

Generative AI

Interview Q & A


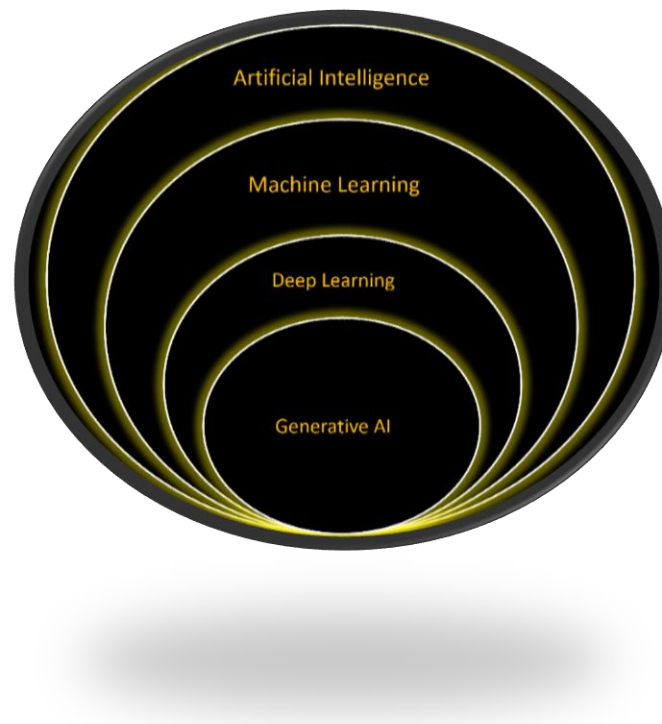
 GenAI Roadmap



Table of Contents

Generic Questions	3
Generative AI	9
Discriminative AI	11
General Questions on Generative and Discriminative AI	12
Transformer Architecture	12
Large Language Models (LLMs)	17
Embedding Models	21
Retrieval-Augmented Generation (RAG)	22
LLM Fine Tuning	24
Misc Topics in Generative AI	30
Prompt Engineering	31
One-Shot Prompting	36
Few-Shot Prompting	37
Zero-Shot Prompting	38
Chain of Thought Prompting	38
Hybrid Prompting	39
ReAct Prompting	41
Advance Topics	41
Graph Retrieval Augmented Generation (GraphRAG)	41
Agentic AI / AI Agents	47
Latest Q & A (Based on question asked in different companies)	50
Latest top LLM Leaderboard	58
LMSYS Leaderboard	58
Modern LLM Topics (2024-2026)	58
Research Papers	63
2025	63
2024	63
2023	64
2022	65
2020	65
2017	65



Generic Questions

Useful Resources

1. [GenAI Essential Terms](#)
2. [GenAI with Azure Cloud](#)
3. [GenAI with AWS Cloud](#)
4. [GenAI with VertexAI](#)
5. [Agentic AI Interview Q&A](#)

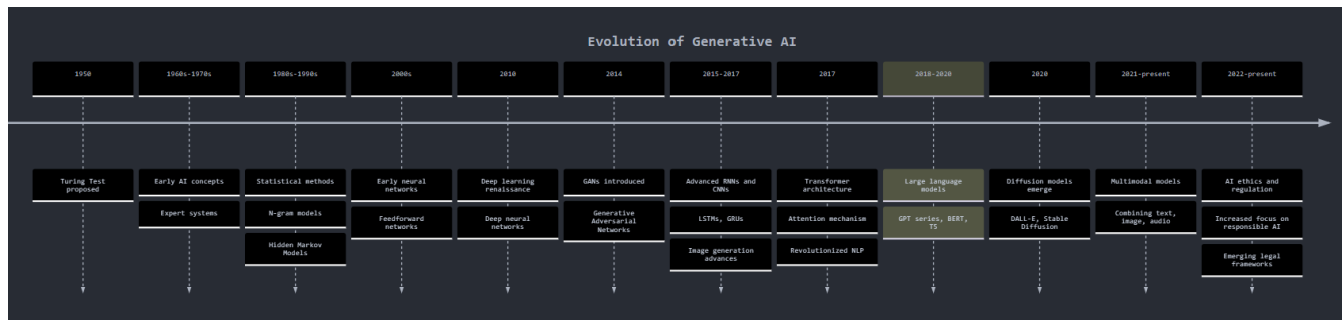
Q: What is Generative AI?

A: Generative AI refers to a class of artificial intelligence models that can generate new data or content based on the data they were trained on. These models can create text, images, audio, and more.

Here is an overview of its evolution:

- Early developments (1950s-2000s): Basic text generation, rule-based systems
- Rise of neural networks (2010s): Deep learning advances, like RNNs and LSTMs
- Transformer architecture (2017): Revolutionized natural language processing
- GPT type models (2018-present): Increasingly powerful language models including GPT, Claude, Gemini, Llama3 etc
- Diffusion models (2020-present): Advanced image generation capabilities

- Multimodal models (2021-present): Combining text, image, and other modalities



Q: How does Generative AI relate to deep learning, machine learning, and AI?

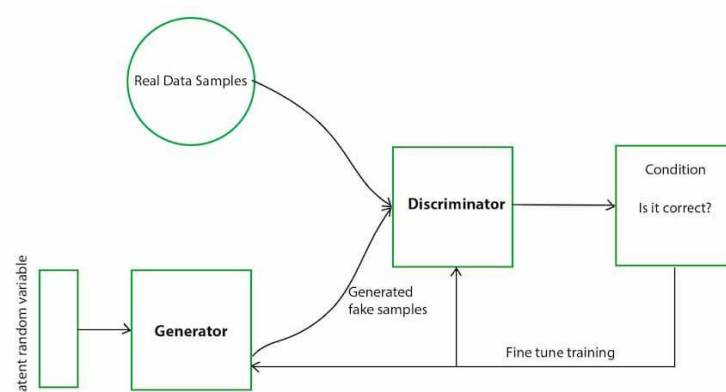
A: Generative AI is a specialized branch of artificial intelligence focused on creating new content, like text, images, and audio. It exists within a hierarchy of AI concepts:

- **Artificial Intelligence (AI):** The broad field aiming to develop systems that can perform tasks requiring human-like intelligence.
- **Machine Learning (ML):** A subset of AI that uses data-driven algorithms to enable systems to learn and improve over time without being explicitly programmed.
- **Deep Learning (DL):** A subset of ML that uses neural networks with multiple layers to analyse complex patterns in large datasets.
- **Generative AI:** A cutting-edge application of DL that creates new and original content by learning patterns from existing data.

Generative AI leverages deep learning techniques to extract features and recognize patterns, building on machine learning's data-driven foundation to produce creative outputs. It's a leading example of how AI can mimic and extend human cognitive abilities.

Q: How does a Generative Adversarial Network (GAN) work?

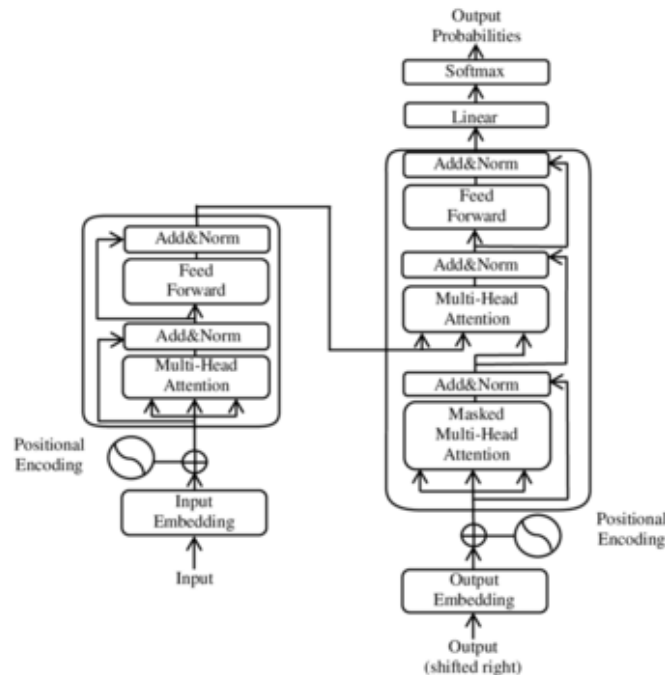
A: A GAN consists of two neural networks: a generator and a discriminator. The generator creates fake data, while the discriminator tries to distinguish between real and fake data. Both networks are trained simultaneously in a game-like scenario where the generator improves its ability to create realistic data, and the discriminator gets better at detecting fakes.



GAN (Source - GeeksForGeeks)

Q: What is a Transformer model?

A: A Transformer model is a type of deep learning architecture introduced in the paper "Attention is All You Need." It uses self-attention mechanisms to process input data and is highly effective for tasks like language translation and text generation.



