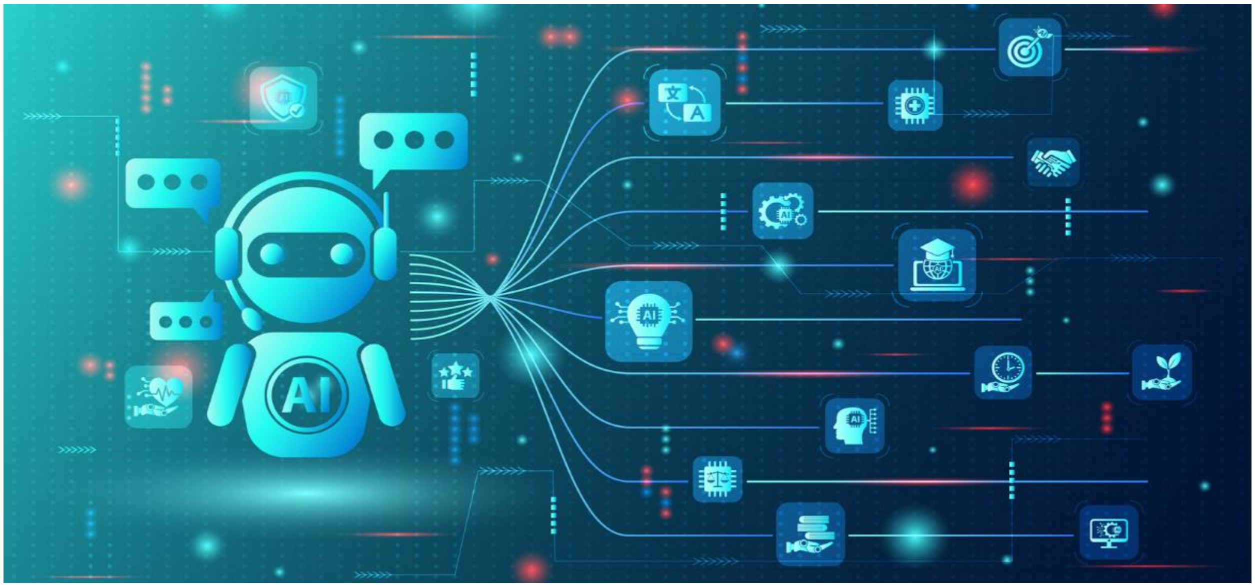
Agentic AI – Research Task



Presented by: Huma Mohsin

Roll # 00268003

Saturday (7 to 10) PM

Task 1 includes: -

1. What is Agentic AI?

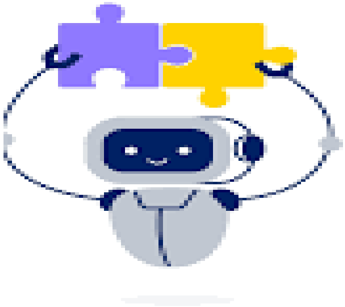
2. What are LLMs?

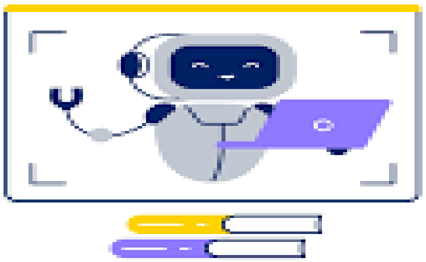
3. What is Generative AI?

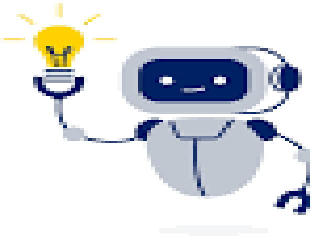
4. Difference between Generative AI and Agentic AI

5. OpenAI's Agents SDK – What it is, why it is used, Benefits.

1. What is Agentic AI?





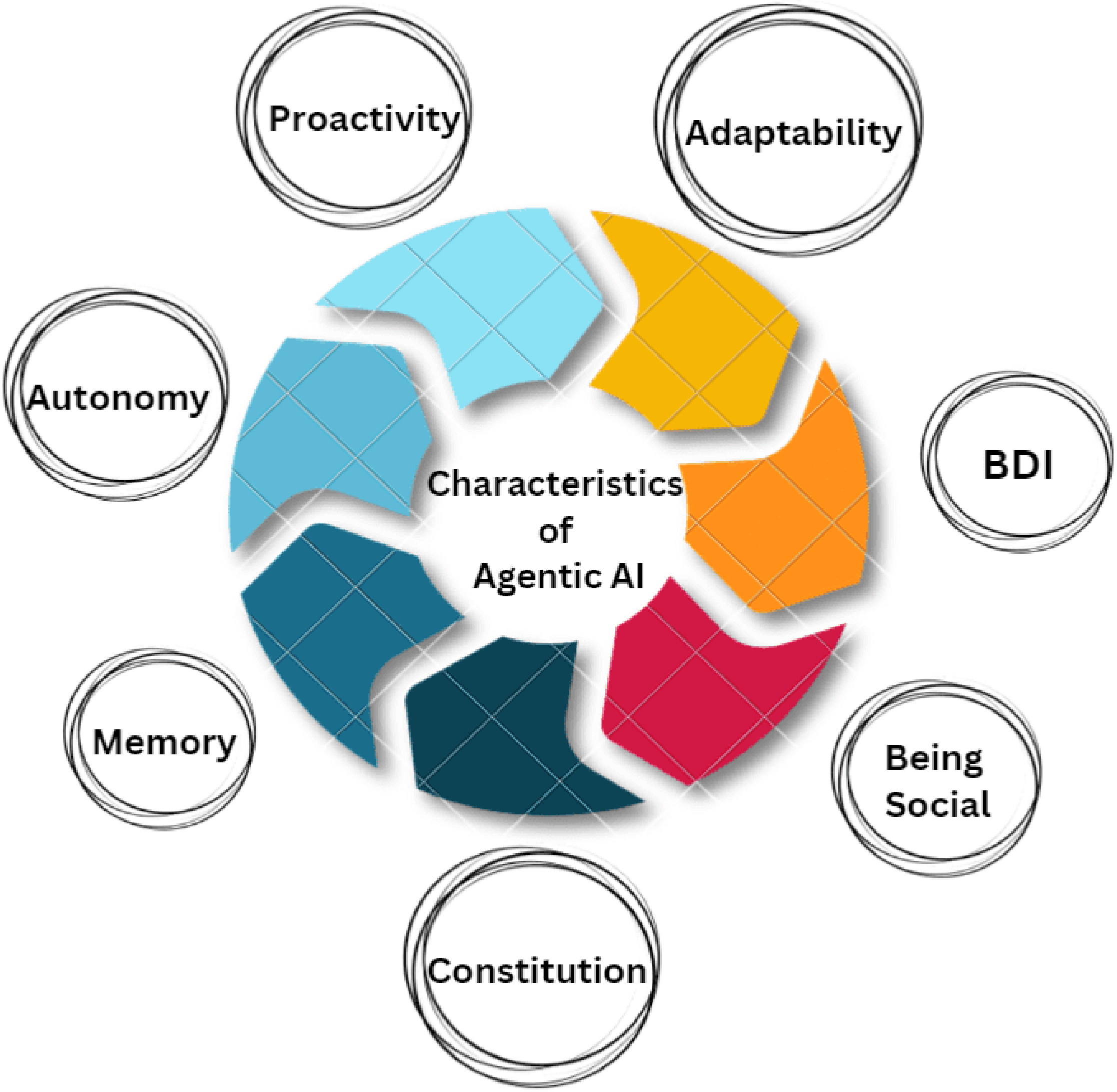


Agentic AI describes AI systems that can act independently, make decisions, and adapt to their environment with minimal human intervention.

- These systems, often referred to as AI agents, are capable of planning, reasoning, and executing actions autonomously, driven by goals and learning from interactions.
- Goal-oriented: completes tasks without step-by-step instructions
- Uses tools/APIs to interact with environments
- Example: Reads an email, checks calendar, drafts response

Key Characteristics of Agentic AI:

Agentic AI embodies a few critical characteristics that define its capabilities:



- **Autonomy:**

Agentic AI agents can operate independently, making decisions and performing actions without constant human guidance.

- **Contextual Understanding:**

They can understand the context of a situation and adapt their behavior accordingly.

- **Decision-Making:**

They can evaluate different options and make choices to achieve specific goals.

- **Learning and Adaptation:**

They can learn from their experiences and improve their performance over time.

- **Collaboration:**

They can collaborate with humans to solve complex problems or automate workflows.

- **Beliefs, Desires, and Intentions (BDI**):

A common model used in agent-oriented programming (AOP) is the BDI model, where agents are characterized by their beliefs (information about the world), desires (goals or objectives), and intentions (plans of action).

