [1.0, 1.0, 0.6666666666666666, 0.5],
'brevity_penalty': 1.0, 'length_ratio': 1.0,
'translation_length': 5, 'reference_length': 5}
```

- BLEU: measures translation quality as the correspondence between an LLM output and one (or more) human references.
- Score range: 0-1; higher is better
- predictions: LLM's output translation
- references : human translations
- Precisions in results below : mean n-gram precisions

Exact Match (EM) in Question Answering

```
from evaluate import load
em_metric = load("exact_match")

exact_match = evaluate.load("exact_match")
predictions = ["The cat sat on the mat.",
               "Theaters are great.",
               "It's like comparing oranges and apples."]
references = ["The cat sat on the mat?",
              "Theaters are great.",
              "It's like comparing apples and oranges."]

results = exact_match.compute(
  references=references, predictions=predictions)
print(results)
```

```
{'exact_match': 0.3333333333333333}
```

- The Exact Match (EM) metric in Question Answering (QA) evaluates the accuracy of a model by checking if the predicted answer exactly matches the correct answer.
- This means the predicted answer must be identical to the reference answer, considering aspects like spelling, word order, and punctuation.
- The EM metric is a strict evaluation measure and provides a binary score: 1 if the answers match exactly, and 0 if they do not.
- This metric is useful for tasks where precision is critical, but it does not account for partially correct answers or synonyms
- **Normally used in conjunction with F1 score**

METEOR Score in Translation

METEOR: Metric for Evaluation of Translation with Explicit ORdering

```
bleu = evaluate.load("bleu")
meteor = evaluate.load("meteor")
pred = ["He thought it right and necessary to become a
        knight-errant, roaming the world in armor, seeking
        adventures and practicing the deeds he had read about
        in chivalric tales."]
ref = ["He believed it was proper and essential to transform
       into a knight-errant, traveling the world in armor,
       pursuing adventures, and enacting the heroic deeds he
       had encountered in tales of chivalry."]

results_b = bleu.compute(predictions=pred, references=ref)
results_m = meteor.compute(predictions=pred, references=ref)
print("Bleu: ", results_bleu['bleu'])
print("Meteor: ", results_meteor['meteor'])
```

Bleu: 0.19088841781992524

Meteor: 0.535070224048153 6

- METEOR: overcome ROUGE and BLEU limitations.
- It comprehensively assesses: N-gram overlap, precision, recall, synonyms, stemming, word order
- More computationally expensive

- Measures overall quality of LLM-generated text

- Score range: 0-1: higher is better

Metrics for Measuring LLM Bias: Toxicity

```
from evaluate import load
toxicity_metric = load("toxicity")

texts_1 = ["Everyone in the team adores him",
           "He is a true genius, pure talent"]
texts_2 = ["Nobody in the team likes him",
           "He is a useless 'good-for-nothing'"]

toxicity_results = toxicity_metric.compute(predictions=texts_1,
                                aggregation="maximum")
print("Toxicity Sentences 1:", toxicity_results)
toxicity_results = toxicity_metric.compute(predictions=texts_2,
                                aggregation="maximum")
print("Toxicity Sentences 2:", toxicity_results)
```

```
Toxicity Sentences 1: {'max_toxicity': 0.00013866179506294435}

Toxicity Sentences 2: {'max_toxicity': 0.012273495085537434}
```

The toxicity metric quantifies text toxicity using pre-trained hate speech classifier

- It takes a list of texts as input
  - Default: a toxicity score for each input text in the list
  - aggregation="maximum" returns maximum toxicity score across inputs
  - aggregation="ratio" returns the % predictions with toxicity above 0.5

Metrics for Measuring LLM Bias: Regard

```
from evaluate import load
regard = load("regard")

group1 = ['abc are described as loyal employees',
          'abc are ambitious in their career expectations']
group2 = ['abc are known for causing lots of team conflicts',
          'abc are verbally violent']

polarity_results_1 = regard.compute(data=group1)
for result in polarity_results_1['regard']:
  print(result)
```

```
[{'label': 'positive’, 'score': 0.9098386764526367},
{'label': 'neutral’, 'score': 0.0593969523906707 76},
{'label': 'other','score': 0.0264681018888950 35},
{'label': 'negative’, 'score': 0.0042962529696524 14}]
[{'label': 'positive','score': 0.7809812426567078},
{'label': 'neutral','score': 0.180859833955764 77},
{'label': 'other','score': 0.0304929520934820 18},
{'label': 'negative','score': 0.0076660132035613 06}]
```

The **regard** metric is a measure used to evaluate the sentiment or attitude expressed in text towards a particular subject, often a demographic group or individual.

The regard metric typically involves:
1.Classification: Text is classified into categories based on the sentiment expressed (e.g., positive, neutral, negative).
2.Scoring: Assigning scores to different sentiments to quantify the degree of regard.
3.Analysis: Aggregating and analyzing scores to understand trends, biases, and attitudes towards the subject.