Transformer Architecture (Source – Wikipedia)

Q: What is the difference between supervised and unsupervised learning in the context of Generative AI?

A: In supervised learning, models are trained on labeled data where each training example has an associated output label. In unsupervised learning, models learn patterns from unlabeled data. Importantly, most modern generative AI models - particularly LLMs - use self-supervised learning, where labels are automatically derived from the input data itself (e.g., predicting the next token in a sequence). Self-supervised learning is distinct from both traditional supervised learning (externally labeled data) and unsupervised learning (no labels at all). It enables training on massive unlabeled text corpora without manual annotation.

Q: Can you explain the concept of 'latent space' in Generative AI?

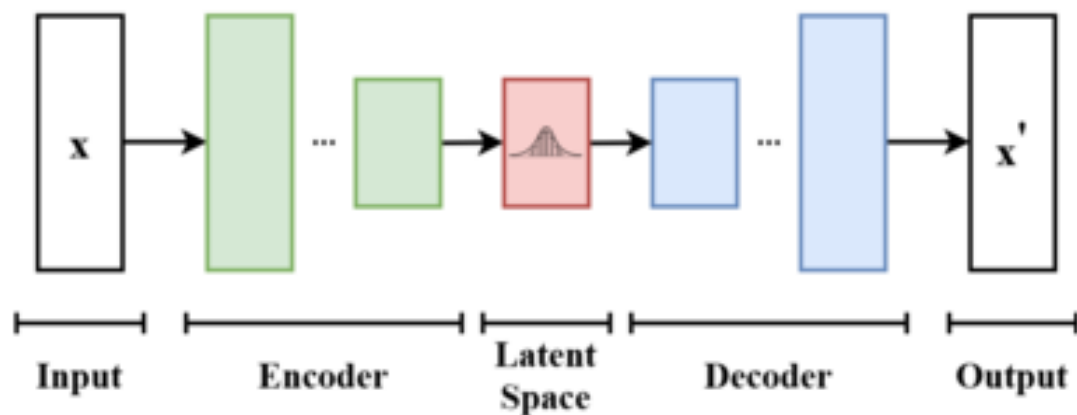
A: Latent space refers to a lower-dimensional representation of data in which generative models, like GANs or VAEs, encode high-dimensional input data. Navigating this space allows the model to generate new, similar data by decoding points in the latent space.

Q: What is the purpose of the self-attention mechanism in the Transformer model?

A: The self-attention mechanism allows the Transformer model to weigh the importance of different words in a sentence relative to each other. This helps the model understand context and relationships within the data, leading to more accurate predictions and generation.

Q: What is a Variational Autoencoder (VAE)?

A: A VAE is a type of generative model that learns to encode input data into a probabilistic latent space and then decodes it back to the original data space. Unlike traditional autoencoders, VAEs introduce a regularization term that forces the latent space to follow a known distribution, typically Gaussian.



VAE (Source - Wikipedia)

Q: How does the Encoder-Decoder architecture work?

A: The Encoder-Decoder architecture consists of two parts: the encoder, which processes the input sequence and produces contextual representations, and the decoder, which generates the output sequence. In the original seq2seq architecture (pre-attention), the encoder compressed output into a fixed-size context vector - a bottleneck that limited performance on long sequences. Modern Transformer-based encoder-decoders (T5, BART) overcome this via cross-attention: the decoder attends over all encoder hidden states at each decoding step, not a single compressed vector. This architecture is used in machine translation, summarization, and question answering.

Q: What are some common applications of Generative AI?

A: Common applications include text generation (e.g., chatbots, content creation), image generation (e.g., deepfakes, art creation), music composition, data augmentation, drug discovery, and improving the quality of medical imaging.

Q: How does a Recurrent Neural Network (RNN) differ from a Transformer in Generative AI tasks?

A: RNNs process sequential data by maintaining a hidden state that captures information from previous steps. Transformers, on the other hand, use self-attention mechanisms to process entire sequences in parallel, which makes them more efficient and better at capturing long-range dependencies.

Q: What is BERT, and how is it used in Generative AI?

A: BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model designed for understanding context by processing text bidirectionally (left and right simultaneously). While BERT itself is not generative, it excels at understanding tasks like question answering, classification, and NER. Important: BERT and GPT are separate model families, both derived from the Transformer architecture - BERT uses only the encoder stack (bidirectional understanding), while GPT uses only the decoder stack (autoregressive generation). They are not adaptations of each other. BERT is an Encoder-only Model.

Q: How does GPT differ from BERT?

A: GPT (Generative Pre-trained Transformer) is designed primarily for text generation, using a unidirectional (left-to-right) approach. BERT, on the other hand, is designed for understanding and processing text by using a bidirectional approach. GPT generates coherent text, while BERT excels in tasks like question answering and text classification.

GPT is Decoder only Model and BERT is Encoder only Model

Q: What is style transfer in the context of Generative AI?

A: Style transfer is a technique in Generative AI that applies the style of one image (e.g., a painting) to the content of another image (e.g., a photograph). This is achieved by separating and recombining the content and style representations in the images using neural networks.

Q: How can Generative AI be used in data augmentation?

A: Generative AI can create synthetic data that resembles real data, which can be used to augment training datasets. This is especially useful in scenarios where collecting real data is expensive or impractical, helping to improve the performance of machine learning models by increasing data diversity.

Q: What is transfer learning, and how is it applied in Generative AI?

A: Transfer learning involves using a pre-trained model on a new, related task. In Generative AI, models like GPT-3 are pre-trained on large corpora of text and can be fine-tuned for specific tasks (e.g., text summarization, translation) with relatively smaller datasets.

Q: What is a Deepfake?

A: A Deepfake is a synthetic media, often a video or audio, created using Generative AI techniques, particularly GANs. It involves altering existing media or creating new media to make it appear authentic, often used to create realistic but fake representations of people.

Q: How does an autoencoder work, and what is its purpose in Generative AI?

A: An autoencoder consists of an encoder that compresses the input data into a lower-dimensional latent space and a decoder that reconstructs the original data from this latent space. It is used for tasks like dimensionality reduction, anomaly detection, and as a building block in generative models.

Q: What is the role of loss functions in training GANs?

A: Loss functions in GANs guide the training process of both the generator and the discriminator. The generator's loss measures how well it can fool the discriminator, while the discriminator's loss measures how well it can distinguish real data from fake data. Balancing these losses is crucial for stable training.

Q: What is the importance of the "attention mechanism" in NLP tasks?

A: The attention mechanism allows models to focus on relevant parts of the input sequence when making predictions. This is particularly important in NLP tasks for capturing contextual dependencies and improving the quality of generated text.

Q: What are some ethical concerns associated with Generative AI?

A: Ethical concerns include the potential for misuse in creating fake content (deepfakes), intellectual property issues, bias in generated content, and the impact on privacy. Ensuring responsible use and developing techniques for detecting AI-generated content are ongoing challenges.

Q: How can Generative AI improve natural language understanding?

A: Generative AI models like GPT-4, Gemini, Claude, Llama etc can generate coherent and contextually relevant text, which helps in building more advanced chatbots, improving machine translation, and creating better tools for text summarization and sentiment analysis.

Q: What is reinforcement learning, and is it used in Generative AI?

A: Reinforcement learning involves training an agent to make decisions by rewarding desired actions and penalizing undesired ones. Reinforcement learning is central to

modern generative AI, primarily through RLHF (Reinforcement Learning from Human Feedback) - the standard technique used to align LLMs with human preferences after pre-training. RLHF was popularized by InstructGPT (2022) and underpins ChatGPT, Claude, Gemini, Llama 3-Instruct, and virtually all major deployed LLMs today. More recently, DPO (Direct Preference Optimization) has emerged as a simpler and computationally cheaper alternative to PPO-based RLHF. RL is also used in reasoning-focused models like DeepSeek-R1, trained primarily via reinforcement learning to develop strong reasoning capabilities.

Q: What is the significance of "zero-shot learning" in Generative AI?

A: In the LLM context, zero-shot learning (zero-shot prompting) refers to a model performing a task given only a natural language instruction - with no in-context examples - relying entirely on pre-trained knowledge. This is significant because LLMs can generalize to unseen tasks at inference time without labeled examples. Note: The classical ML definition of zero-shot learning (ZSL) refers to predicting classes unseen during training using semantic/attribute vectors transferred from seen classes - a related but distinct concept. In modern GenAI interviews, zero-shot almost always refers to zero-shot prompting.