- **Being Social:**

  Agentic AI systems are designed to interact and collaborate with other agents and humans. Their social aspect enables them to engage in meaningful communication, negotiate, and cooperate with both digital and human entities to achieve common objectives.

- **Memory:**

  Long-term memory is essential for Agentic AI to store and retrieve previous experiences, interactions, and learnings. This memory allows agents to improve their decision-making over time, recall past decisions or actions, and learn from past mistakes.

How Agentic AI Works:

Perception: Collects data from its environment through sensors, APIs, databases, or user interactions to ensure up-to-date information.

Reasoning: Processes the collected data to extract meaningful insights using natural language processing (NLP), computer vision, or other AI capabilities.

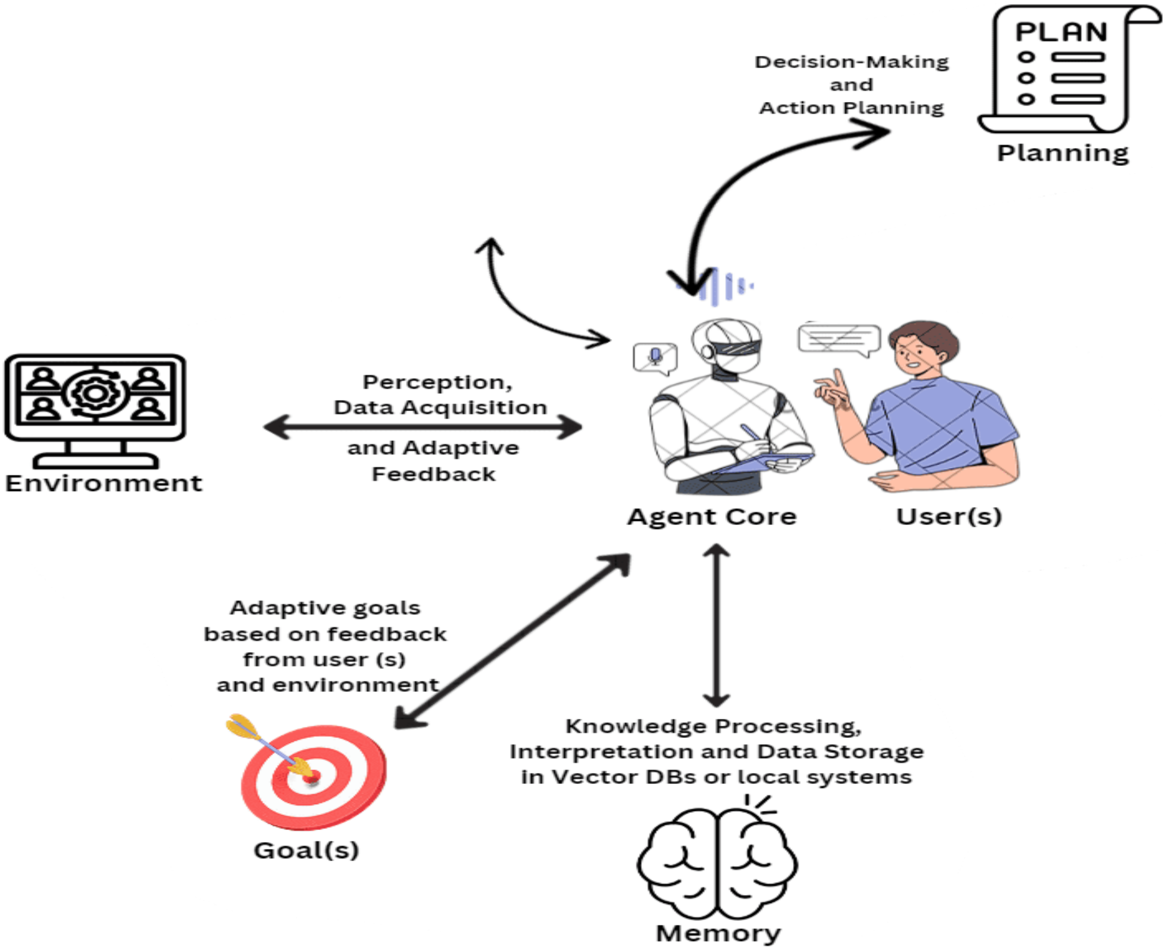
Goal Setting: Sets objectives based on predefined goals or user inputs and develops a strategy to achieve these goals.

Decision-Making: Evaluates multiple possible actions and chooses the optimal one based on factors such as efficiency, accuracy, and predicted outcomes.

Execution: Executes the selected action by interacting with external systems or providing responses to users.

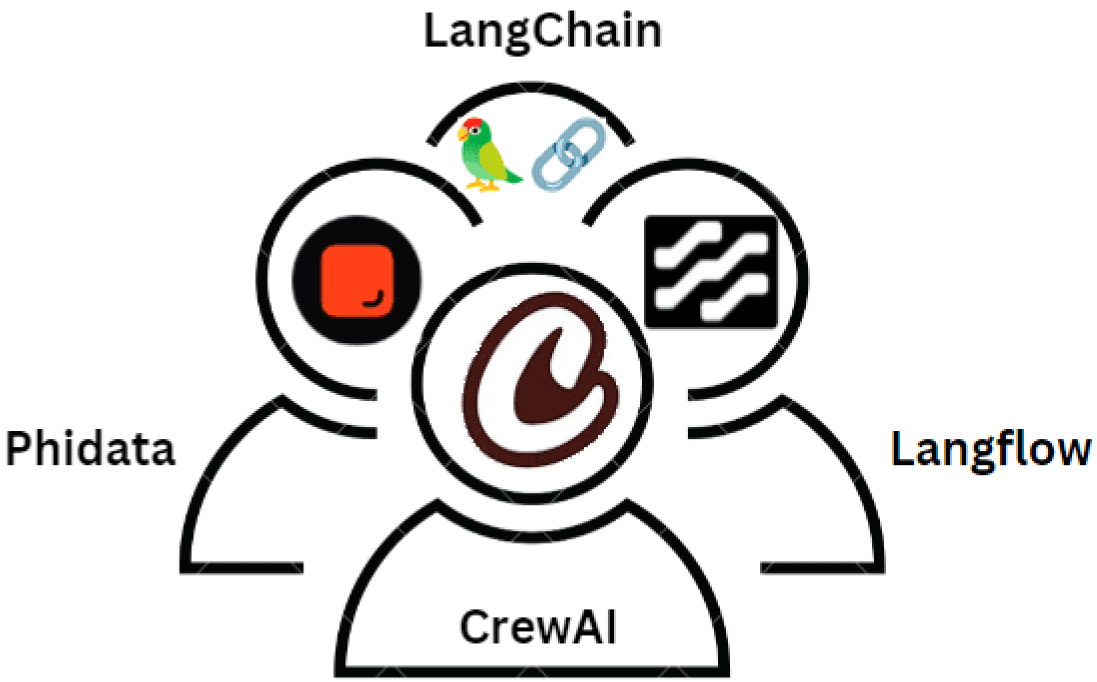
Learning and Adaptation: Evaluates the outcome of the action, gathers feedback, and refines its strategies over time.

Orchestration: Coordinates and manages systems and agents to automate AI workflows, track progress, manage resources, and handle failure events.



FRAMEWORKS FOR BUILDING AGENTIC AI

When developing Agentic AI systems, choosing the right framework is crucial for ensuring efficiency, scalability, and seamless integration with other technologies. Below are some of the most popular frameworks that we will be using for building AI agents:



Agentic AI Frameworks

- **CrewAI**: Its focus on natural language understanding, decision-making, and adaptability makes it a great choice for developing intelligent systems that can plan, learn, and execute tasks in dynamic environments.

- **Phidata**: Phidata is designed for creating agents that combine data-driven insights with decision-making processes. Its ability to integrate large datasets (vast structured and unstructured data) and perform real-time analysis allows agents to work more efficiently and humanely.
- **OpenAI Gym**: OpenAI Gym is widely used to develop reinforcement learning (RL) agents. It provides a flexible environment for testing and training AI agents, making it a great option for building agents that need to learn through interaction with their environment.
- **Langflow**: It excels in allowing users to visually define complex workflows, providing intuitive tools for managing multiple interactions, and ensuring seamless communication between agents and external systems.

CONCLUSION

Agentic AI is transforming the landscape of intelligent systems, shifting from traditional reactive models to proactive, autonomous agents. With capabilities like autonomy, proactivity, and adaptability, these systems can make decisions, plan actions, and evolve based on real-time feedback, enabling them to handle complex, dynamic tasks.