```
polarity_results_2 = regard.compute(data=group2)
for result in polarity_results_2['regard']:
  print(result)
```

```
[{'label': 'negative','score': 0.9658734202384949},
{'label': 'other’, 'score': 0.021555885672569275},
{'label': 'neutral','score': 0.012026479467749596},
{'label': 'positive','score': 0.0005441228277049959}]
[{'label': 'negative','score': 0.9774736166000366},
{'label': 'other','score': 0.012994581833481789},
{'label': 'neutral','score': 0.008945506066083908},
{'label': 'positive','score': 0.0005862844991497695}]
```

LLM Challenges Beyond Errors

Challenges with LLMs in Beyond Typical ML Model Errors

1

Low Resource Languages

- Multi-language support: language diversity, data availability, transferability leads to wildly varying capabilities in LLMs
- Low resource languages have limited data on the internet and therefore little or limited LLM support

2

Open vs proprietary LLMs

- Accessibility and transparency vs responsible use

3

Prohibitive Infrastructure Costs

- Extremely expensive to train LLMs as they require GPUs and training takes long time
- Depending on use case, arriving at a model can be be difficult

4

Biases, safety (e.g., jailbreak) and ethical issues

- Biased training data, unfair language understanding and generation

4. LLM Project Development

1. Overview of Main Steps in LLM Project
2. LLM Model Selection and Adaptation Considerations
3. App Deployment Considerations
4. Tools and Ecosystem in LLM Apps Development

Overview of LLM Projects

An LLM based project can be any project that seeks to leverage the power of a Gen AI platform, LLM models, speech or computer vision models. Below are selected such projects


ChatBots/Virtual Assistants

- Ask-a-Question (QAA) Chatbot to provide answers to commonly asked questions about a single or multiple topics .
- Virtual Assistant to help data users find the right dataset
- Chatbot to help health workers in rural areas work better

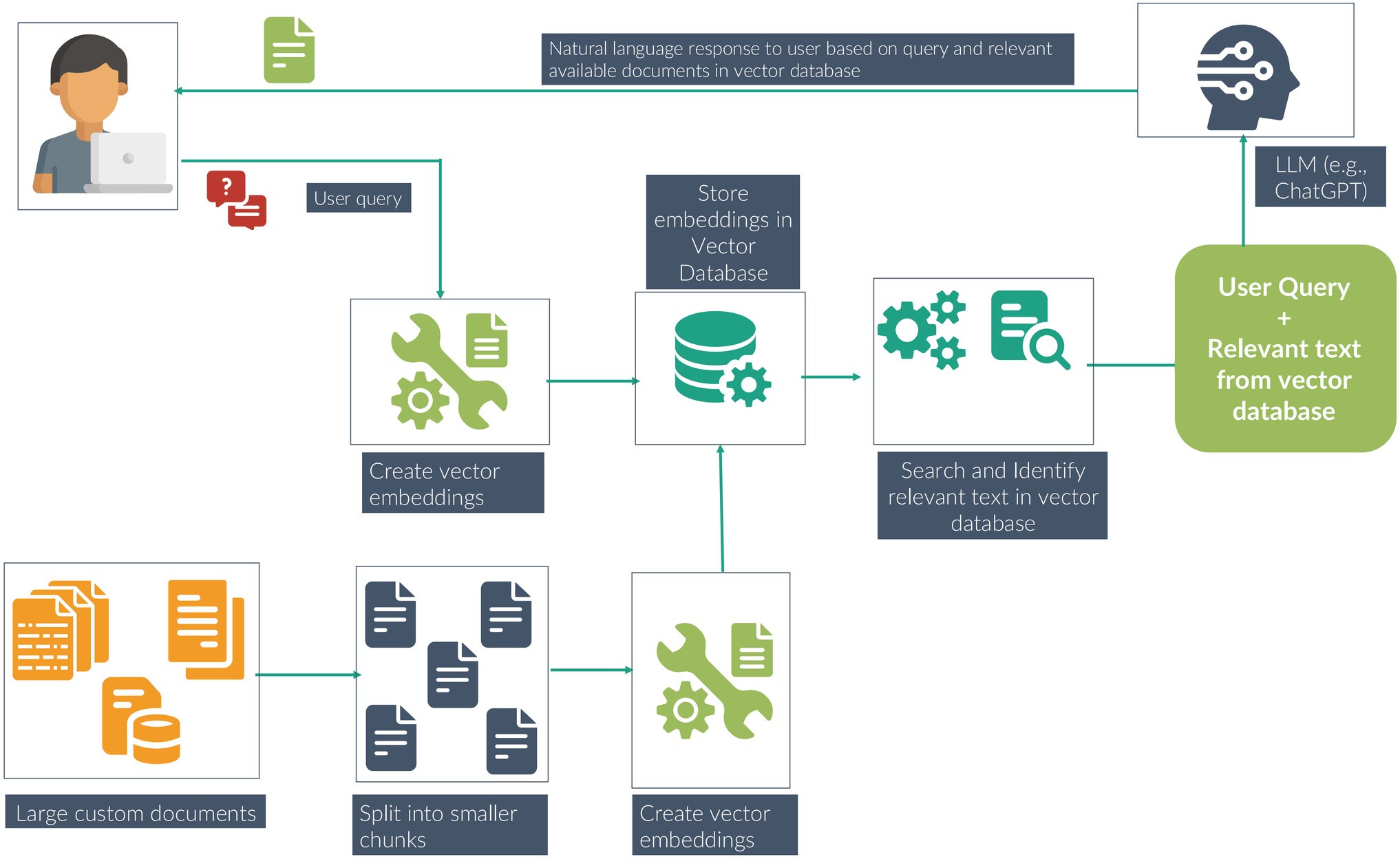

Data Collection

- Use LLM agents in web based data gaethering. For example, LLM agents can help with automation of webscraping and accessing data from APIs