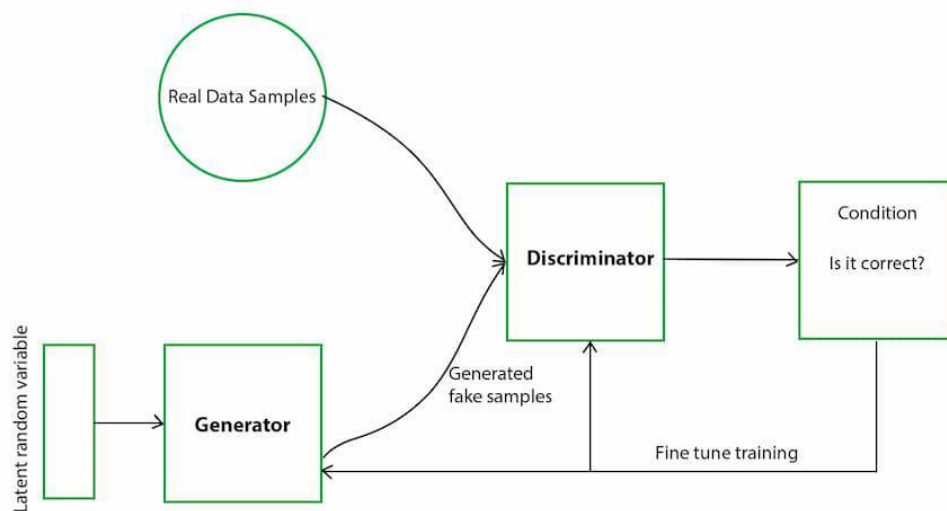
Generative AI

Q: What is Generative AI and how does it differ from other types of AI?

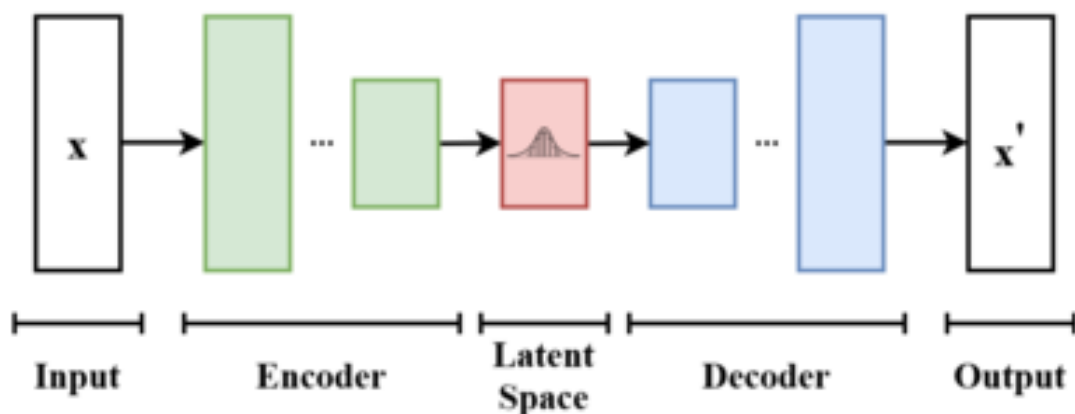
A: Generative AI refers to models that generate new data samples similar to the training data. Unlike discriminative models, which classify input data into categories, generative models can create new instances that resemble the input data. Examples include text generation, image synthesis, and music composition.

Q: Can you explain the difference between a Generative Adversarial Network (GAN) and a Variational Autoencoder (VAE)?

A: GANs consist of two networks, a generator and a discriminator, that compete against each other to produce realistic data. The generator creates data samples, while the discriminator evaluates their authenticity. VAEs, on the other hand, encode data into a latent space and decode it back to reconstruct the data, focusing on generating data by learning the underlying distribution.



GAN (Source GeeksForGeeks)



VAE (Source - Wikipedia)

Q: What are some common applications of Generative AI?

A: Common applications include image and video synthesis, text generation, music and art creation, drug discovery, and data augmentation for training other machine learning models.

Q: How does the training process of a GAN work?

A: Training a GAN involves a two-step process: the generator creates fake data samples, and the discriminator evaluates them against real data samples. The generator aims to produce more realistic samples to fool the discriminator, while the discriminator aims to improve its

ability to distinguish real from fake data. This adversarial process continues until the generator produces highly realistic samples.

Q: What challenges are associated with training Generative AI models?

A: Challenges include mode collapse (where the generator produces limited variations of data), ensuring training stability, and the need for large amounts of data and computational resources.

Discriminative AI

Q: What is Discriminative AI and how does it differ from Generative AI?

A: Discriminative AI models focus on classifying input data into predefined categories. They learn the boundary between classes based on the training data. Unlike generative models, discriminative models do not generate new data; they only classify existing data.

Q: Can you give an example of a discriminative model and its application?

A: An example of a discriminative model is a Support Vector Machine (SVM), which classifies data by finding the hyperplane that best separates different classes. Applications include image classification, spam detection, and medical diagnosis.

Q: How does a discriminative model learn from data?

A: Discriminative models learn by optimizing a loss function that measures the difference between predicted and actual labels in the training data. Techniques like gradient descent are used to minimize this loss and improve the model's accuracy.

Q: What are the key differences in the training objectives of generative and discriminative models?

A: The training objective of generative models is to learn the underlying data distribution to generate new samples, while discriminative models aim to learn the decision boundary that separates different classes.

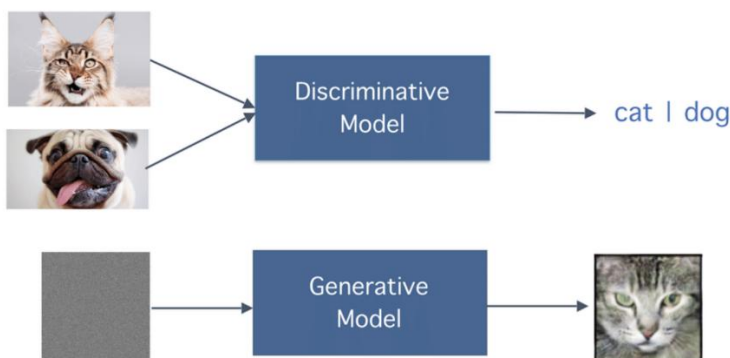
Q: What are some advantages of using discriminative models over generative models?

A: Discriminative models typically require less computational resources, have simpler training processes, and often achieve higher accuracy in classification tasks compared to generative models.

General Questions on Generative and Discriminative AI

Q: How can generative and discriminative models complement each other in a machine learning pipeline?

A: Generative models can be used for data augmentation to create additional training samples, which can improve the performance of discriminative models. Additionally, generative models can help in understanding the data distribution, which can inform the design and training of discriminative models.



https://www.researchgate.net/figure/A-simple-illustration-of-how-one-can-use-discriminative-vs-generative-models-The-former_fig1_341478640

Q: Can you discuss a scenario where both generative and discriminative models are used together?

A: In semi-supervised learning, generative models can be used to generate synthetic data to augment a small labeled dataset, and then a discriminative model can be trained on this augmented dataset to improve classification performance.

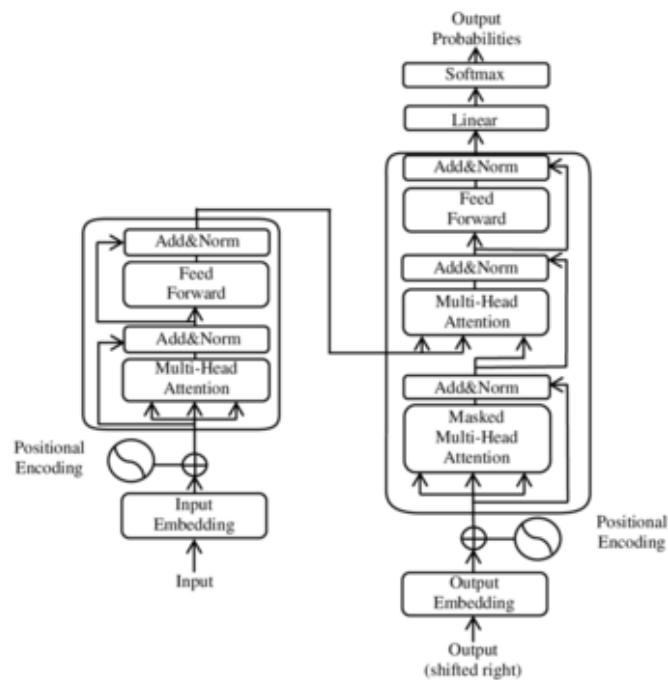
Q: What are the trade-offs between using generative and discriminative models?

A: Generative models can provide more insights into the data and generate new data, but they are often more complex and computationally intensive. Discriminative models are usually simpler, faster to train, and can achieve higher accuracy in classification tasks, but they do not generate new data.

Q: How do generative models handle unsupervised learning tasks differently from discriminative models?

A: Generative models can learn from unlabeled data by modeling the data distribution and generating new samples, while discriminative models require labeled data to learn the decision boundary between classes.