The frameworks available for developing AI agents, such as **CrewAI**, **Phidata**, and **Langflow**, provide powerful tools for building autonomous systems capable of interacting with both humans and other agents. Agentic AI offers immense potential to automate tasks, streamline workflows, and drive intelligent decision-making. As this technology evolves, it will continue to redefine the role of AI in our daily lives and industries, turning machines into capable collaborators and decision-makers.

2. What are LLMs?

LLMs are powerful AI models trained on a vast amount of text data (like books, websites, and articles). Their goal is to **understand and generate human-like language**.

Some key features:

- Use **deep learning**, especially transformer architectures

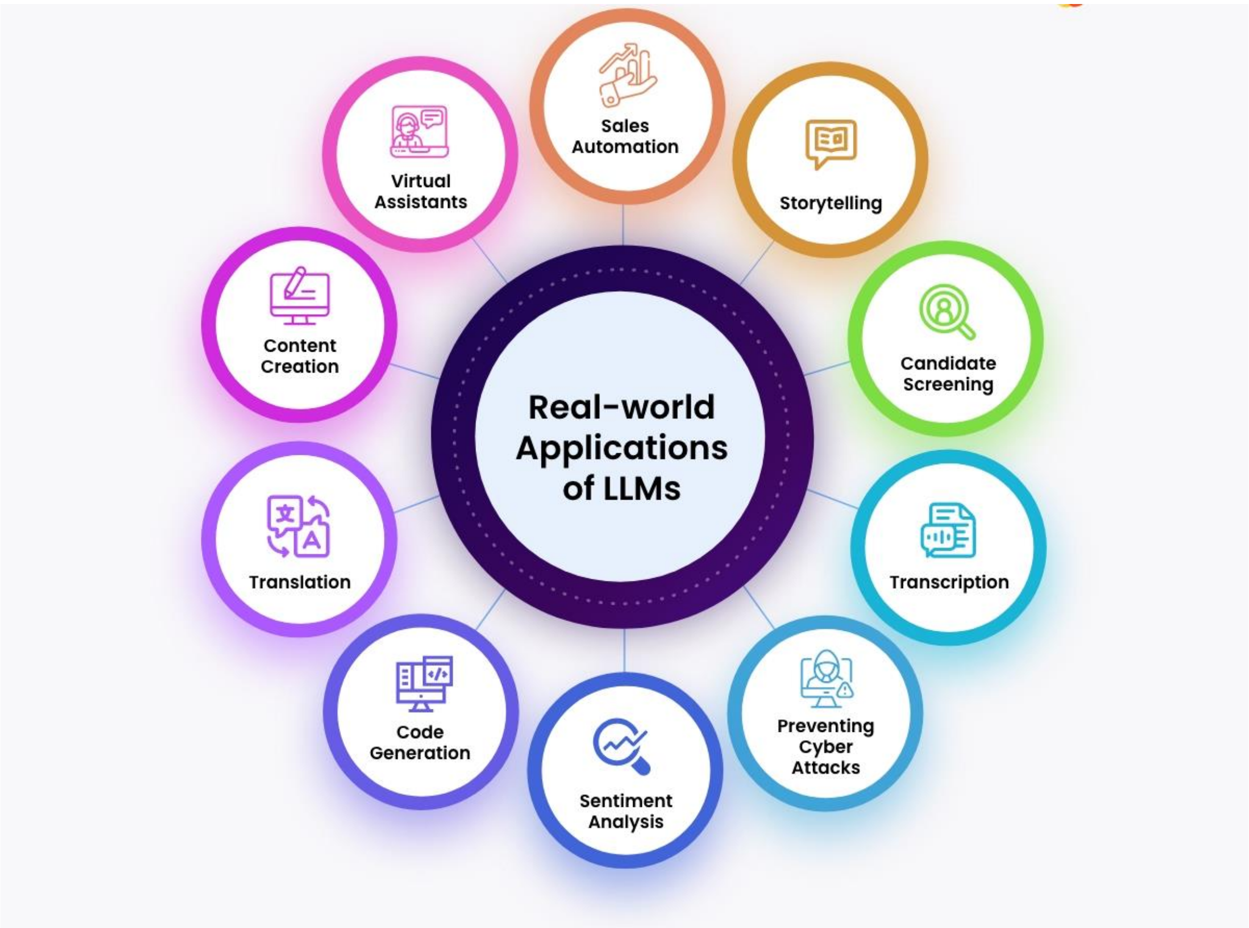
- Predict the **next word** in a sentence, allowing them to generate coherent text
- LLM applications can perform numerous tasks including writing essays, creating poetry, coding, and even engaging in general conversation.

Famous LLMs:

- **GPT-4 / GPT-3.5** by OpenAI
- **Claude** by Anthropic
- **Gemini (Bard)** by Google
- **LLaMA** by Meta

💡 LLMs are the **foundation** of Agentic AI. They provide the **reasoning and language capabilities**, while agents give them the **structure and autonomy** to act in real-world scenarios.

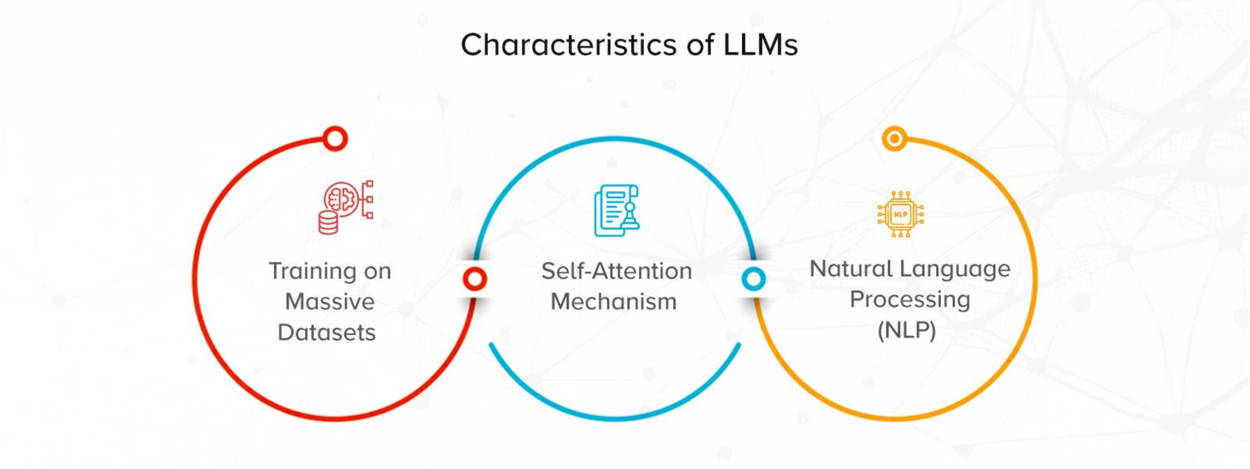
Most popular applications of large language models:



1. **Chatbots** – Provide human-like conversations for customer support or virtual assistance.
2. **Content Creation** – Generate text for blogs, ads, emails, and marketing material.
3. **Translation** – Translate text between multiple languages accurately and fluently.
4. **Summarization** – Condense large texts into concise summaries.
5. **Sentiment Analysis** – Detect emotions or opinions expressed in text.

6. **Code Generation –** Write and suggest code in various programming languages.
7. **Search Engines –** Improve search relevance using semantic understanding.
8. **Tutoring –** Act as personal tutors by explaining concepts and answering questions.
9. **Virtual Assistants –** Automate scheduling, reminders, and task management.
10. **Medical Diagnosis –** Assist in analyzing symptoms and suggesting possible diagnoses.

