1. What is Generative AI, and how does it differ from traditional AI?

Definition of Generative AI

Generative AI is a class of artificial intelligence that generates new data, content, or patterns based on training data. It can produce human-like text, images, audio, and even videos.

How It Differs from Traditional AI

Feature	Traditional AI (Discriminative)	Generative AI
Goal	Recognizes and classifies patterns	Creates new patterns based on learned data
Example Task	Spam detection, fraud detection, sentiment analysis	Text generation, image synthesis, music composition
Model Type	Discriminative models like Logistic Regression, Random Forest, CNNs	Generative models like GANs, VAEs, Transformers (GPT, DALL·E)
Training Data	Needs labeled data for supervised learning	Can work with unlabeled data and learn distribution

Example

- **Traditional AI**: A model that classifies an email as spam or not spam.
- **Generative AI**: A model that generates an entirely new email mimicking human writing style.

---

2. Key Milestones in the Evolution of AI Models

The evolution of AI has seen several major breakthroughs:

Year	Milestone	Description
1950s	Turing Test	Alan Turing proposed a test to measure machine intelligence.
1957	Perceptron	The first neural network was introduced by Frank Rosenblatt.
1980s	Backpropagation	The backpropagation algorithm improved neural network training.
1997	Deep Blue	IBM’s Deep Blue defeated world chess champion Garry Kasparov.
2012	AlexNet	Convolutional Neural Networks (CNNs) revolutionized image recognition.

Year	Milestone	Description
2014	GANs	Generative Adversarial Networks (GANs) were introduced by Ian Goodfellow.
2017	Transformers	Google introduced the Transformer model, which led to breakthroughs in NLP.
2020+	GPT-3, DALL·E, CLIP	Large-scale generative AI models emerged for text, images, and multimodal applications.

---

3. Major Applications of Generative AI Across Industries

Generative AI is transforming multiple industries:

Industry	Application
Healthcare	Drug discovery, medical image generation, personalized treatment plans
Finance	Synthetic data generation for fraud detection, algorithmic trading
Entertainment	AI-generated music, deepfake videos, game character design
E-commerce	AI-powered product descriptions, virtual shopping assistants
Education	AI-generated tutoring content, personalized learning

Example: Generative AI for Text Generation

```
from transformers import pipeline

# Load GPT-3.5 or similar model
generator = pipeline("text-generation", model="gpt2")

# Generate text
text = generator("Once upon a time, in a futuristic world,", max_length=50)
print(text[0]["generated_text"])
```

---

4. Difference Between Generative and Discriminative Models

Feature	Generative Models	Discriminative Models
Function	Learns data distribution and generates new data	Learns decision boundaries to classify data

Feature	Generative Models	Discriminative Models
Example Models	GANs, VAEs, Transformers	Logistic Regression, SVM, Random Forest
Use Cases	Image synthesis, text generation	Spam detection, fraud detection

Example

- **Generative Model**: A GAN that generates realistic human faces.
- **Discriminative Model**: A CNN that classifies images as "cat" or "dog."

Code Example: Generative vs Discriminative

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic data
X, y = make_classification(n_samples=1000, n_features=20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Discriminative model
clf = LogisticRegression()
clf.fit(X_train, y_train)
print(f"Accuracy: {clf.score(X_test, y_test)}")
```

---

5. Transformer Architecture and Its Advantage Over RNNs and CNNs

Limitations of RNNs and CNNs

- **RNNs (Recurrent Neural Networks)**
  - Sequential processing makes them slow.
  - Hard to handle long-range dependencies.
- **CNNs (Convolutional Neural Networks)**
  - Excel in image tasks but struggle with sequential data.

Transformer Architecture

- **Self-Attention Mechanism**: Helps the model focus on important words.
- **Parallel Processing**: Unlike RNNs, transformers process input in parallel.
- **Positional Encoding**: Captures word order information.

Example: Self-Attention in Transformer