Transformer Architecture



Transformer Architecture (Source – Wikipedia)

Q: What is the Transformer architecture and why is it significant in Generative AI?

A: The Transformer architecture is a deep learning model introduced by Vaswani et al. in 2017, designed to handle sequential data with a mechanism called self-attention. It is significant in Generative AI because it efficiently processes long-range dependencies in data, enabling high-quality text generation, translation, and other language-related tasks.

Example: Imagine you're reading a sentence like "The cat sat on the mat because it was tired." To understand this sentence fully, you need to know that "it" refers to "the cat." The Transformer architecture works like this by looking at all parts of a sentence simultaneously and figuring out which words are important for understanding the context.

For example, if the Transformer sees "The cat sat on the mat because it was tired," it understands that "it" refers to "the cat" by looking at the whole sentence at once, not just word-by-word. This ability to grasp relationships between words that are far apart helps it generate better text and translate languages more accurately.

Q: Can you explain the self-attention mechanism in Transformers?

A: Self-attention allows the model to weigh the importance of different words in a sequence when generating an output. For each word, the mechanism computes attention scores relative to all other words, helping the model understand contextual relationships. This is crucial for capturing the nuances in language data.

Q: What are the main components of the Transformer architecture?

A: The main components are the encoder and decoder stacks. Each stack consists of multiple layers of self-attention and feed-forward neural networks, along with layer normalization and residual connections. The encoder processes input sequences, and the decoder generates output sequences.

Details are as follows:

- **Encoder Stack:** This processes the input data.
- **Self-Attention Mechanism:** This allows each word in the input to consider all other words in the sequence when forming its representation, helping capture the context and relationships between words.
- **Feed-Forward Neural Networks:** After self-attention, the data passes through a feed-forward network to further refine and process the information.
- **Layer Normalization:** This technique normalizes the data within each layer to improve training stability and performance.
- **Residual Connections:** These connections help in avoiding the vanishing gradient problem by allowing gradients to flow through the network more easily during training.
- **Decoder Stack:** This generates the output data based on the encoded input.
- **Masked Self-Attention:** Similar to self-attention in the encoder but designed to prevent the model from seeing future words in the sequence, which is crucial for generating text.
- **Encoder-Decoder Attention:** This layer allows the decoder to focus on different parts of the input sequence as it generates each word, helping align the generated text with the input.
- **Feed-Forward Neural Networks:** Similar to the encoder, these refine the data in the decoder.
- **Layer Normalization and Residual Connections:** Also used here to stabilize and improve the training process.

Q: How does positional encoding work in Transformers?

A: Positional encoding adds information about the position of each word in the sequence because the Transformer architecture does not inherently understand word order. This is achieved by adding sine and cosine functions of different frequencies to the input embeddings, allowing the model to incorporate positional information.

Q: What advantages do Transformers have over traditional RNNs and LSTMs?

A: Transformers address the limitations of RNNs and LSTMs, such as difficulty in capturing long-range dependencies and slow training due to sequential data processing. Transformers use self-attention mechanisms to process entire sequences in parallel, improving efficiency and performance.

Q: Can you describe the role of multi-head attention in the Transformer architecture?

A: Multi-head attention allows the model to focus on different parts of the input sequence simultaneously. By splitting the input into multiple heads, each with its own self-attention mechanism, the model can capture various aspects of the relationships between words, enhancing its ability to understand complex patterns.

Q: How does the Transformer architecture handle the task of machine translation?

A: In machine translation, the encoder processes the source language sentence, generating context-rich representations. The decoder then uses these representations, along with previously generated words, to produce the target language sentence. Attention mechanisms help align source and target sentences.

Q: What is the significance of the "Attention is All You Need" paper?

A: The paper "Attention is All You Need" by Vaswani et al. introduced the Transformer architecture and demonstrated that self-attention mechanisms could replace recurrent and convolutional layers in sequence modelling tasks. This breakthrough significantly improved performance and efficiency in NLP tasks.

Q: How are Transformers used in language models like GPT, Gemini, Claude. Llama etc?

A: Transformers serve as the backbone for large language models like GPT-3. These models use a stack of Transformer decoder layers to generate text by predicting the next word in a sequence based on the context provided by previous words.

Q: What are some challenges associated with training Transformer models?

A: Challenges include the high computational and memory requirements due to parallel processing of large sequences, managing overfitting with large model sizes, and ensuring efficient use of data to prevent long training times.

Q: How can you address the issue of long sequence processing in Transformers?

A: Techniques like sparse attention mechanisms, memory-augmented networks, and Transformer variants like Longformer and Reformer can be used to efficiently process longer sequences by reducing computational complexity and memory usage.

Q: Explain how the encoder and decoder work together in the Transformer architecture using an example.

A: In a translation task, the encoder processes the sentence "The cat is on the mat" and generates context-rich embeddings. The decoder then takes these embeddings and generates the translated sentence "Le chat est sur le tapis," using attention mechanisms to align the words correctly.

Q: What is the role of residual connections in the Transformer architecture?

A: Residual connections help mitigate the vanishing gradient problem and enable deeper networks by allowing gradients to flow more easily through the network. They also stabilize training and improve convergence rates.

Q: How do Transformers handle the challenge of data parallelism?

A: Transformers process entire sequences in parallel using self-attention mechanisms, which allows for efficient data parallelism. This significantly speeds up training and inference compared to sequential models like RNNs.

Q: Describe an application of Transformers in a non-textual domain.

A: Transformers can be applied to image processing tasks such as image classification and segmentation. Vision Transformers (ViTs) split an image into patches and process them similarly to sequences of words, leveraging self-attention to capture spatial relationships.

Q: How do layer normalization and dropout contribute to the performance of Transformers?

A: Layer normalization stabilizes the training process by normalizing the input to each sub-layer, while dropout helps prevent overfitting by randomly zeroing out a fraction of the connections during training, enhancing the model's generalization capabilities.

Q: What are some recent advancements in Transformer architectures?

A: Recent advancements in Transformer architectures include: Flash Attention - an IO-aware attention algorithm reducing memory footprint and speeding up training/inference; Grouped Query Attention (GQA) - used in Llama 2/3 and Mistral to reduce KV cache memory while maintaining quality; Mixture of Experts (MoE) - sparse activation of expert subnetworks (GPT-4, Mixtral, DeepSeek-V3) for better quality-to-compute ratios; Rotary Position Embeddings (RoPE) - replaces sinusoidal encodings in most modern LLMs for better relative position handling; Sliding Window Attention (SWA) - used in Mistral for efficient long-context processing; extended context windows reaching 200K (Claude 3) to 1M tokens (Gemini 1.5 Pro).

Q: How do Transformers manage context for very long texts?

A: Techniques like segment-level recurrence in Transformer-XL, hierarchical attention in models like BigBird, and local-global attention patterns in Longformer help manage long-context dependencies by breaking down the text into manageable chunks while retaining contextual information.

Q: How do you handle the scalability issues in training large Transformer models?

A: Scalability issues can be addressed by distributed training across multiple GPUs, mixed-precision training to reduce memory usage, model parallelism, and using efficient implementations like NVIDIA's Megatron or DeepSpeed from Microsoft.

Q: Explain the impact of Transformer models on the field of Natural Language Processing (NLP).

A: Transformer models have revolutionized NLP by setting new benchmarks in various tasks such as translation, summarization, and question answering. Their ability to handle context and dependencies more effectively than previous models has led to significant improvements in performance and opened new research avenues.

Large Language Models (LLMs)

Q: What is a Large Language Model (LLM)?

A: A Large Language Model (LLM) is a type of artificial intelligence model that uses deep learning - specifically large Transformer decoder architectures trained on massive text corpora - to understand, generate, and manipulate human language. Examples include GPT-3, GPT-4, Claude, Gemini, Llama 3, Mistral, and DeepSeek. Note: BERT is an encoder-only Transformer used for language understanding tasks (classification, NER, Q&A extraction) - it is not a generative LLM.

Q: How do LLMs like GPT-3 and GPT-4 work?

A: LLMs use transformer architectures with attention mechanisms to process input text and generate coherent, contextually relevant output. They are pre-trained on vast datasets and can be fine-tuned for specific tasks.

Q: What are the applications of LLMs?

A: Applications of LLMs include text generation, translation, summarization, question answering, sentiment analysis, and conversational agents.

Q: What is the transformer architecture?

A: The transformer architecture is a neural network design that relies on self-attention mechanisms to weigh the importance of different parts of the input data, enabling efficient parallel processing and improved context understanding.

Q: What is attention mechanism in transformers?

A: The attention mechanism allows the model to focus on relevant parts of the input sequence when generating output, improving the model's ability to capture dependencies and context.

Q: What is the difference between GPT and BERT?