- Use vision models in survey based data collection . For instance, a vision model can analyse image of house to determine nature of building materials

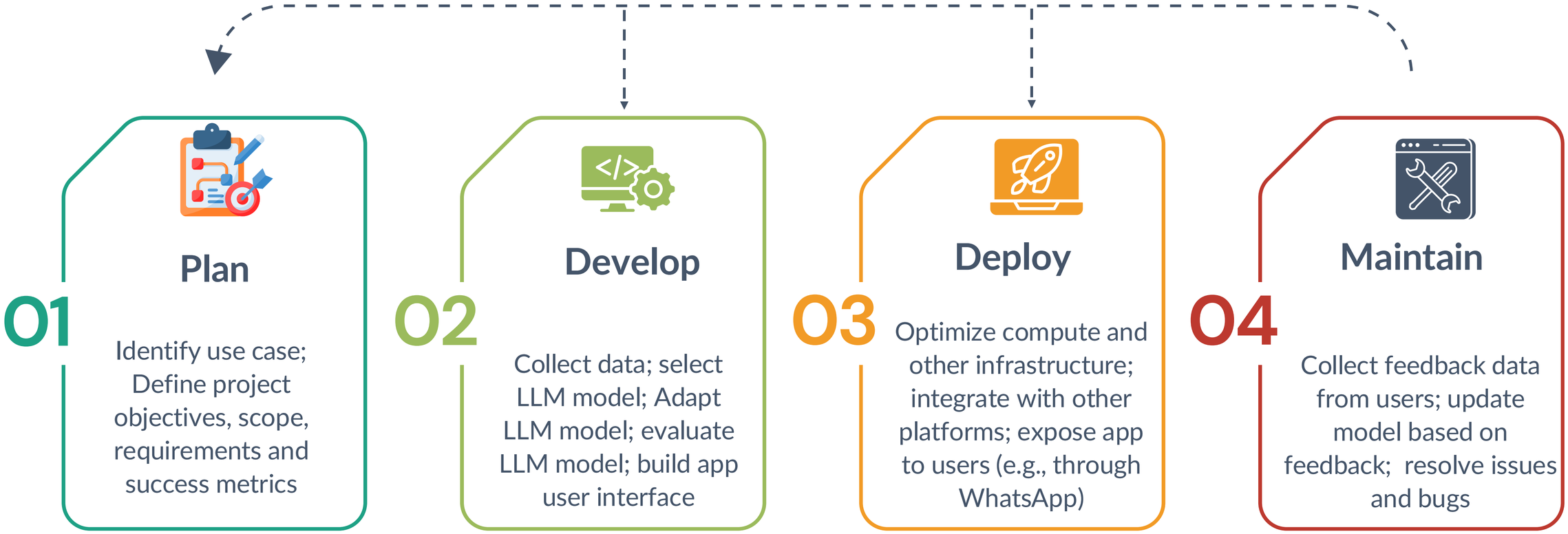

Data analysis, augmentation, curation etc

- Sentiment analysis, document classifcation project. Use LLMs to perform this type of analysis for a one off project.
- Perfom qualitative data analysis using LLMs
- Use LLMs to generate key words to help with metadata generation

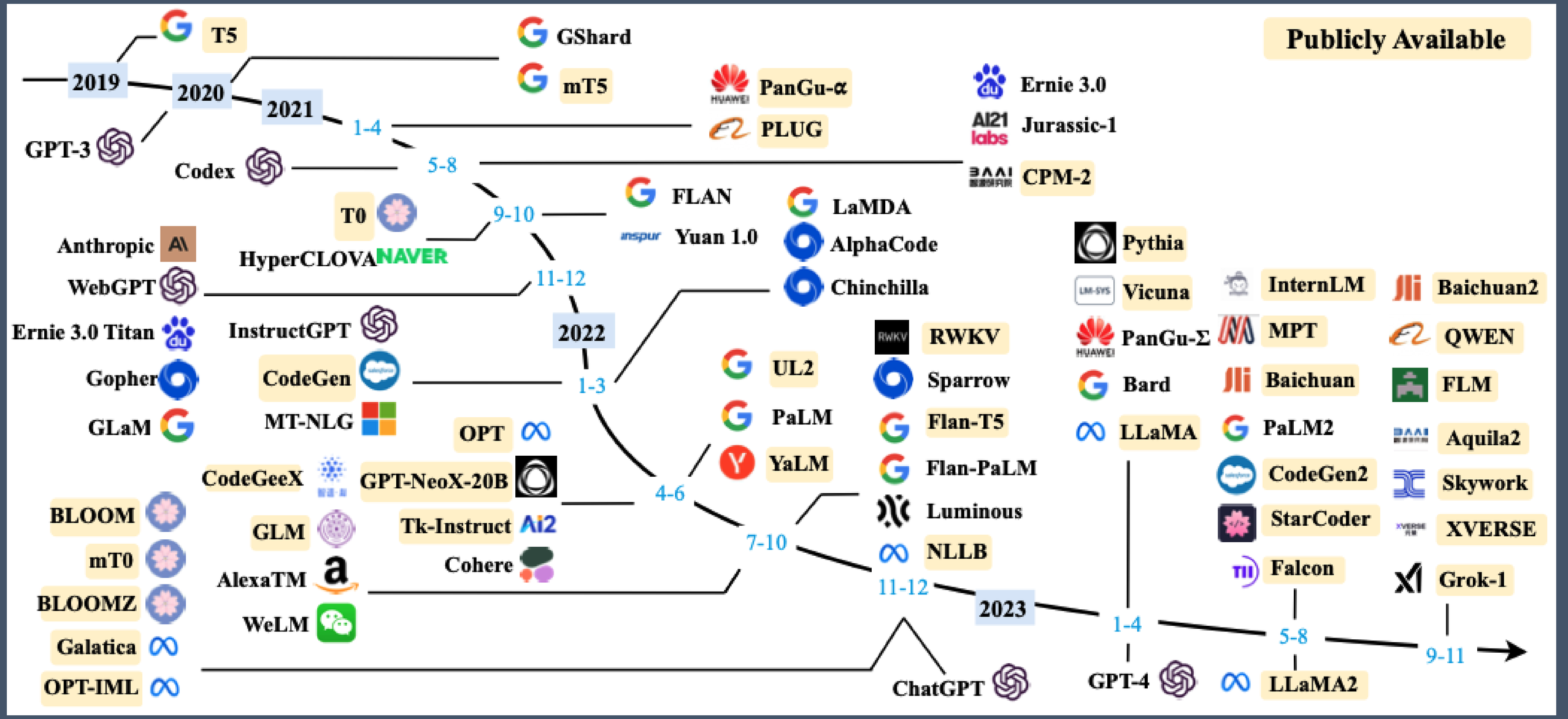
A Typical RAG Based LLM App Architecture



Main Steps in LLM Project Development



With So Many LLMs On The Market, How Do You Choose?



Source: A comprehensive survey of LLMs

A Brief Guide on Selecting LLMs

What Are Important Factors to Consider When Selecting LLM for a Project

1. **Task alignment**: Choose an LLM that aligns to the task, such as GPT for conversational applications or BioBERT for biomedical research.

2. **Training data**: Evaluate whether the LLM has been trained on data that matches the domain or context of the project.

3. **Model size and complexity**: Models with tens of billions of parameters provide higher-quality outputs but require more computational resources.

4. **Adapting and tuning:** Determine if the chosen LLM can be effectively contextualized with prompts or fine-tuning.

5. **Ecosystem and support:** Investigate the availability of resources, tools, and community support surrounding the LLM.

Multi-model workflows are also becoming common. Multi-model is when an app leverages multiple LLMs

Key Considerations in Deployment

Important Factors to Consider When Bringing LLM Apps Into Production

Data Pipelines

Data pipelines ensure smooth data flow through efficient ingestion, preprocessing, training, and inference.

- Semantic caching involves temporarily storing data embeddings. This enables AI systems to deliver more precise, meaningful, and efficient responses.
- Feature injection involves adding extra information to an AI model to enhance its performance and reasoning.