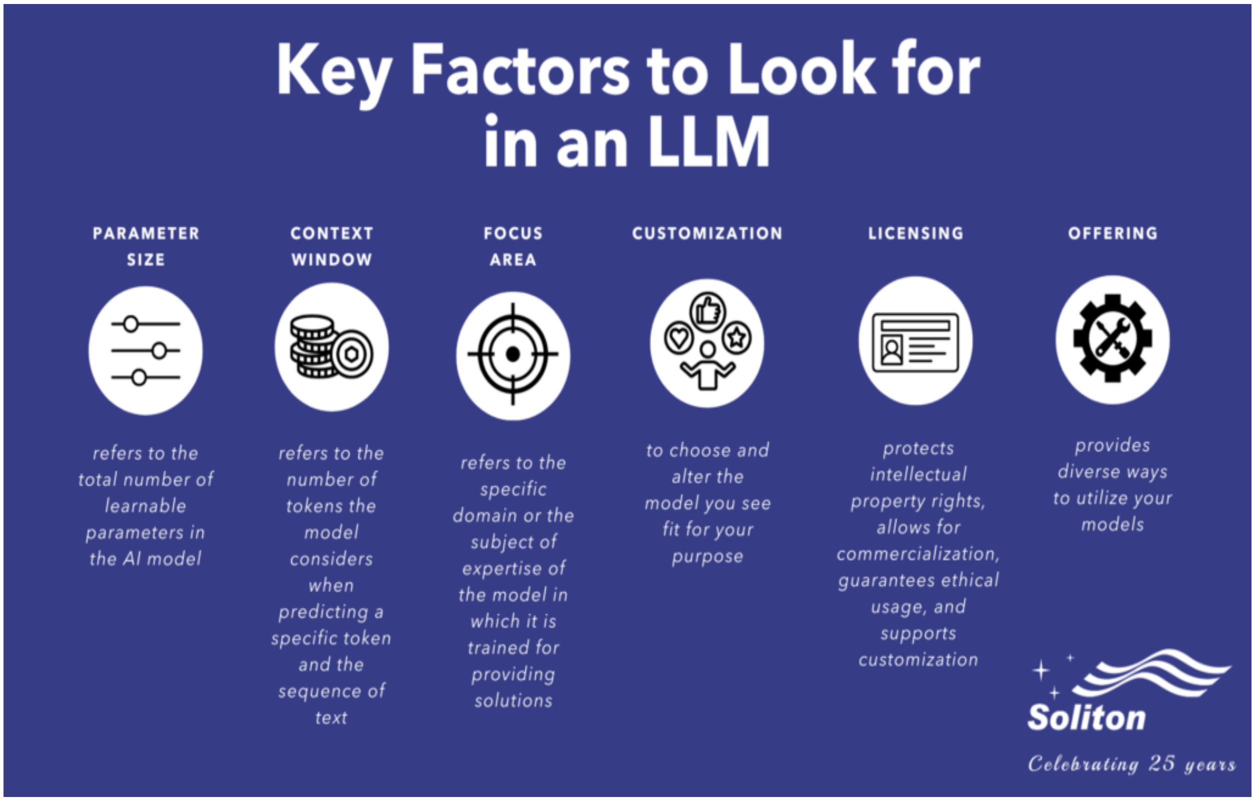
Key Characteristics of LLMs:


Characteristics of LLMs
Training on Massive Datasets
Self-Attention Mechanism
Natural Language Processing (NLP)
NLP

- **Training on Massive Data:** LLMs are trained on diverse and vast datasets including books, websites, articles, and social media text, enabling broad knowledge and language understanding.
- **Natural Language Processing (NLP):** They perform advanced NLP tasks such as text generation, translation, summarization, and sentiment analysis with high accuracy.

- **Self-Attention Mechanism:** Built on the Transformer architecture, LLMs use self-attention to understand relationships between words and maintain context across long text sequences.
- **Scalability:** The models scale up with billions of parameters, improving performance and adaptability to various tasks.
- **Context Awareness:** Capable of understanding and responding based on the context of previous inputs or conversations.
- **Transfer Learning:** Can be fine-tuned for specific tasks after general training, saving time and resources.

Some Key Factors to Look for in an LLM



Key Factors to Look for in an LLM

PARAMETER SIZE

refers to the total number of learnable parameters in the AI model

CONTEXT WINDOW

refers to the number of tokens the model considers when predicting a specific token and the sequence of text

FOCUS AREA

refers to the specific domain or the subject of expertise of the model in which it is trained for providing solutions

CUSTOMIZATION

to choose and alter the model you see fit for your purpose

LICENSING

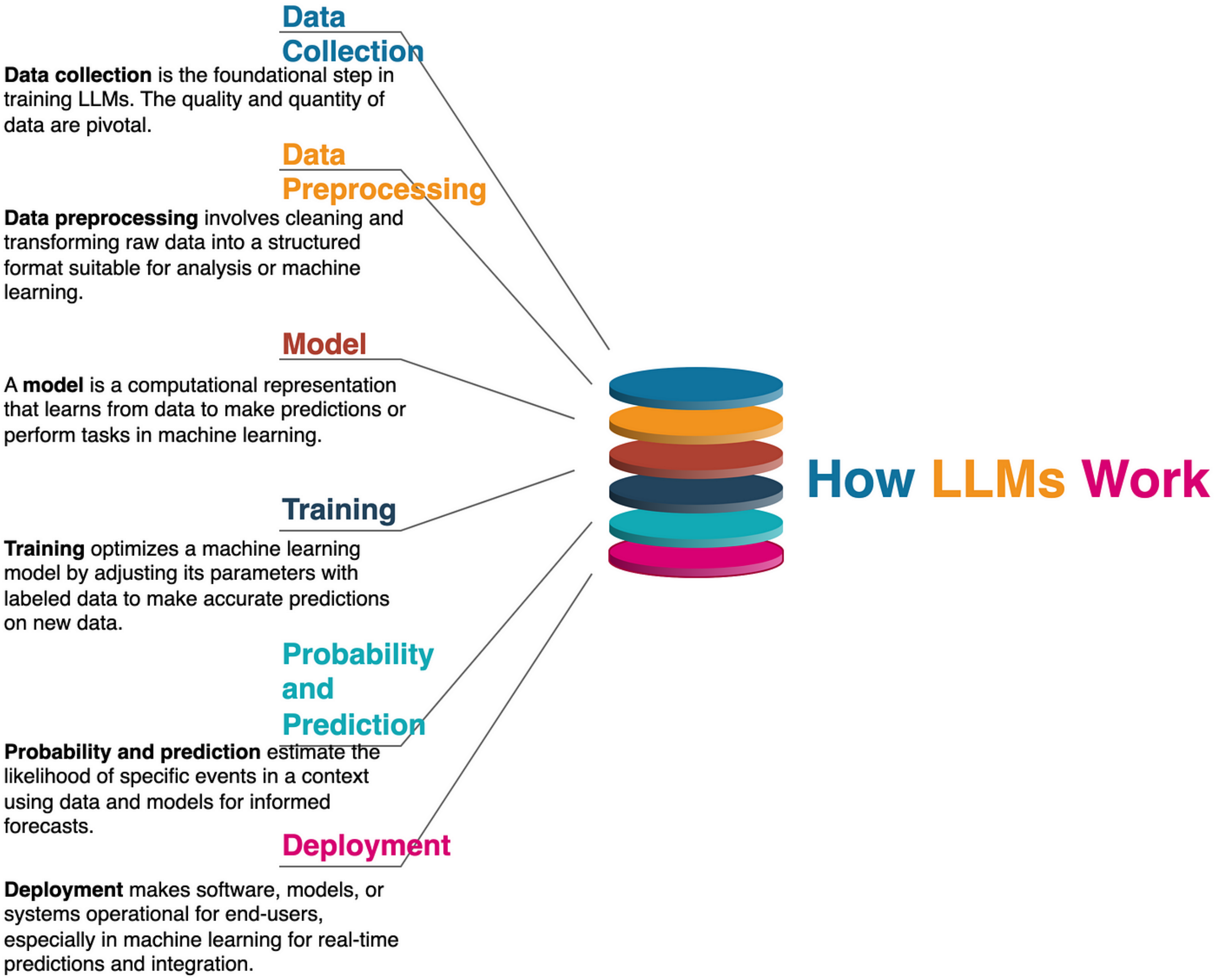
protects intellectual property rights, allows for commercialization, guarantees ethical usage, and supports customization

OFFERING

provides diverse ways to utilize your models

HOW LLMS WORK:

LLMs use deep learning (especially the **Transformer architecture**) to learn patterns in language. They break down sentences into tokens, analyze relationships between words, and generate responses based on probabilities of what comes next.



Data Collection

Data collection is the foundational step in training LLMs. The quality and quantity of data are pivotal.

Data Preprocessing

Data preprocessing involves cleaning and transforming raw data into a structured format suitable for analysis or machine learning.

Model

A **model** is a computational representation that learns from data to make predictions or perform tasks in machine learning.

Training

Training optimizes a machine learning model by adjusting its parameters with labeled data to make accurate predictions on new data.