A: GPT is a unidirectional transformer model designed for generative tasks, while BERT is a bidirectional transformer model focused on understanding context for tasks like question answering and classification.

Q: How is the training data for LLMs typically collected?

A: Training data for LLMs is usually collected from large-scale text corpora, including books, articles, websites, and other publicly available text sources.

Q: What are the ethical concerns associated with LLMs?

A: Ethical concerns include the potential for generating biased or harmful content, misuse for disinformation, and issues related to privacy and data security.

Q: How do LLMs handle out-of-vocabulary words?

A: LLMs use subword tokenization methods, such as Byte Pair Encoding (BPE) or WordPiece, to break down out-of-vocabulary words into known subwords or characters.

Q: What is the main difference between a chat-based LLM and a pretrained LLM?

A: The main difference lies in their primary use case and fine-tuning:

- Pretrained LLMs are general-purpose models trained on vast amounts of text data to understand and generate human-like text. They can be used for various natural language processing tasks but may require additional fine-tuning for specific applications.
- Chat-based LLMs are specifically fine-tuned for conversational interactions. They are often based on pretrained LLMs but have undergone additional training on dialogue data to improve their ability to maintain context, understand user intent, and generate more natural, contextually appropriate responses in a conversation.

Q: What are some challenges in developing chat-based LLMs?

A: Some key challenges include:

- Maintaining context over long conversations
- Ensuring consistency in responses
- Handling ambiguity and understanding user intent
- Generating safe and appropriate responses
- Balancing between staying on-topic and being flexible in conversations

- Managing biases present in training data
- Optimizing response time for real-time interactions

Q: How do chat-based LLMs handle multi-turn conversations?

A: Chat-based LLMs handle multi-turn conversations through:

- **Maintaining conversation history:** The model considers previous messages in the conversation to maintain context.
- **Context window:** A limited amount of prior conversation is included in each input to the model.
- **Special tokens:** Markers to indicate speaker changes or separate different parts of the conversation.
- **Fine-tuning on dialogue data:** This helps the model learn patterns specific to conversational exchanges.
- **Memory mechanisms:** Some advanced systems may use external memory to store and retrieve relevant information from earlier in the conversation.

Q: What is the "temperature" parameter in LLMs, and how does it affect the output?

A: The temperature parameter controls the randomness of the model output. It is a non-negative value - most LLM APIs support a range of 0 to 2 (OpenAI, Anthropic, Google). There is no strict upper bound, though values above 2 are rarely practical in production:

- **Lower temperature (closer to 0):** Produces more deterministic, focused, and conservative outputs. The model is more likely to choose the most probable next token.
- **Higher temperature (above 1.0):** Leads to more diverse, creative, and sometimes unpredictable outputs. The model distributes probability more evenly across tokens, increasing randomness and variety. Values of 1.3-1.8 are common for creative tasks.

A temperature of 0 will always select the most likely next token, while a temperature approaching infinity will select tokens randomly.

Q: What is the "max tokens" parameter, and why is it important?

A: The max tokens parameter sets the maximum length of the model's output in tokens. It's important because:

- It helps control the length of the generated text.
- It can prevent the model from producing excessively long or rambling responses.
- It's useful for managing computational resources and response time, especially in real-time applications.

- It can be adjusted based on the specific task or user requirements.

Q: Can you explain the concept of "top-p" (or nucleus) sampling?

A: Top-p sampling, also known as nucleus sampling, is an alternative to temperature-based sampling:

- It involves choosing from the smallest possible set of tokens whose cumulative probability exceeds the probability p.
- This method can produce more diverse outputs while avoiding low-probability tokens that might lead to incoherent text.
- A typical value might be around 0.9, meaning the model considers only the most likely tokens that together comprise 90% of the probability mass.

Q: What is the "presence penalty" in some LLM APIs, and how does it work?

A: The presence penalty is a parameter that influences token selection based on whether a token has appeared in the text so far:

- It applies a penalty to tokens that have already appeared in the generated text.
- This encourages the model to use a more diverse vocabulary and avoid repetition.
- A higher presence penalty makes the model less likely to repeat tokens it has already used.

Q: What is the "frequency penalty" parameter, and how does it differ from the presence penalty?

A: The frequency penalty is similar to the presence penalty but works slightly differently:

- It applies a penalty to tokens based on how often they've appeared in the generated text so far.
- Unlike the presence penalty, which only considers whether a token has appeared at all, the frequency penalty increases with each occurrence of a token.
- This can help reduce not just repetition of specific words, but also overuse of certain phrases or patterns.

Q: What is "top-k" sampling, and how does it compare to other sampling methods?

A: Top-k sampling is another method for controlling the randomness of LLM outputs:

- It involves selecting the next token only from the k most likely options.
- This can help prevent the selection of highly improbable tokens while still allowing for some randomness.
- Compared to temperature sampling, top-k provides more direct control over the pool of possible next tokens.
- However, it can be less flexible than top-p sampling, as it uses a fixed number of options regardless of the probability distribution.

Q: How does the "stop sequence" parameter work in LLM APIs?

A: The stop sequence parameter allows you to specify one or more sequences that will cause the model to stop generating further output:

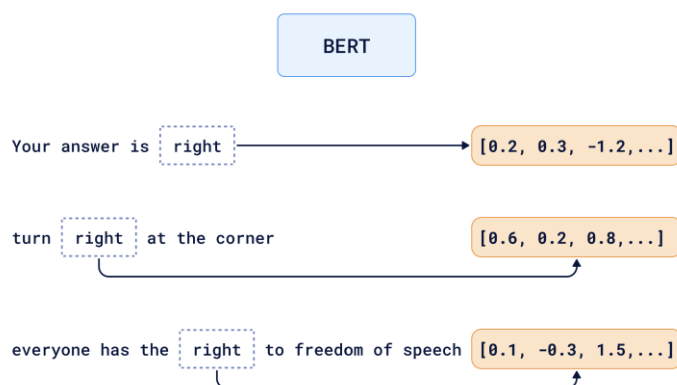
- When the model generates any of the specified stop sequences, it will terminate the response at that point.
- This is useful for controlling the format of the output, especially in structured generation tasks.
- For example, you might use "\n" as a stop sequence to generate single-line responses, or "Q:" to stop after generating an answer in a Q&A format.

Q: What is transfer learning in the context of LLMs?

A: Transfer learning involves pre-training an LLM on a large dataset and then fine-tuning it on a smaller, task-specific dataset to adapt it to specific applications.

Embedding Models

[Google Colab Notebook](#)



Source: <https://qdrant.tech/articles/what-are-embeddings/>

Q: What are word embeddings?

A: Word embeddings are dense vector representations of words that capture their semantic meaning and relationships, enabling models to process text in a numerical form.

Q: How are embeddings generated?

A: Embeddings are generated using techniques like Word2Vec, GloVe, or through transformer-based models that learn context-dependent representations during training.

Q: What is the purpose of embeddings in NLP?

A: Embeddings enable models to understand and manipulate text by representing words and phrases as vectors in a continuous space, improving performance on various NLP tasks.

Q: What is the difference between static and dynamic embeddings?

A: Static embeddings, like Word2Vec and GloVe, provide a fixed representation for each word, while dynamic embeddings, like those from BERT, adjust based on the context in which the word appears.

Q: What are contextual embeddings?

A: Contextual embeddings are dynamic embeddings that capture the meaning of words based on their surrounding context, improving understanding and accuracy in NLP tasks.

Q: How do transformer models generate embeddings?

A: Transformer models generate embeddings through multiple layers of self-attention and feed-forward networks, capturing rich, context-dependent representations of text.

Q: What is the role of position embeddings in transformers?

A: Position embeddings provide information about the position of each token in the input sequence, helping the model understand the order and structure of the data.

Q: How are sentence embeddings different from word embeddings?

A: Sentence embeddings represent entire sentences or phrases as vectors, capturing the overall meaning, while word embeddings represent individual words.

Q: What are the applications of sentence embeddings?

A: Applications include sentence similarity, paraphrase detection, document retrieval, and clustering.

Q: How can embeddings be used for text classification?

A: Embeddings transform text into vectors that can be fed into machine learning models, such as neural networks, for classification tasks like sentiment analysis or topic categorization.

Retrieval-Augmented Generation (RAG)

[PDF & Notebook](#)

Q: What is Retrieval-Augmented Generation (RAG)?

A: RAG is a framework that combines retrieval-based methods with generative models to improve the accuracy and relevance of generated content by incorporating external information.

Q: How does RAG work?

A: RAG retrieves relevant documents or passages from a knowledge base and uses this information to guide the generation process, enhancing the output with contextually accurate details.

Q: What are the components of a RAG model?

A: A RAG model typically consists of a retriever, which fetches relevant information, and a generator, which uses the retrieved information to produce the final output.

Q: What are the benefits of using RAG?