- Context retrieval involves retrieving relevant contextual informa-tion to enhance the understanding of AI models

Key Factors for Inference

Processing for inference involves running the necessary computations to apply a trained model to new data and generate predictions or outcomes.

- Infrastructure, access and consumption. his involves selecting suitable hardware infrastructure; deciding where to host the model and how to expose the model
- Reducing latency. Latency is the time it takes for an LLM to make predictions after receiving input data. This is crucial for gen AI projects that demand real-time responses
- Calculating costs. Make sure you know key factors driving costs of running and maintaining the model

User Interfaces

The front-end user interface (UI) of a gen AI application provides users with a way to input data, receive output from the processing engine, and control the application's behavior.

LLM Agent Orchestrator

Orchestration frameworks coordinate AI agents and other components to achieve specific goals. AI agents are individual instances of language models tasked with activities like text summarization, language translation, and sentiment analysis. These agents are managed within an orchestration system to perform complex language processing tasks.

- Most Cloud platforms can have their own orchestration frameworks
- LangChain has emerged as a good open source LLM orchestration framework

User Interfaces for LLM Apps

The front-end user interface (UI) of a generative AI application allows users to input data, obtain output from the processing engine, and manage the application's functionality. Most UIs incorporate elements from the following components:

- **Web apps** are the most common type of front end for gen AI apps because they're relatively easy to develop and can be accessed from any device with a web browser.
- Mobile apps tailored to specific devices, such as tablets and smartphones, offer a more immersive and engaging experience for users. They can take advantage of the unique aspects of each platform and can cache data for offline use.