```
import torch
import torch.nn.functional as F

# Example sentence embeddings
query = torch.rand(1, 10)  # Query vector
key = torch.rand(1, 10)    # Key vector
value = torch.rand(1, 10)  # Value vector

# Compute attention scores
attention_scores = F.softmax(torch.matmul(query, key.T) / 
torch.sqrt(torch.tensor(10.0)), dim=-1)
attention_output = torch.matmul(attention_scores, value)

print("Self-Attention Output:", attention_output)
```

---

6. Self-Attention Mechanism and Its Importance

How Self-Attention Works

- Assigns different weights to words in a sentence.
- Helps understand relationships between words even when far apart.

Example

Sentence: *"The cat sat on the mat."*

- "cat" and "mat" are related, so self-attention assigns them higher weight.

---

7. Computational Complexity of Self-Attention

Self-attention has **$O(n^2)$** complexity due to pairwise attention computation.

Optimizations

- **Sparse Attention**: Reduces number of attention computations.
- **Linformer**: Reduces complexity to **$O(n)$**.
- **Longformer**: Uses local attention windows.

---

8. Challenges in Training Large-Scale Generative Models

Challenges

1. **High Computational Costs**: Training models like GPT-4 requires massive GPUs.
2. **Data Bias**: AI models inherit biases from training data.
3. **Hallucinations**: Generative models can generate false information.
4. **Memory and Storage**: Storing large models is expensive.
5. **Energy Consumption**: Training LLMs requires substantial electricity.

Possible Solutions

- **Efficient Fine-tuning**: Parameter-efficient fine-tuning (LoRA, QLoRA).
- **Distillation**: Compress large models into smaller ones.
- **Federated Learning**: Train models without centralizing data.

OpenAI APIs

9. What are the primary use cases for OpenAI's GPT-4, DALL·E, and Whisper APIs?

OpenAI provides APIs for different generative tasks:

GPT-4 (Text Generation)

- **Use Cases**:
  - Chatbots (Customer support, virtual assistants)
  - Content generation (Articles, blogs, ads)
  - Code generation (AI-assisted coding)
  - Data analysis (Summarization, trend detection)

  Example: Using GPT-4 for text generation

```
import openai

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[{"role": "user", "content": "Explain quantum computing in 
simple terms."}]
)
print(response['choices'][0]['message']['content'])
```

DALL·E (Image Generation)

- **Use Cases**:
  - Generating unique artwork and illustrations

- Product design and concept visualization
- Marketing and branding visuals
- Text-to-image synthesis for creative projects

Example: Generating an image using DALL·E

```
response = openai.Image.create(
    model="dall-e-2",
    prompt="A futuristic city skyline at sunset",
    n=1,
    size="1024x1024"
)
print(response['data'][0]['url'])  # Returns the image URL
```

Whisper (Speech Recognition)

- **Use Cases**:
    - Automated transcription of meetings, podcasts, and lectures
    - Real-time captioning for videos
    - Multilingual speech-to-text conversion
    - Voice-enabled chatbots and assistants

  Example: Using Whisper for speech-to-text

```
import openai

audio_file = open("audio.mp3", "rb")
transcript = openai.Audio.transcribe("whisper-1", audio_file)
print(transcript["text"])
```

---

10. How does GPT-4 handle prompt engineering, and what strategies improve response quality?

Prompt engineering is the process of designing inputs to maximize a language model's effectiveness.

Best Practices:

1. **Be Clear and Specific**
    - Bad: "Tell me about AI."
    - Good: "Explain how transformers improve NLP compared to RNNs."
2. **Use Context and Roles**

```
prompt = "You are an AI tutor. Explain deep learning in simple terms."
```

3. **Format Inputs Clearly**

```
prompt = "Summarize the following text in 3 bullet points
```

4. **Provide Examples**