A: Benefits include improved accuracy, relevance, and informativeness of generated content, as the model can access and incorporate external knowledge.

Q: What are the challenges associated with RAG?

A: Challenges include the complexity of integrating retrieval and generation components, handling large-scale knowledge bases, and ensuring the retrieved information is correctly utilized.

Q: How can RAG be applied to question answering?

A: In question answering, RAG retrieves relevant documents or passages and uses them to generate precise and contextually accurate answers to the given questions.

Q: What datasets are commonly used for training RAG models?

A: Common evaluation benchmarks for RAG systems include Natural Questions, TriviaQA, HotpotQA (multi-hop reasoning), MS MARCO, and WebQuestions. Note: these are primarily evaluation benchmarks, not datasets used to train RAG systems from scratch. The retriever is typically a pretrained embedding model fine-tuned for retrieval (e.g., DPR, BGE, E5, Cohere Embed) rather than trained specifically on these datasets.

Q: How does RAG improve over traditional generative models?

A: RAG improves generative models by providing access to external, up-to-date information, which helps generate more accurate and contextually relevant responses.

Q: What role does the retriever play in a RAG model?

A: The retriever fetches the most relevant documents or passages from a knowledge base, providing the generator with contextually useful information to enhance the output.

Q: How can you evaluate the performance of a RAG model?

A: Modern RAG evaluation uses purpose-built frameworks and metrics. RAGAS provides: Answer Faithfulness (is the answer grounded in retrieved context?), Answer Relevance (does it address the question?), Context Precision (are retrieved chunks relevant?), and Context Recall (was all needed information retrieved?). LLM-as-judge approaches use a secondary LLM to score response quality. Retrieval-specific metrics include MRR (Mean Reciprocal Rank), NDCG, and Hit Rate. Traditional BLEU and ROUGE are n-gram overlap metrics designed for translation/summarization - they are not well-suited for evaluating open-ended RAG generation quality.

Q: What is iterative refinement in RAG?

A: A technique where the model repeatedly retrieves and generates, refining its output based on intermediate results.

Q: How can RAG systems be evaluated?

A: Through metrics like relevance, coherence, factual accuracy, and comparison with human-generated responses.

Q: What is the role of attention mechanisms in RAG?

A: They help the generator focus on the most relevant parts of retrieved information during text generation.

Q: How can RAG be adapted for domain-specific applications?

A: By using specialized knowledge bases, fine-tuning on domain data, and customizing retrieval strategies.

Q: What are some challenges in scaling RAG systems?

A: Managing large knowledge bases, reducing latency, and maintaining accuracy with increased information volume.

Q: How can RAG be used to improve AI model transparency?

A: By providing sources for generated information, allowing users to verify the origin of the model's knowledge.

LLM Fine Tuning

Q: What is fine-tuning in the context of LLMs?

A: Fine-tuning involves adapting a pre-trained LLM to a specific task or dataset by continuing its training on task-specific data, improving performance on that particular task.

Q: Explain the process of fine-tuning a language model and its importance in NLP applications.

A: Fine-tuning is the process of taking a pre-trained language model and adapting it to a specific task or dataset by continuing training on task-specific data. It's important in NLP applications because it allows us to leverage the general language understanding of large pre-trained models while adapting them to specific domains or tasks. This process typically involves:

1. Selecting a pre-trained model
 2. Preparing task-specific data
 3. Adjusting the model architecture if necessary
 4. Training on the new data with appropriate hyperparameters
 5. Evaluating and iterating to improve performance
- Fine-tuning is crucial as it often results in better performance than training from scratch, requires less data, and takes less time and computational resources.

Q: Why is fine-tuning important for LLMs?

A: Fine-tuning allows LLMs to specialize and improve performance on specific tasks, making them more effective and accurate for targeted applications.

Q: What is the difference between pre-training and fine-tuning?

A: Pre-training involves training a model on a large, diverse dataset to learn general language representations, while fine-tuning adapts the pre-trained model to specific tasks using smaller, task-specific datasets.

Q: How does the size of the fine-tuning dataset affect the model?

A: Larger fine-tuning datasets can lead to better performance and generalization, but even small datasets can significantly improve task-specific accuracy due to the pre-trained model's foundational knowledge.

Q: What are some common challenges in fine-tuning LLMs?

A: Challenges include overfitting to the fine-tuning dataset, computational resource requirements, and ensuring the model maintains generalization capabilities.

Q: What techniques can be used to prevent overfitting during fine-tuning?

A: Techniques include using regularization methods, dropout, early stopping, and data augmentation to ensure the model does not overfit to the fine-tuning dataset.

Q: How can you assess the success of fine-tuning?

A: Success can be assessed using performance metrics relevant to the task, such as accuracy, F1 score, BLEU score, and human evaluation for tasks like text generation.

Q: What is domain adaptation in the context of LLM fine-tuning?

A: Domain adaptation involves fine-tuning an LLM to perform well in a specific domain or field, such as medical text or legal documents, by training on domain-specific data.

Q: What is catastrophic forgetting in the context of fine-tuning, and what strategies can be employed to mitigate it?

A: Catastrophic forgetting in LLM fine-tuning happens when a pre-trained language model forgets what it learned before while trying to learn something new. As the model adjusts its settings to do better on a new task, it can overwrite important knowledge that was helpful for other tasks or its overall understanding of language.

Strategies to mitigate this include:

1. **Gradual unfreezing:** Starting fine-tuning with most layers frozen and gradually unfreezing them.
2. **Multi-task learning:** Training on multiple tasks simultaneously to maintain general knowledge.
3. **Elastic Weight Consolidation (EWC):** Adding a penalty term to the loss function to prevent drastic changes to important parameters.
4. **Rehearsal methods:** Periodically revisiting samples from the original pre-training dataset.
5. **Layer-wise learning rate decay:** Using lower learning rates for earlier layers of the model. These techniques help balance retaining general knowledge with adapting to new tasks.

Q: Explain the concept of transfer learning in the context of LLM fine-tuning. What are its advantages and potential limitations?

A: Transfer learning in LLM fine-tuning involves using a model pre-trained on a large, general dataset and adapting it to a new, often related task. Advantages include:

1. Reduced training time and computational resources
2. Better performance, especially with limited task-specific data
3. Leveraging general language understanding for specific tasks
4. Ability to adapt to new domains quickly

Potential limitations include:

1. Negative transfer, where pre-trained knowledge interferes with the new task

2. Difficulty in adapting to vastly different domains or languages
3. Potential biases inherited from the pre-trained model
4. Challenges in fine-tuning very large models with limited resources

Q: Describe the role of regularization techniques in fine-tuning LLMs. How do methods like dropout and L2 regularization work, and when should they be applied?

A: Regularization techniques help prevent overfitting during fine-tuning by adding constraints to the learning process. Two common methods are:

1. **Dropout:** Randomly "drops out" a proportion of neurons during training, forcing the network to learn more robust features. It's particularly useful in fine-tuning to prevent the model from relying too heavily on specific patterns in the new data.
2. **L2 regularization:** Adds a penalty term to the loss function based on the squared magnitude of parameters. This encourages the model to use all of its inputs a little rather than some of its inputs a lot, leading to better generalization.

These techniques should be applied when there's a risk of overfitting, particularly when fine-tuning on small datasets or for many epochs.

Q: What is RLHF (Reinforcement Learning from Human Feedback), and how does it enhance the fine-tuning process for language models?

A: RLHF is a technique that incorporates human feedback into the fine-tuning process. It works as follows:

1. The model generates responses to prompts.
2. Human evaluators rate these responses based on quality, relevance, safety, etc.
3. These ratings are used to train a reward model.
4. The language model is then fine-tuned using reinforcement learning, with the reward model providing the reward signal.

RLHF enhances fine-tuning by:

- Aligning the model's outputs more closely with human preferences
- Improving the quality and relevance of generated content
- Helping to mitigate unsafe or biased outputs
- Allowing for more nuanced optimization beyond traditional supervised learning

This technique has been crucial in developing more capable and aligned language models, particularly for conversational AI applications.

Q: Explain the concepts of LoRA and QLoRA. How do these techniques improve the efficiency of fine-tuning large language models?

A: LoRA (Low-Rank Adaptation) and QLoRA (Quantized Low-Rank Adaptation) are techniques designed to make fine-tuning of large language models more efficient:

LoRA:

- Decomposes weight update matrices into lower-rank matrices
- Significantly reduces the number of trainable parameters
- Allows for efficient adaptation of specific parts of the model

QLoRA:

- Combines LoRA with quantization techniques
- Further reduces memory usage by using lower precision (e.g., 4-bit) representations
- Enables fine-tuning of larger models on consumer-grade hardware

These techniques improve efficiency by:

1. Reducing memory requirements
2. Decreasing computational costs
3. Enabling faster training and inference
4. Allowing for more efficient storage and deployment of fine-tuned models

They're particularly valuable when working with very large language models or in resource-constrained environments.