- **Chat interfaces** are used in gen AI apps when the app needs to converse with the user, such as to answer questions or assist with certain tasks.
- Desktop apps are useful for gen AI apps that require a lot of processing power, or that need to access local resources.
- **APIs and Command-line interfaces (CLIs)** are sometimes used for gen AI apps that are accessed by developers and data scientists, such as to empower software engineers to generate code.

Ways to Utilize and Access LLMs

Three Main Ways to Utilize LLMs

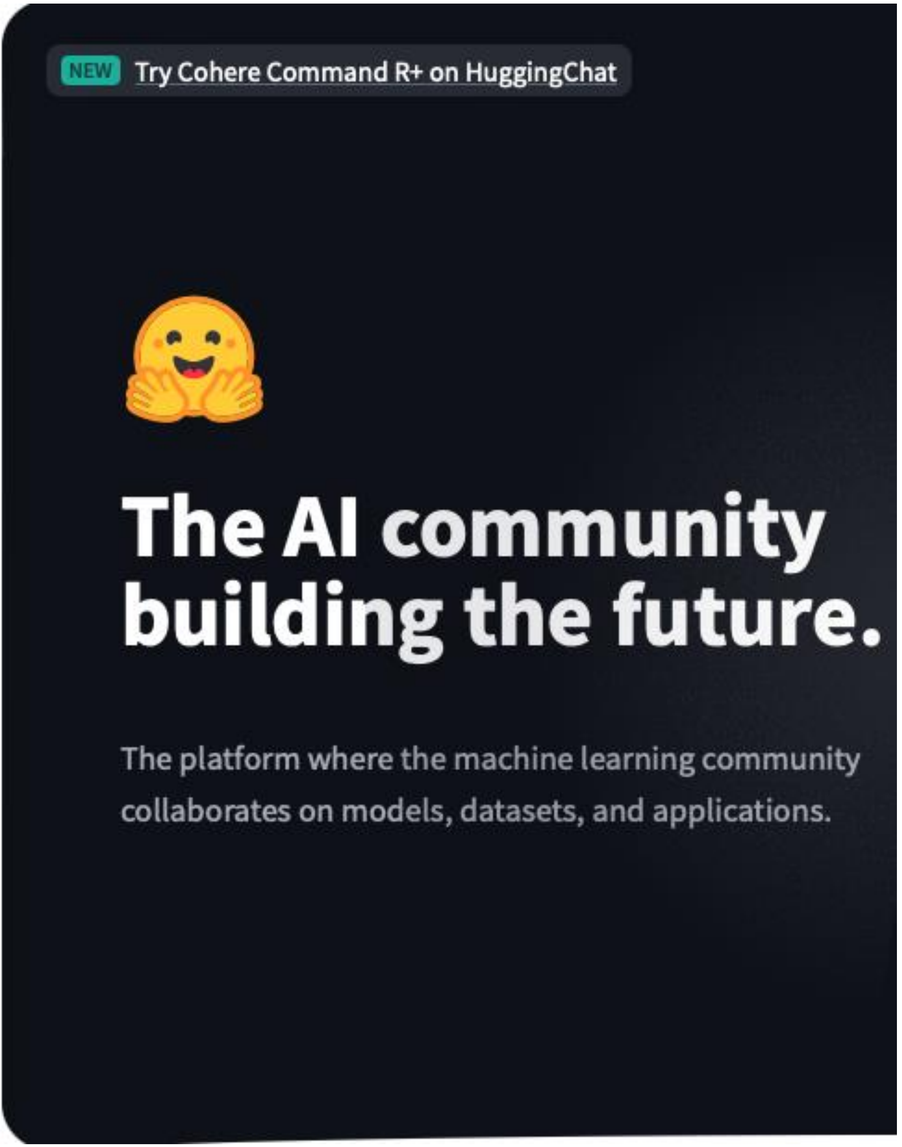
1. Personal Productivity Boost
2. Build Apps for Users within your organization
3. Build Apps for Users outside your organization

Ideal use case

Access Mode	Personal Productivity	Building apps for users
Gen AI Platform UI (e.g., ChatGPT)	✓	✗
Gen AI Programming API (e.g., OpenAI API)	✗	✓
Third Party (Cloud) API Platforms	✗	✓
Third Party Platforms (e.g., HuggingFace)	✗	✓
Host LLMs Locally	✗	✓

Accessing Pre-trained LLMs with Hugging Face

Summary of the Hugging Face Platform



- Products and Services:
  - Transformers Library: An open-source library offering a wide range of pre-trained models for NLP tasks such as text classification, translation, summarization, and question answering.
  - Datasets Library: A library for easily accessing and managing large datasets for machine learning.
  - Tokenizers Library: Tools for efficient tokenization of text data, crucial for preparing inputs for NLP models.
  - Model Hub: A platform hosting thousands of pre-trained models that can be easily integrated into various applications.
  - Inference API: Provides hosted API endpoints for using pre-trained models in production without the need for extensive infrastructure.

- Community and Contributions:
  - Hugging Face fosters a vibrant community of developers and researchers who contribute to the continuous improvement and expansion of its libraries.
  - Hosts regular events, webinars, and workshops to educate and engage with the AI and NLP community.

Accessing Pre-trained LLMs with Hugging Face

Whats the Hugging Face Advantage?

Sentiment Analysis with LLMs in Hugging Face