```
prompt = "Translate 'I love programming' into Spanish and French."
```

---

11. Differences between GPT-3.5 and GPT-4 in terms of performance and capabilities

Feature	GPT-3.5	GPT-4
Reasoning Ability	Moderate	Stronger logical reasoning
Context Length	~4,096 tokens	~32,768 tokens
Creativity	Good	More nuanced and accurate
Code Generation	Decent	Significantly better for complex tasks

---

12. How does DALL·E generate images from text, and what are its limitations?

DALL·E uses **Diffusion Models**, which gradually refine noise into a meaningful image based on text prompts.

Limitations:

- **Inconsistent Text Rendering**: Struggles with generating legible text in images.
- **Biases in Data**: Trained on public datasets, leading to biases.
- **High Computational Cost**: Image generation is resource-intensive.

Google Vertex AI

13. What is Google Vertex AI, and how does it support Generative AI workloads?

Google Vertex AI is a managed ML platform offering:

- **Pre-trained LLMs** (PaLM, Imagen)
- **AutoML for custom models**
- **MLOps tools for deployment**

14. Google Vertex AI vs. OpenAI API

Feature	Google Vertex AI	OpenAI API
Customization	Full model fine-tuning	Limited fine-tuning
Integration	GCP ecosystem	Standalone API
Model Availability	PaLM, Imagen	GPT, DALL·E, Whisper

15. Advantages of Google Vertex AI for Custom Training

- **Hyperparameter Tuning**
- **Scalable Compute on GCP**
- **End-to-End MLOps Support**

Anthropic Claude API

16. What is Claude, and how does it differ from OpenAI's GPT models?

- **Claude** (by Anthropic) focuses on safety and interpretability.
- **Less aggressive fine-tuning** compared to OpenAI.

17. Safety Features in Claude

- **Contextual Safety Filters**
- **Reduced Bias and Hallucination**
- **Strict Ethical AI Guardrails**

Cohere API

18. What differentiates Cohere in the LLM space?

- **Focus on Enterprise NLP**
- **Multilingual Models**
- **Powerful Embeddings API**

19. Cohere Embeddings for NLP

- Used for **search**, **recommendation engines**, and **semantic analysis**.

```
import cohere

co = cohere.Client("API_KEY")
response = co.embed(["This is an example sentence."])
print(response.embeddings)
```

Hugging Face Inference API

20. What is it, and how does it enable model access?

Hugging Face **Inference API** allows developers to:

- **Use pre-trained models** via API calls.
- **Deploy custom models easily.**

21. How does it compare to OpenAI and Google APIs?

Feature	Hugging Face	OpenAI	Google Vertex AI
Open-source	✅	❌	❌
Fine-tuning	✅	Limited	✅
API-based Inference	✅	✅	✅

Meta's Llama API

22. What is LLaMA, and what are its strengths/limitations?

- **Strengths**: Open-source, efficient, strong multilingual support.
- **Limitations**: Requires local deployment, lacks APIs like OpenAI.

23. LLaMA 2 vs. GPT-4

Feature	LLaMA 2	GPT-4
Open-source	✅	❌
API Available	❌	✅

Mistral AI API

24. What makes Mistral AI unique?

- **Smaller, optimized models**
- **Focus on enterprise AI efficiency**

25. Enterprise Optimizations in Mistral

- **Sparse Mixture of Experts (SMoE)**
- **Lower latency and power consumption**

Azure OpenAI Service

26. What is it, and how does it integrate OpenAI models with enterprises?

Azure **provides OpenAI models** with enterprise-grade security and compliance.

27. Benefits of Azure OpenAI Service

- **Scalability on Azure Cloud**
- **Integration with Microsoft Products (Power BI, Office 365)**

Amazon Bedrock

28. Multi-model AI deployment on Amazon Bedrock

Supports **multiple foundation models** from **Anthropic, AI21 Labs, Stability AI**.

29. Amazon Bedrock vs. Traditional Cloud AI APIs

Feature	Amazon Bedrock	Traditional APIs
Multi-model Support	✅	❌
Customization	✅	Limited

Google AI Studio & IBM Watson AI

30. Google AI Studio vs. IBM Watson

Feature	Google AI Studio	IBM Watson AI
Ease of Use	Developer-friendly	Enterprise-grade tools
Model Focus	Text, image generation	Business AI solutions