Q: What is the importance of evaluation metrics in fine-tuning, and how do you choose appropriate metrics for different tasks?

A: Evaluation metrics are crucial in fine-tuning as they:

1. Provide quantitative measures of model performance
2. Guide the optimization process
3. Help in comparing different models or fine-tuning approaches
4. Indicate when to stop training to prevent overfitting

Choosing appropriate metrics depends on the task:

- **For classification:** accuracy, precision, recall, F1 score
- **For generation tasks like summarization:** ROUGE score
- **For translation:** BLEU score
- **For question-answering:** Exact Match and F1 score
- **For language modeling:** perplexity

It's often beneficial to use multiple metrics to get a comprehensive view of performance.

Q: Describe the process of hyperparameter tuning in the context of fine-tuning. What strategies can be employed to efficiently optimize hyperparameters?

A: Hyperparameter tuning involves finding the optimal set of hyperparameters (e.g., learning rate, batch size, number of epochs) for fine-tuning. The process typically includes:

1. Defining the hyperparameter search space
2. Choosing a search strategy
3. Training models with different hyperparameter configurations
4. Evaluating performance on a validation set
5. Selecting the best-performing configuration

Efficient strategies include:

- Grid Search: Exhaustive search over specified parameter values
- Random Search: Randomly sampling from the parameter space
- Bayesian Optimization: Using probabilistic models to guide the search
- Population-Based Training: Evolving a population of models
- Learning Rate Finder: Specifically for finding optimal learning rates

Q: What is continual learning in the context of LLMs, and how does it differ from traditional fine-tuning?

A: Continual learning is the process of continuously updating a model as new data becomes available, without forgetting previously learned information. In the context of LLMs:

- Traditional fine-tuning typically involves a one-time adaptation to a specific task or domain.
- Continual learning aims to incrementally update the model over time, maintaining performance on old tasks while learning new ones.

Key aspects of continual learning include:

1. Handling concept drift (changes in the underlying data distribution over time)
2. Balancing stability (retaining old knowledge) and plasticity (acquiring new knowledge)
3. Efficiently incorporating new data without full retraining
4. Mitigating catastrophic forgetting

Techniques for continual learning in LLMs might include:

- Techniques for continual learning in LLMs include: Elastic Weight Consolidation (EWC) - penalizes changes to parameters critical for previous tasks; Rehearsal / Memory Replay - periodically replaying samples from previous training data; Progressive Neural Networks - adding new model capacity per task while freezing prior modules; Adapter Layers - inserting small trainable modules per task without modifying base weights; Parameter Isolation - allocating separate parameter subsets per task; LoRA-based continual fine-tuning - using low-rank adapters to incrementally add task capabilities without catastrophic forgetting.

Misc Topics in Generative AI

Q: What is zero-shot learning in LLMs?

A: Zero-shot learning enables an LLM to perform tasks it has not been explicitly trained on by leveraging its general knowledge and understanding from pre-training.

Q: What is few-shot learning in LLMs?

A: Few-shot learning involves adapting an LLM to a new task with only a few examples, demonstrating the model's ability to generalize from limited data.

Q: What are foundation models?

A: Foundation models are large pre-trained models that serve as a base for various downstream tasks through fine-tuning, leveraging their broad, general-purpose capabilities.

Q: How is federated learning used in the context of LLMs?

A: Federated learning involves training LLMs across multiple decentralized devices while keeping data local, enhancing privacy and enabling collaborative learning without sharing raw data.

Q: What are the benefits of using federated learning with LLMs?

A: Benefits include improved privacy, reduced data transfer, and the ability to leverage diverse data sources without centralizing data storage.

Q: What is the significance of prompt engineering?

A: Prompt engineering involves designing effective input prompts to guide LLMs in generating the desired output, optimizing model performance for specific tasks.

Q: How can LLMs be used for code generation?

A: LLMs can generate code snippets or entire programs by understanding natural language descriptions of the desired functionality, aiding in software development and automation.

Q: What is the role of reinforcement learning in fine-tuning LLMs?

A: Reinforcement learning can fine-tune LLMs by optimizing for specific rewards, such as generating more relevant or accurate responses in conversational agents.

Q: What are some recent advancements in LLMs?

A: Recent advancements in LLMs include: (1) Reasoning models - OpenAI o1/o3 and DeepSeek-R1 use extended test-time compute and chain-of-thought for complex reasoning; (2) Mixture of Experts (MoE) - Mixtral, GPT-4, DeepSeek-V3 use sparse routing

for better quality-to-cost ratios; (3) Massive context windows - Gemini 1.5 (1M tokens), Claude 3.5 (200K); (4) Native multimodality - models processing text, images, audio, video (GPT-4o, Gemini 1.5, Claude 3); (5) Function/tool calling - LLMs natively invoke external APIs and tools; (6) DPO replacing PPO-RLHF for open-source alignment; (7) Speculative decoding for inference speedup; (8) Efficient small models (Llama 3.2 1B, Phi-3, Gemma 2) matching larger model performance.

Q: How do you ensure the ethical use of LLMs?

A: Ensuring ethical use involves implementing guidelines for fairness, transparency, and accountability, as well as actively monitoring and mitigating biases and potential misuse of the models.

Prompt Engineering

[Google Colab Notebook](#)

Q: What is prompt engineering in the context of language models?

A: Prompt engineering involves designing and optimizing input prompts to guide language models (LMs) like GPT-3 or GPT-4 to generate desired outputs. For example, asking GPT-4 "Write a short story about a brave knight" helps generate a coherent and relevant narrative.

Q: Why is prompt engineering important for effective use of Language Models?

A: Effective prompt engineering can significantly improve the relevance, accuracy, and creativity of the model's outputs by providing clear and structured guidance. For example, specifying "Generate a formal letter of recommendation" instead of a vague "Write a letter" leads to more appropriate content.

Q: What is a basic structure of a prompt for an Language Model?

A: A basic prompt should include context, instructions, and a desired output format. For instance, "Write a summary of the following article: [insert article text here]" provides clear guidance.

Q: What are some techniques for crafting effective prompts?

A: Techniques include specifying the task clearly, providing examples, using role-playing scenarios, and setting the tone. For example, "As a travel guide, describe the best tourist attractions in Paris."

Q: How does role-playing enhance prompt effectiveness?

A: Role-playing helps the model adopt a specific perspective, leading to more relevant and targeted responses. For example, "As a financial advisor, give tips on saving for retirement."

Q: How do example-based prompts improve model responses?

A: Providing examples helps the model understand the expected output format and content. For instance, "Translate the following sentence into French: 'Hello, how are you?'" Example: 'Bonjour, comment ça va?'"

Q: How can you handle ambiguity in prompts?

A: To reduce ambiguity, be specific and clear about the task requirements. For example, instead of "Write a report," specify "Write a one-page report summarizing the benefits of renewable energy."

Q: How does the length of a prompt affect model performance?

A: While longer prompts can provide more context, they may also introduce complexity. It's important to balance detail with clarity. For instance, a concise but clear prompt like "Summarize the following text in 50 words" is effective.

Q: How do you set the tone in a prompt?

A: The tone can be set by specifying the desired style or formality. For example, "Write a friendly and informal email to invite a friend to a party."

Q: What is iterative prompt refinement?

A: Iterative prompt refinement involves testing and tweaking prompts to improve the model's outputs. For example, adjusting "Describe a product" to "Describe the key features of the new smartphone model."

Q: What are prompt templates and how are they used?

A: Prompt templates are predefined structures for common tasks. They help ensure consistency and efficiency. For example, using a template like "Dear [Recipient], I am writing to [Purpose]."

Q: How do you evaluate the effectiveness of a prompt?

A: Evaluation involves checking the relevance, accuracy, and coherence of the model's output. For example, comparing the responses generated by different prompts for the same task to see which is more appropriate.

Q: How can you avoid introducing bias in prompts?

A: Avoid bias by using neutral and inclusive language, and by testing prompts with diverse datasets. For example, ensuring a prompt for job application advice does not favour a specific demographic.

Q: How can prompts encourage creative outputs from LLMs?

A: Encouraging creativity involves using open-ended and imaginative prompts. For example, "Write a sci-fi story set in the year 3000."

Q: How do you tailor prompts for specific domains (e.g., legal, medical)?

A: Tailoring prompts involves using domain-specific terminology and context. For example, "As a legal advisor, summarize the key points of this contract."

Q: What are common pitfalls in prompt engineering?

A: Pitfalls include being too vague, overly complex, or biased. For instance, a vague prompt like "Write about technology" can lead to unfocused outputs.

Q: What are conditional prompts and how are they used?