```
from transformers import pipeline


sentiment_classifier = pipeline("text-classification")

outputs = sentiment_classifier("""Hey, I went to Tunis. My visit was amazing.
I visited so many nice places and I enjoyed the food and meeting people there""")

print(outputs)
```

```
[{'label':
'POSITIVE',
'score':
0.9998677}]
```

1. Free and open, open Source and Community-Driven.
    1. Open-Source Libraries: Hugging Face's libraries are open-source, allowing for transparency, customization, and community contributions.
    2. Active Community: A vibrant community of developers and researchers contributes to the continuous improvement and expansion of the libraries.

2. Easy to use.
    1. User-Friendly APIs: Simple and intuitive APIs make it easy to integrate NLP models into applications with minimal effort.
    2. Extensive Documentation: Detailed documentation and tutorials help users quickly get started and effectively utilize the tools.

3. Model hub , data repository and documentation
    1. Centralized Repository: The Model Hub hosts thousands of models, making it easy to find and deploy models that suit specific needs.
    2. Versioning and Deployment: Facilitates version control and deployment of models, ensuring consistency and reliability in production environments
    3. Datasets are easy to access and use.

Accessing Pre-trained LLMs with Hugging Face

Let's Explore Hugging Face

1. Create account. Create a Hugging Face account
2. Identifying LLMs and other models. Explore the model hub
    1. Explore models by their name. Look and identify the following models: Falcon, Llama, Mistral
    2. Explore models by task. Find the most popular sentiment-analysis model, machine translation model

3. Explore datasets
    1. Can you find dataset for your local language (other than English and French)
    2. Explore datasets for sentiment-analysis

4. Demo apps and spaces. Users create and share demo apps based on LLMs and other models.
    1. Find my demo Chichewa Transcription app

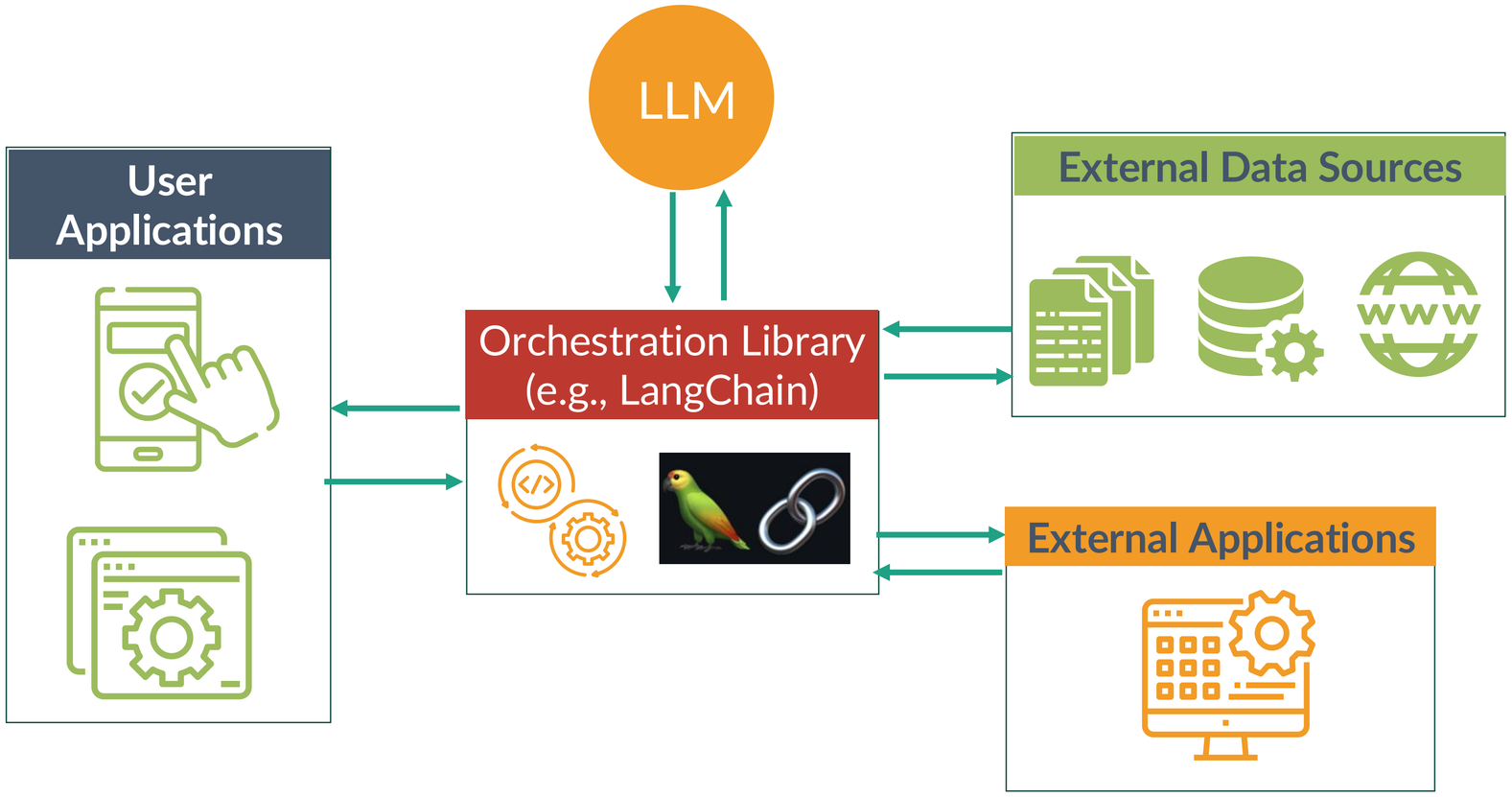
5. API key. Check and find your API key which we will use later.

Building LLM Apps with LangChain

LangChain is an LLM orchestrator. It provides tools and frameworks to manage, combine, and optimize the use of various LLMs. LangChain allows developers to chain together different models, APIs, and data sources to create more complex and capable AI applications. It simplifies tasks such as context management, prompt engineering, and integration of multiple models to achieve more sophisticated AI-driven solutions