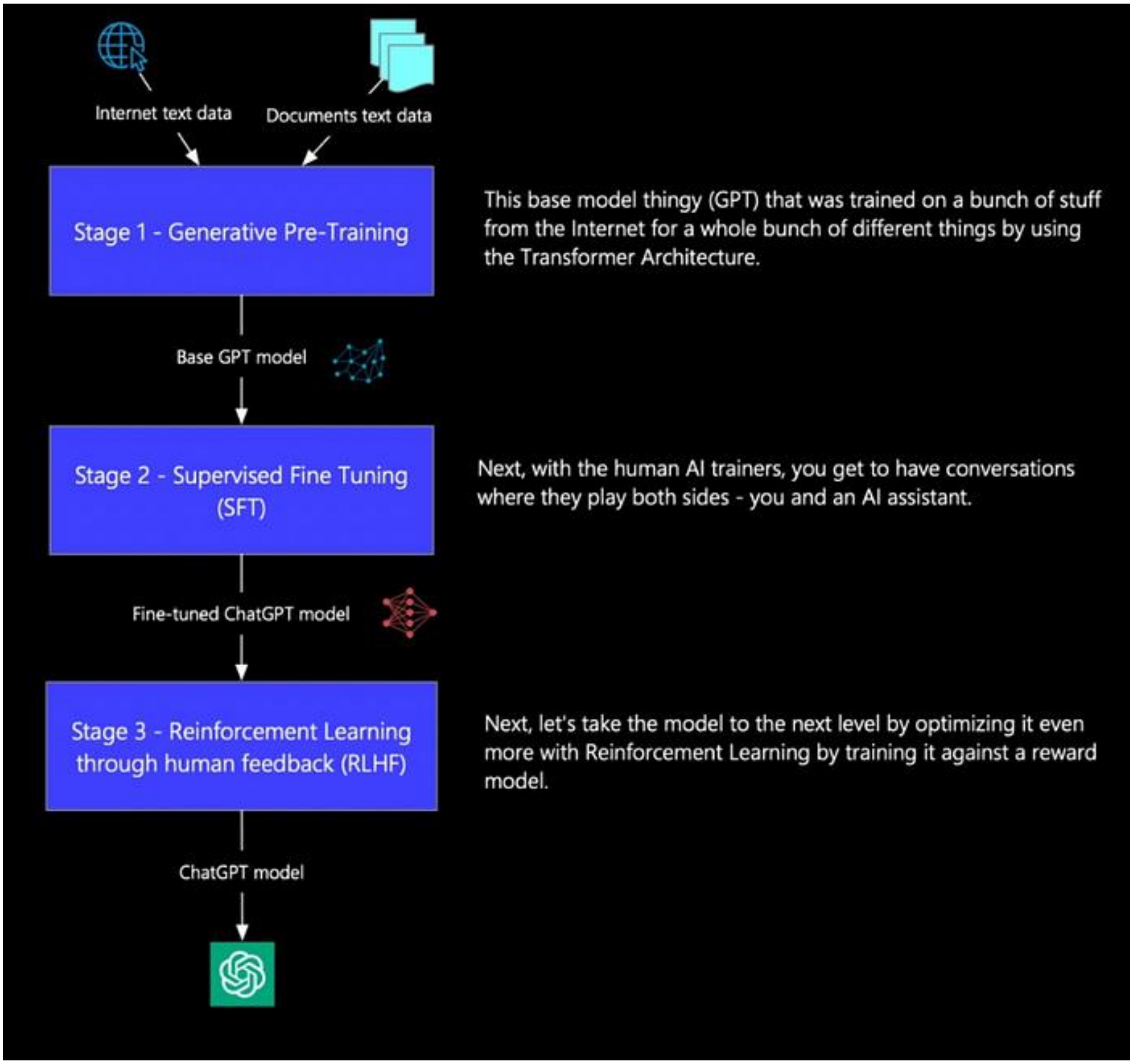
Probability and Prediction

Probability and prediction estimate the likelihood of specific events in a context using data and models for informed forecasts.

Deployment

Deployment makes software, models, or systems operational for end-users, especially in machine learning for real-time predictions and integration.

HOW LLM MODELS ARE TRAINED


Internet text data
Documents text data
Stage 1 - Generative Pre-Training
Base GPT model
Stage 2 - Supervised Fine Tuning (SFT)
Fine-tuned ChatGPT model
Stage 3 - Reinforcement Learning through human feedback (RLHF)
ChatGPT model

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

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

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

STAGES OF TRAINING:

Stage 1 — Pre-training:

- In this phase, Large Language Models (LLMs) like GPT-3 are trained on a massive dataset from the internet to predict the next word in a sequence of text.
- The data is cleaned, preprocessed, and tokenized, and transformer architectures are commonly used for this purpose.
- The model learns language patterns but doesn't yet understand instructions or questions.

Stage 2 — Supervised Fine-Tuning or Instruction Tuning:

- In this stage, the model is provided with user messages as input and AI trainer responses as targets.
- The model learns to generate responses by minimizing the difference between its predictions and the provided responses.
- It begins to understand instructions and learns to retrieve knowledge based on them.

Stage 3 — Reinforcement Learning from Human Feedback (RLHF):

- RLHF is applied as a second fine-tuning step to align the model with human preferences, focusing on being helpful, honest, and harmless (HHH).
- RLHF helps improve the model's behavior and alignment with human values, ensuring it provides useful, truthful, and safe responses.

This involves two sub-steps:

- **Training Reward Model Using Human Feedback:** Multiple model outputs for the same prompt are generated and ranked by human labelers to create a reward model. This model learns human preferences for HHH content.
- **Replacing Humans with Reward Model for Large-Scale Training:** Once the reward model is trained, it can replace humans in labeling data. Feedback from the reward model is used to further fine-tune the LLM at a large scale.

EXAMPLES OF LLMS

1. GPT-3 (Generative Pre-Trained Transformer 3):

- Developed by OpenAI, GPT-3 is known for its ability to perform a wide range of language tasks, including translation, summarization, and question answering.

2. BERT (Bidirectional Encoder Representations from Transformers):

- Developed by Google, BERT excels in understanding the context of words in a sentence, making it effective for tasks like search query understanding and sentiment analysis.

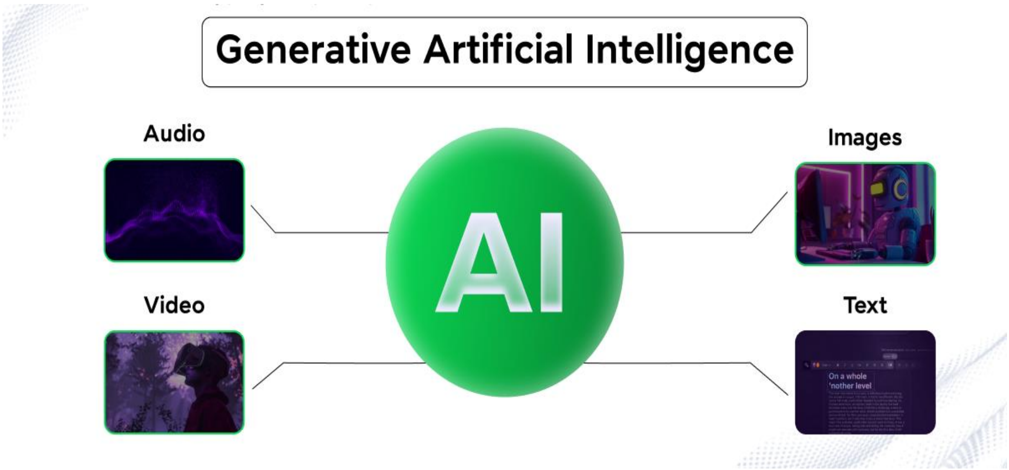
3. T5 (Text-to-Text Transfer Transformer):

- Also developed by Google, T5 treats every NLP task as a text generation problem, enabling a unified approach to various tasks such as translation, summarization, and classification.

4. PaLM (Pathways Language Model):

- Another model by Google, PaLM is designed for large-scale understanding and generation of language, with a focus on improved efficiency and performance.

3. What is Generative AI?

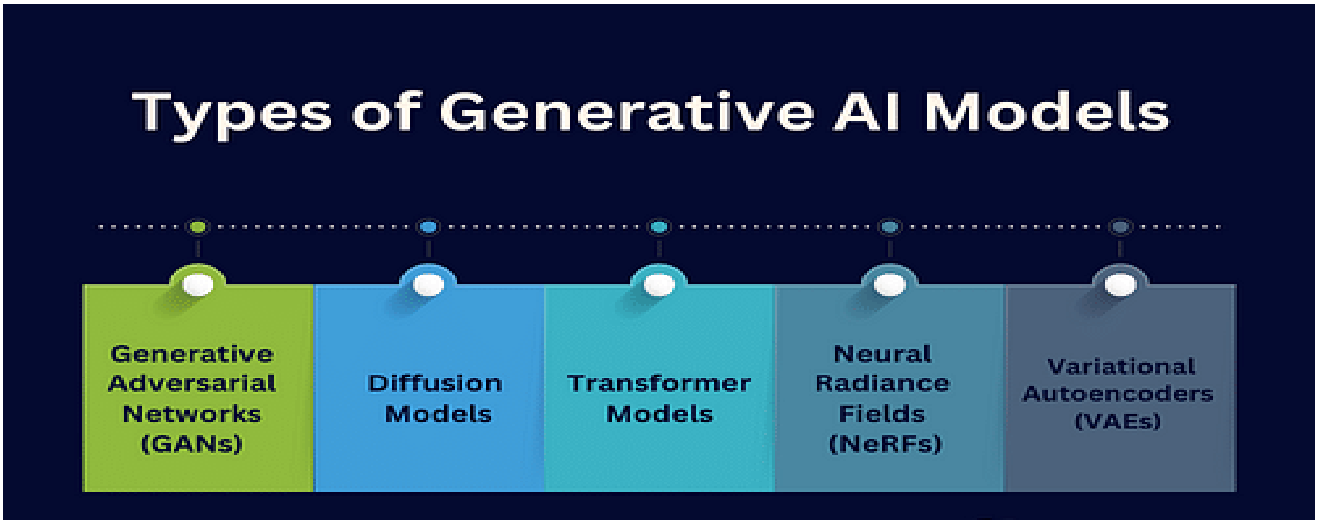


Generative AI is a type of artificial intelligence that creates new content, such as text, images, music, and videos, based on data it has been trained on. Unlike traditional AI that analyzes existing data, Generative AI produces novel outputs by predicting patterns and making new creations.