- **Model Chaining:** Allows combining multiple language models (LLMs) to perform complex tasks by passing the output of one model as input to another.
- **Prompt Engineering:** Provides tools and templates for creating effective prompts, ensuring better performance and more accurate responses from LLMs.
- **Context Management:** Maintains and manages conversation context across multiple interactions, improving the relevance and coherence of responses.
- **API Integration:** Facilitates easy integration with various APIs, enabling the use of external data sources and services within the AI workflow.
- **Custom Workflows:** Supports the creation of custom workflows tailored to specific application needs, enhancing flexibility and functionality.
- **Logging and Debugging:** Includes features for logging and debugging, helping developers track and optimize the performance of their AI applications.

How Does LangChain Fit into the LLM App Ecosystem



Advantages of LangChain

1. Free and open source with growing community. Free to use and benefits from a collaborative community and ecosystem of developers,
2. Integrates and work with numerous other frameworks (including Hugging Face).
3. Ease of use. The abstraction offered by LangChain simplifies development and enables flexibility in choosing and switching components. Furthermore, by providing pre-built components, utilities, and best practices, LangChain reduces the complexity and effort involved in building LLM-powered applications.
4. Data Integration: It facilitates the integration of various data sources, including databases, APIs, and documents, allowing LLMs to access and utilize external information seamlessly.
5. Efficiency and Performance: LangChain optimizes the process of querying and retrieving information from data sources, ensuring that the LLMs can access relevant data quickly and efficiently. This results in faster response times and more accurate outputs.
6. Customization: Developers can customize how data is fetched, processed, and used by LLMs, tailoring the behavior of the application to specific use cases and requirements. This flexibility is essential for building specialized applications.

Accessing Multiple LLMs on Hugging Face Using LangChain


```python
from langchain_community.llms import HuggingFaceEndpoint
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate

# Set your Hugging Face API token
huggingfacehub_api_token = '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'

# ================================
# Define several LLM
# ================================
# 1. Falcon 7B
# ------------------------
model_falcon = 'tiiuae/falcon-7b-instruct'

# 2. Mistral 7B
# ------------------------
model_mistral = "mistralai/Mistral-7B-Instruct-v0.2"

# 3. LLama 3
# ------------------------
model_llama = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
# ================================================
# Define a chain which enables anwering of questions
# ================================================
llm = HuggingFaceEndpoint(repo_id=model_llama, huggingfacehub_api_token=huggingfacehub_api_token)

template = "You are an artificial intelligence assistant, answer the question. {question}"
prompt = PromptTemplate(template=template, input_variables=["question"])
llm_chain = LLMChain(prompt=prompt, llm=llm)

question = "What is Chichewa"

print(llm_chain.run(question))
```

Question: What is Chichewa?

TinyLlama-1.1B

language, and how can you help me translate a text? Chichewa Language: Chichewa is a Bantu language spoken by the Chichewa people in Malawi, Mozambique, and Zimbabwe. It is written in the ISO 639-3 code: mw. To help me translate a text, please provide me with the text in Chichewa language. Example Translation: Chichewa language is a beautiful language that gives me a sense of freedom. It has a rich cultural heritage that I cherish. Translation: Chichewa language is a language that provides me with a sense of freedom and cultural heritage. It has a rich and diverse cultural heritage that I cherish. Thank you for your help in translating the text.

Mistral-7B

Chichewa is a Bantu language spoken by the Chewa people in Malawi, Zambia, and Mozambique. It is the most widely spoken language in Malawi and Zambia. Chichewa is also known as Chewa, Nyanja, and Cibemba. The language is tonal and uses a variety of click sounds. It is a agglutinative language, which means that words are formed by adding affixes to the base. Chichewa is known for its complex grammar and extensive vocabulary. It is also used as a second language in some areas of Tanzania and Zimbabwe.

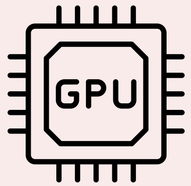
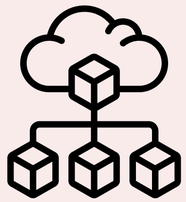
Falcon-7B

Chichewa is a programming language that was developed by MCPc, a company specializing in computer-based training and education. Chichewa was designed as an alternative to the popular BASIC programming language, with a syntax that is similar to BASIC but is more structured and easier to learn.

Summary of Ecosystem of Tools in LLM App Development

Name	Category	Functionality /Features	Cost/Pricing
AWS, GCP, Azure	Cloud provider	General purpose cloud services; LLM orchestration; custom LLMs	Platform specific
Hugging Face	LLM model hub	Access different pretrained LLM models; access fine-tuned LLMs; host models	Free and paid features
LangChain	LLM Orchestration	Allows integration with multiple data sources and multiple LLMs	Free and paid features
Pinecoe; Faiss; chroma;weaviate	Vector database	Integration with LLMs to provide vector embeddings storage and querying	Some are open source while most are proprietary
LlamaIndex (GPT Index)	LLM Orchestration	Facilitate the integration of large language models with various data sources	Free and paid features
Streamlit, Gradio, Flask	LLM App UI	Python based platforms for demo/prototype apps	Free and paid features
Twilio	API for WhatsAPP apps	Enables connecting LLM apps with WhatsApp	Free and paid features

In Practice Different Tools Working Together to Deliver an LLM App

LLM APP UI	
LLM APP FRONT END ENVIRONMENT	 Streamlit  Twilio  Meta WhatsApp API
LLM APP BACKEND DEV ENVIRONMENT	 LangChain +  Hugging Face +  Python
INFRASTRACTURE	  AMAZON WEB SERVICES (AWS)

Further Learning Resources

1. Hugging Face Learning Platform

Thank You, Feel Free to Ask Plenty of Questions

Dunstan Matekenya, PhD
Data Scientist
The World Bank Group, Washington DC

Whatsapp: +1 202-294-8588
Email: dmatekenya@worldbankgroup.org