Key Concepts

1. Generative Models:

These are algorithms trained on massive datasets to learn patterns and generate new content.


Types of Generative AI Models
Generative Adversarial Networks (GANs)
Diffusion Models
Transformer Models
Neural Radiance Fields (NeRFs)
Variational Autoencoders (VAEs)

□ **Generative Adversarial Networks (GANs):**
A generative model using two competing neural networks—generator and discriminator—to create realistic data like images and text.

□ **Diffusion Models:**
A generative technique that adds and removes noise to create high-quality data samples, commonly used in image generation.

□ **Transformer Models:**
Deep learning models that use self-attention to understand and generate natural language, forming the core of Large Language Models (LLMs).

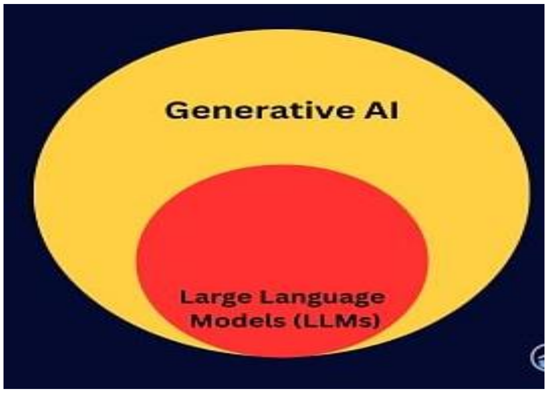
□ **Neural Radiance Fields (NeRFs):**
A technique that reconstructs photorealistic 3D scenes from 2D images using neural networks and multi-view analysis.

□ **Variational Autoencoders (VAEs):**
A generative model that encodes data into latent space and decodes it to generate new, diverse content like images or text.

2. Foundation Models:

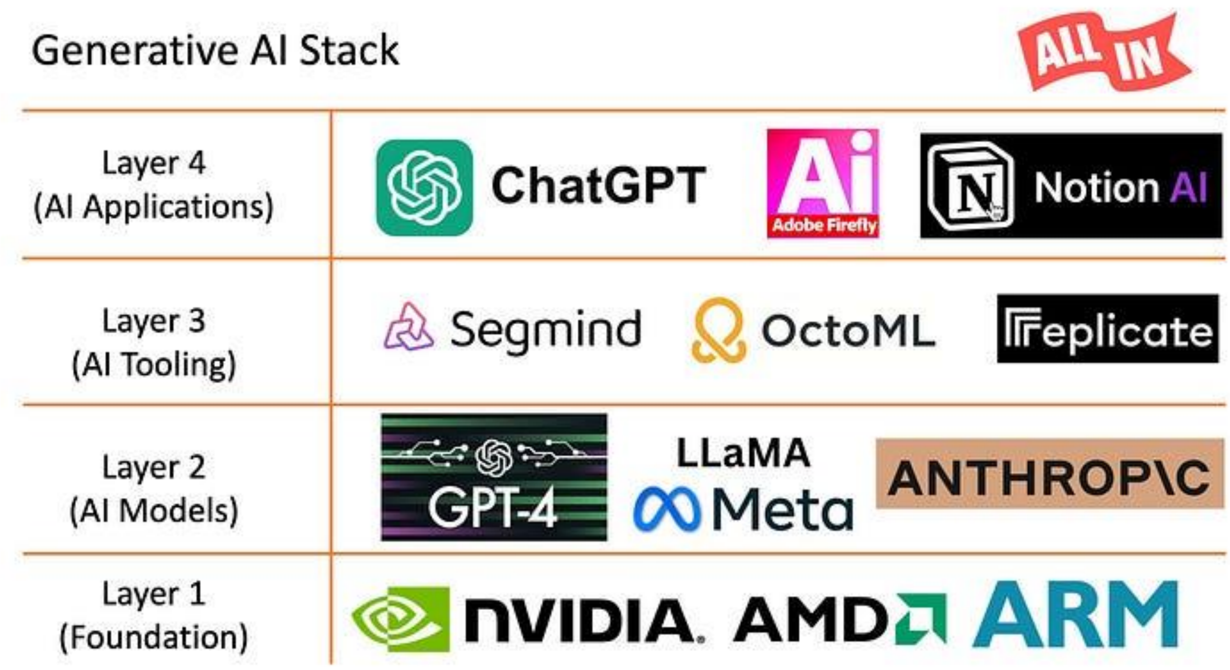
Large AI models that can perform multiple tasks and are used to power Generative AI applications.

3.Training Data:



Generative AI models are trained on vast amounts of text, images, audio, or video data to learn the underlying patterns and structures.

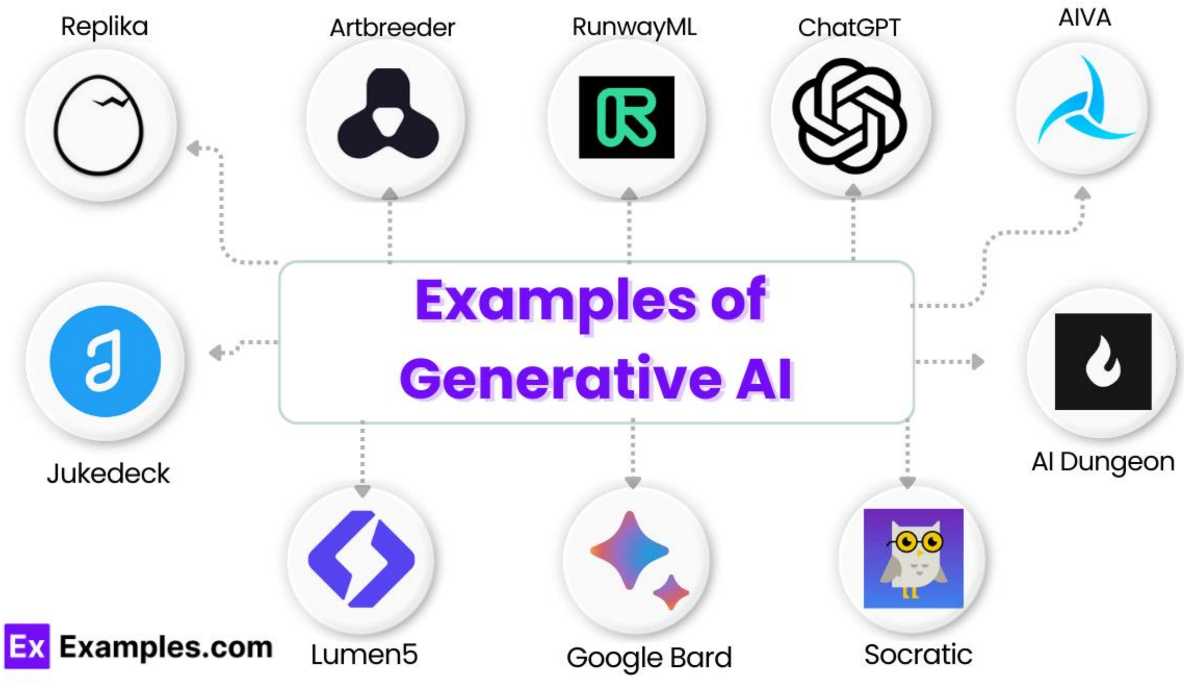
Generative AI Stack:



Generative AI Stack

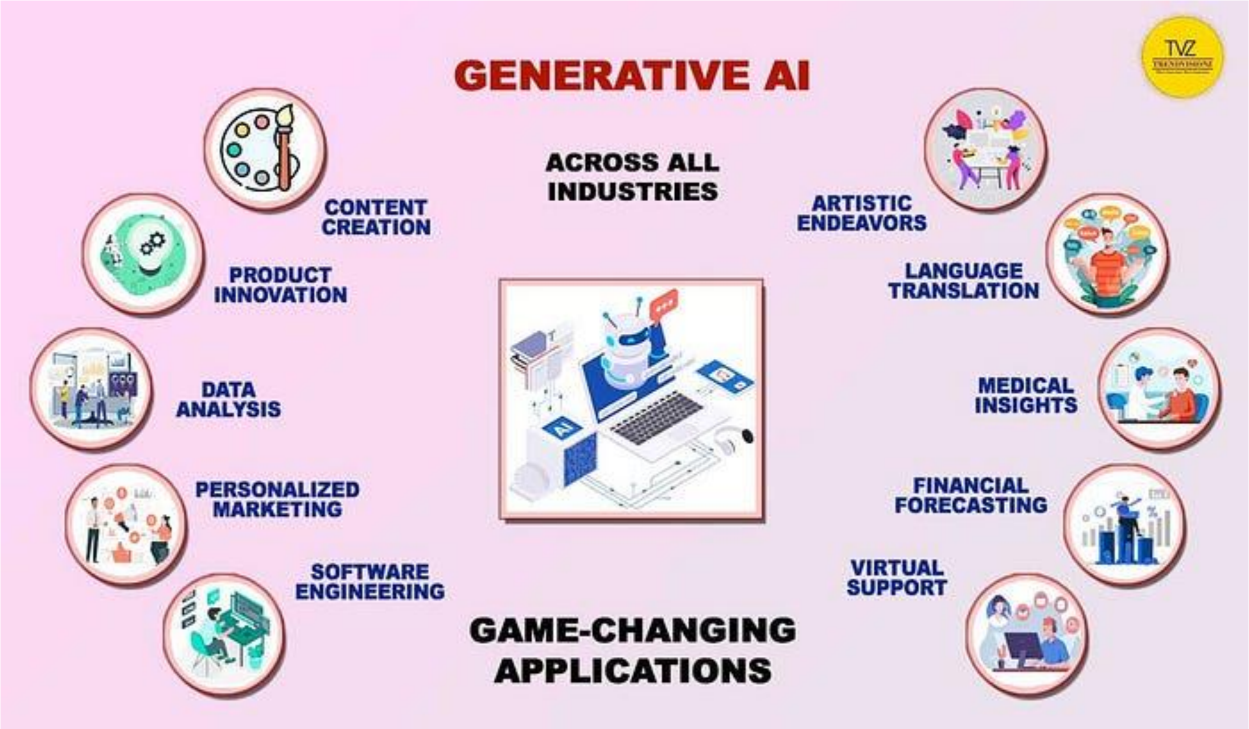
Layer 4 (AI Applications)	ChatGPT, Ai Adobe Firefly, Notion AI
Layer 3 (AI Tooling)	Segmind, OctoML, replicate
Layer 2 (AI Models)	GPT-4, LLaMA Meta, ANTHROP\C
Layer 1 (Foundation)	NVIDIA, AMD, ARM

Examples of Generative AI:



- **Text Generation:** Creating articles, poems, scripts, or code.
- **Image Generation:** Producing realistic or artistic images.
- **Music Generation:** Composing new melodies, harmonies, or arrangements.
- **Video Generation:** Creating videos from text prompts or other inputs

TOP USE CASES OF GENERATIVE AI (GENAI):



1. **Content Generation:**
   Create blog posts, social media captions, emails, and marketing content using tools like ChatGPT and Jasper.
2. **Image Generation:**
   Generate realistic or artistic images from text prompts using models like DALL·E, Midjourney, and Stable Diffusion.

3. **Code Generation:**
   Write or complete code snippets using tools like GitHub Copilot or Amazon CodeWhisperer.
4. **Chatbots and Virtual Assistants:**
   Power intelligent, conversational agents for customer support and personal productivity.
5. **Video and Audio Creation:**
   Create music, voiceovers, and even full videos with tools like Synthesia, Descript, or Suno AI.
6. **Product Design and Prototyping:**
   Generate visual designs and UI/UX prototypes based on user requirements.
7. **Personalized Learning/Tutoring:**
   Adapt learning material to student needs with AI-powered tutoring platforms.
8. **Medical Imaging and Diagnostics:**
   Assist in generating and analyzing scans for faster, AI-supported diagnoses.
9. **Game Development:**
   Auto-generate levels, characters, environments, and dialogues in video games.
10. **Legal and Financial Document Drafting:**
    Generate contracts, reports, and financial summaries with greater speed and accuracy.

DIFFERENCE BETWEEN GENERATIVE AI AND AGENTIC AI

Feature	Generative AI	Agentic AI
Definition	AI that creates new content (text, images, audio, code, etc.) from learned data.	AI that not only generates content but also sets goals, plans, makes decisions, and acts.
Core Function	Focuses on content creation using patterns in data.	Focuses on autonomous decision-making and task execution.
Example Technologies	GPT-4, DALL·E, Stable Diffusion.	OpenAI Agents, Auto-GPT, BabyAGI.
Autonomy Level	Low — requires human prompt/input to generate output.	High — can operate independently once goals are defined.

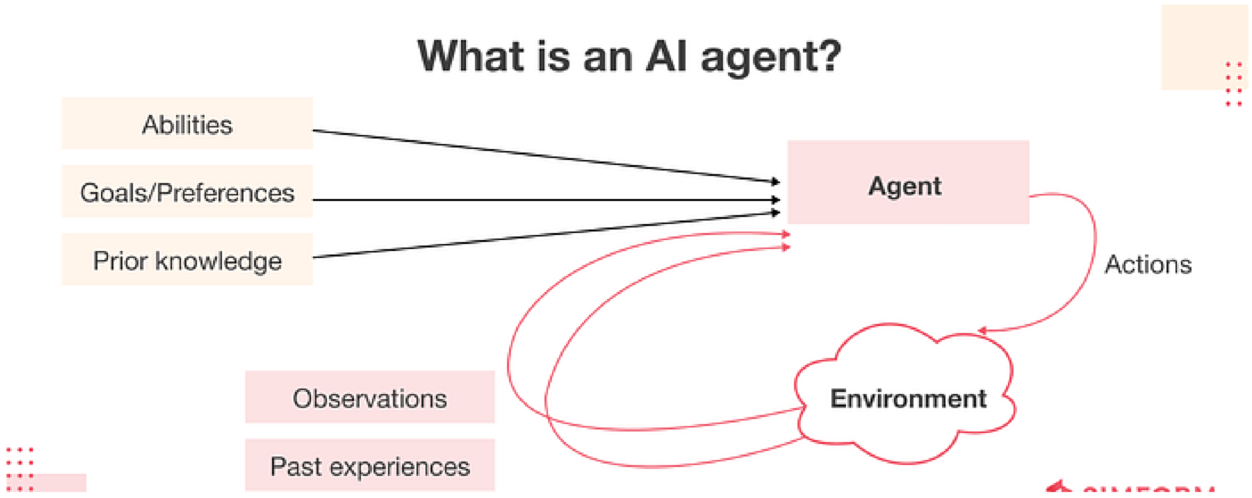
Feature	Generative AI	Agentic AI
Memory & Planning	Lacks persistent memory or long-term planning.	Has memory, can reason, plan, adapt, and learn from past actions.
Use Cases	Content creation, translation, summarization, image generation.	Task automation, workflow orchestration, autonomous agents, assistants.
Tool Usage	Cannot access or operate external tools or APIs by default.	Can use tools, call APIs, access data, and complete multi-step tasks.
Feedback Handling	Minimal — doesn't adapt based on user feedback over time.	Adapts and learns from feedback and results to improve performance.

SUMMARY:

- **Generative AI**. is creative and powerful at generating content.
- **Agentic AI** is proactive, goal-driven, and capable of acting autonomously in complex environments.

4. OpenAI’s Agents SDK – What it is, why it is used, Benefits.

OPENAI’S AGENTS SDK



The OpenAI Agents SDK is a Python-based package that lets you create AI applications with minimal fuss. It's an upgrade from OpenAI's earlier "Swarm" project, now production-ready and focused on simplicity.

With just a few building blocks (called *primitives*), you can build sophisticated AI workflows. These primitives are:

- **Agents**: Smart AI units that can think and act.
- **Handoffs**: A way for agents to pass tasks to each other.
- **Guardrails**: Safety checks to keep everything on track.

WHY USE THE AGENTS SDK?

The SDK is built with two main goals:

1. **Simple but Powerful**: It has just enough features to be useful, but not so many that it's hard to learn.
2. **Flexible**: It works well right away, but you can tweak it to fit your needs.

Here are its standout features:

- **Agent Loop**: Automatically manages how agents think, use tools, and finish tasks.

- **Python-First**: Uses Python’s natural features (like functions and loops) instead of forcing you into new frameworks.
- **Handoffs**: Lets agents delegate tasks to specialists.
- **Guardrails**: Checks inputs or outputs to catch problems early.
- **Function Tools**: Turns any Python function into something an agent can use.
- **Tracing**: Shows you a step-by-step view of what your agents are doing.

CORE CONCEPTS

1. AGENTS

What They Are: Agents are the heart of the SDK.

How They Work: An agent takes your input, thinks about it using its LLM brain, and produces an output. For example, a “Math Tutor” agent could solve math problems and explain the steps.

Example:

```
from agents import Agent

agent = Agent(
    name="Math Tutor",
    instructions="You provide help with math problems. Explain your reasoning step-by-step."
)
```

Here, the agent knows its job is to help with math and explain things clearly.

Adding Tools: Agents can use tools to do more than just talk. A tool is a Python function the agent can call. For instance:

```
from agents import function tool

@function tool
def add numbers(a: int, b: int) -> int:
    return a + b

agent = Agent(
    name="Math Tutor",
    instructions="Use tools to solve math problems.",
    tools=[add_numbers]
)
```

Now, if you ask, “What’s 5 + 3?”, the agent can call add_numbers(5, 3) and reply, “8.”

2. HANDOFFS

What They Are: Handoffs let one agent pass a task to another agent better suited for it. It’s like a receptionist handing you off to a specialist at a help desk.

How They Work: You list “handoff agents” when creating an agent. The main agent decides when to delegate based on its instructions. For example, a “Triage Agent” might decide if a question is about history or math and pass it to the right expert.

Example:

```
from agents import Agent

history_tutor = Agent(
    name="History Tutor",
    instructions="Answer historical questions clearly."
)

math_tutor = Agent(
    name="Math Tutor",
    instructions="Solve math problems with explanations."
)

triage_agent = Agent(
    name="Triage Agent",
    instructions="Pass history questions to History Tutor and math questions to Math Tutor.",
    handoffs=[history_tutor, math_tutor]
)
```

If you ask the triage agent, “Who was the first U.S. president?”, it hands off to the History Tutor, who answers, “George Washington.”

Key Points:

- Handoffs make teamwork possible between agents.
- The LLM decides when to hand off based on the input and instructions.
- It’s great for splitting big tasks into smaller, specialized jobs.

3. GUARDRAILS

What They Are: Guardrails are safety nets. They check inputs (what you ask) or outputs (what the agent says) to make sure everything's okay — like ensuring an agent doesn't waste time on off-topic questions.

How They Work: You define a guardrail with a function or another agent that runs alongside the main agent. If the check fails (a "tripwire" is triggered), the process stops, saving time and avoiding bad results.

Example: Imagine you only want an agent to handle homework questions:

```
from agents import Agent, InputGuardrail, GuardrailFunctionOutput
from pydantic import BaseModel

class HomeworkCheck(BaseModel):
    is_homework: bool
    reasoning: str

guardrail_agent = Agent(
    name="Guardrail Check",
    instructions="Check if the question is about homework.",
    output_type=HomeworkCheck
)

async def homework_guardrail(ctx, agent, input_data):
    result = await Runner.run(guardrail_agent, input_data)
    check = result.final_output_as(HomeworkCheck)
    return GuardrailFunctionOutput(
        output_info=check,
        tripwire_triggered=not check.is_homework  # Stops if not homework
    )

main_agent = Agent(
    name="Homework Helper",
    instructions="Help with homework questions.",
    input_guardrails=[InputGuardrail(guardrail_function=homework_guardrail)]
)
```

If you ask, “What’s the weather like?”, the guardrail stops it because it’s not homework-related.

Key Points:

- **Input Guardrails**: Check what you ask before the agent starts.
- **Output Guardrails**: Check the agent’s answer before it’s returned.
- They save resources and keep agents focused.

4. RUNNING AGENTS

What It Means: To get an agent to work, you “run” it using the Runner class. This starts the agent’s thinking process.

How It Works: The SDK has an *agent loop*:

1. The agent gets your input.
2. The LLM thinks and either:

- Gives a final answer (loop ends).
- Calls a tool (runs it, feeds the result back, loops again).

- Hands off to another agent (loop continues with the new agent).

3.It stops when done or hits a limit (e.g., too many steps).

Example:

```
from agents import Agent, Runner

agent = Agent(name="Assistant", instructions="You are helpful.")
result = Runner.run_sync(agent, "What’s 2 + 2?")
print(result.final_output)  # Outputs: "4"
```

Options:

- Runner.run_sync(): Runs and waits for the answer (simple).
- Runner.run(): Async version for advanced use.
- Runner.run_streamed(): Shows updates as the agent works (like live typing).

Key Points:

- The loop handles tools and handoffs automatically.
- You control how to run it based on your needs.

5. TOOLS

What They Are: Tools are functions an agent can use to act — like a calculator or a web search. They make agents more than just chatbots.

Types:

- **Function Tools**: Any Python function you write.
- **Hosted Tools**: Built-in options like web search (from OpenAI).
- **Agents as Tools**: One agent can call another as a tool without handing off.

Example:

```
from agents import function_tool

@function_tool
def get_time() -> str:
   return "It’s 3:00 PM."

agent = Agent(
   name="Time Teller",
   instructions="Tell the time when asked.",
   tools=[get_time]
)
```

Ask, “What time is it?” and the agent uses get_time() to reply, “It’s 3:00 PM.”

Key Points:

- Tools are easy to create with @function_tool.
- They let agents interact with the world (e.g., fetch data, compute).

6. TRACING

What It Is: Tracing is like a logbook. It records everything an agent does — thinking, tool calls, handoffs — so you can see and fix problems.

How It Works: Every run creates a "trace" you can view in the OpenAI Dashboard. It's on by default but can be turned off.

Example: When you run an agent, tracing shows:

- What the LLM said.
- Which tools it used.
- Any handoffs that happened.

Key Points:

- Helps you debug and improve your app.
- Viewable in a dashboard for easy analysis.

7. CONTEXT MANAGEMENT

What It Is: Context is the info an agent uses to do its job. There are two kinds:

- **Local Context**: Data your code can access (e.g., user ID).
- **LLM Context**: What the agent “sees” (conversation history).

Example (Local Context):

```
from dataclasses import dataclass

@dataclass
class User:
   name: str

agent = Agent(name="Greeter", instructions="Greet the user by name.")
user = User(name="Alice")
result = Runner.run_sync(agent, "Hi!", context=user)
# Uses user.name in the response: "Hello, Alice!"
```

Key Points:

- Local context helps tools and code.
- LLM context shapes what the agent knows.

8. ORCHESTRATING MULTIPLE AGENTS

What It Means: Orchestration is how agents work together. You can:

- **Let the LLM Decide**: Use handoffs for the agent to pick the next step.
- **Control with Code**: Write Python to chain agents (e.g., Agent A's output feeds Agent B).

Example (Code Control):

```
researcher = Agent(name="Researcher", instructions="Find facts.")
writer = Agent(name="Writer", instructions="Write a summary.")
result1 = Runner.run_sync(researcher, "Facts about cats.")
result2 = Runner.run_sync(writer, result1.final_output)
print(result2.final_output)  # A cat summary
```

Key Points:

- LLM-driven is flexible; code-driven is predictable.
- Mix both for best results.

9. MODELS

What They Are: Models are the LLM brains behind agents (e.g., OpenAI's GPT models). You can choose different ones for speed or power.

Example:

```
agent = Agent(
  name="Fast Agent",
  instructions="Answer quickly.",
    model="o3-mini"  # A fast model
)
```

Key Points:

- Pick models based on task needs (cheap vs. smart).
- Supports OpenAI models; others need tweaks.

PUTTING IT ALL TOGETHER

```
from agents import Agent, Runner, InputGuardrail, GuardrailFunctionOutput
from pydantic import BaseModel
import asyncio

# Guardrail to check for homework
class HomeworkCheck(BaseModel):
    is_homework: bool

guardrail_agent = Agent(name="Check", instructions="Is this homework?", output_type=HomeworkCheck)

async def homework_guardrail(ctx, agent, input_data):
    result = await Runner.run(guardrail_agent, input_data)
    check = result.final_output_as(HomeworkCheck)
    return GuardrailFunctionOutput(output_info=check, tripwire_triggered=not check.is_homework)

# Specialist agents
math_agent = Agent(name="Math Tutor", instructions="Solve math problems.")
history_agent = Agent(name="History Tutor", instructions="Answer history questions.")

# Triage agent with handoffs and guardrail
triage_agent = Agent(
    name="Triage",
    instructions="Route to Math Tutor or History Tutor.",
    handoffs=[math_agent, history_agent],
    input_guardrails=[InputGuardrail(guardrail_function=homework_guardrail)]
)

# Run it
async def main():
    result = await Runner.run(triage_agent, "What’s 5 + 5?")
    print(result.final_output)  # Math Tutor answers: "10"

asyncio.run(main())
```

This setup checks if the question is homework, then routes it to the right agent.