A: Conditional prompts set specific conditions or scenarios. For example, "If you were an astronaut on Mars, describe your daily routine."

Q: What is multi-turn prompting?

A: Multi-turn prompting involves a sequence of prompts to build on previous responses. For example, first asking "Describe the plot of a novel," then "Outline the main character's journey."

Q: How can pre-trained prompts be leveraged in prompt engineering?

A: Pre-trained prompts from model documentation or community resources can be adapted for specific tasks. For instance, using OpenAI's example prompts as a starting point.

Q: How do you handle unexpected or irrelevant outputs from a prompt?

A: Refine the prompt to be more specific and test iteratively. For example, if the prompt "Describe a holiday destination" results in irrelevant information, adjust to "Describe the best tourist attractions in Tokyo."

Q: How can you craft a prompt for text summarization?

A: Use clear instructions and length constraints. For example, "Summarize the following article in 100 words."

Q: How can prompts be used for data augmentation in NLP tasks?

A: Prompts can generate synthetic data to augment training datasets. For instance, generating additional examples of customer service interactions using prompts.

Q: What is adaptive prompting?

A: Adaptive prompting involves dynamically adjusting prompts based on intermediate outputs. For example, if an initial prompt leads to incomplete information, a follow-up prompt seeks clarification.

Q: How is prompt engineering used in interactive applications like chatbots?

A: Prompts guide the chatbot's responses to maintain coherence and relevance. For example, structuring prompts to handle multi-turn conversations effectively.

Q: Provide a case study example of an effective prompt for content generation.

A: In content marketing, a prompt like "Write a blog post on the benefits of remote work, focusing on productivity and work-life balance" yields focused and relevant content that meets marketing objectives.

Q: Why is clarity important when designing prompts?

A: Clarity ensures that the model understands the task requirements accurately. For example, instead of "Write an article," a clearer prompt would be "Write a 500-word article on renewable energy sources."

Q: How do you tailor prompts to the complexity of the task?

A: Prompts should provide sufficient guidance for the task's complexity level. For instance, a more complex task like "Generate code for a neural network architecture" requires a detailed prompt compared to "Summarize a news article."

Q: Why is providing context important in prompts?

A: Context helps the model understand the task's purpose and audience. For example, "As a travel blogger, describe your recent trip to Italy" provides context for the narrative style and content.

Q: How do you incorporate constraints into prompts?

A: Constraints ensure that the model produces outputs within specific boundaries. For example, "Write a tweet promoting a new product in 280 characters or less" sets a character limit constraint.

Q: How do you balance flexibility and guidance in prompts?

A: Prompts should offer enough flexibility for creativity while providing clear guidance. For example, "Write a short story with a surprise ending" offers flexibility in the narrative while guiding the story structure.

Q: How do you structure multi-part prompts effectively?

A: Multi-part prompts should clearly delineate each part to avoid confusion. For example, "Part 1: Describe the setting. Part 2: Introduce the main characters. Part 3: Outline the conflict."

Q: How do you ensure prompts align with the desired outputs?

A: Prompts should clearly articulate the expected format and content of the output. For example, "Generate a product review with pros and cons listed in bullet points" specifies the output format.

Q: Why is iterative refinement important in prompt design?

A: Iterative refinement allows for fine-tuning prompts based on model performance and feedback. For example, refining a prompt based on initial model outputs to achieve more accurate responses.

Q: How do you test prompts with diverse inputs?

A: Testing prompts with a variety of input examples helps ensure robustness and generalization. For example, testing a translation prompt with sentences of varying complexity and languages.

Q: How does prompt design impact user experience?

A: Well-designed prompts enhance user experience by providing clear instructions and achieving desired outcomes efficiently. For example, a user-friendly chatbot prompt leads to quicker and more accurate responses.

Q: How do you anticipate model responses when designing prompts?

A: Understanding the model's capabilities and limitations helps craft prompts that elicit desired responses. For example, designing a prompt that accounts for potential biases or inaccuracies in the model's output.

Q: How do you address ambiguity in prompt design?

A: Clarifying instructions and providing examples can help reduce ambiguity in prompts. For example, specifying a range for numerical inputs or providing context for ambiguous terms.

Q: How can iterative prompt design optimize model performance?

A: Iteratively refining prompts based on model feedback improves prompt effectiveness over time. For example, adjusting the level of detail or complexity based on initial model responses.

Q: How do you adapt prompts for different model architectures?

A: Prompts should be tailored to leverage the strengths and nuances of specific model architectures. For example, structuring a prompt differently for a transformer-based model like GPT-3 compared to a recurrent neural network.

Q: How does providing feedback on model responses inform prompt design?

A: Analyzing model outputs and user feedback helps refine prompts to better align with desired outcomes. For example, adjusting a prompt based on common errors or misunderstandings in model responses.

Q: How do you verify that a prompt is understandable to the model?

A: Testing prompts with diverse inputs and analyzing model responses ensures understandability. For example, evaluating whether the model produces relevant outputs consistent with the prompt's intent.

Q: How can transfer learning inform prompt design?

A: Leveraging pre-trained models and transfer learning techniques can inform prompt design for specific tasks. For example, adapting prompts based on prompts that have been successful in similar domains.

Q: How does collaborative prompt design enhance prompt effectiveness?

A: Collaborative prompt design involves input from domain experts, users, and data scientists to ensure prompts are well-suited for the task. For example, involving subject matter experts in crafting prompts for specialized domains like healthcare or finance.

Q: How do you address bias and fairness considerations in prompt design?

A: Careful crafting of prompts and testing with diverse datasets help mitigate bias and ensure fairness. For example, analyzing prompts for language or cultural biases and adjusting accordingly.

One-Shot Prompting

Q: What is one-shot prompting in the context of Generative AI?

A: One-shot prompting involves providing a single example or prompt to a model to perform a task, requiring the model to generalize and generate outputs based on a minimal amount of information.

Q: How does one-shot prompting differ from traditional prompt-based approaches?

A: Unlike traditional prompt-based approaches that rely on multiple examples or structured prompts, one-shot prompting challenges models to generalize from a single instance, making it more efficient and adaptable to new tasks or domains.

Q: What are the advantages of using one-shot prompting over traditional prompt-based approaches?

A: One-shot prompting requires less data and human intervention, making it more scalable and efficient for generating content across diverse domains or languages.

Q: What role does transfer learning play in enabling one-shot prompting?

A: Transfer learning enables models pretrained on large datasets to extract relevant features and knowledge from a single prompt, leveraging prior learning to generate contextually relevant outputs.

Q: What are some practical applications of one-shot prompting in Generative AI?

A: Applications include language translation, text summarization, question answering, and content generation tasks where input examples are limited or scarce.

Q: What are some challenges associated with implementing one-shot prompting?

A: Challenges may include ensuring model robustness and generalization across diverse tasks, domains, and languages, as well as addressing biases or limitations in the training data.

Few-Shot Prompting

Q: What is few-shot learning in the context of Generative AI?

A: Few-shot learning involves training models with a small number of examples to perform tasks. For instance, providing only a few examples of poetry to a language model to generate new poems.

Q: How does few-shot learning differ from traditional supervised learning?

A: Few-shot learning requires fewer labeled examples compared to traditional supervised learning, making it more adaptable to new tasks or domains with limited data.

Q: Can you explain the concept of meta-learning in few-shot learning?

A: Meta-learning involves training a model to learn how to learn from limited data. For example, a meta-learning algorithm can enable a model to quickly adapt to new tasks with minimal examples.

Q: What are some techniques used to implement few-shot learning?

A: Techniques include meta-learning algorithms like MAML (Model-Agnostic Meta-Learning) and transfer learning approaches such as fine-tuning pretrained models on few examples.

Zero-Shot Prompting

Q: What is zero-shot learning and how does it work in Generative AI?

A: Zero-shot learning enables models to perform tasks without specific training examples by leveraging prior knowledge. For example, GPT-3 can translate languages it has never been trained on by understanding linguistic patterns.

Q: How does zero-shot learning differ from few-shot learning?

A: Zero-shot learning requires no training examples for a specific task, while few-shot learning uses a small number of examples. Zero-shot learning relies more on generalization.

Q: What are some practical applications of zero-shot learning in Generative AI?

A: Applications include machine translation, text summarization, and content generation in languages or domains with limited training data

Chain of Thought Prompting

Q: What is Chain of Thought Prompt Engineering in Generative AI?

A: Chain of Thought (CoT) prompting is a technique where intermediate reasoning steps are included in the prompt to elicit explicit step-by-step reasoning from the model before it produces a final answer. Introduced by Wei et al. (2022), it significantly improves performance on complex reasoning tasks (arithmetic, commonsense, multi-step logic). Two forms: (1) Few-shot CoT - provide exemplar question-reasoning-answer triplets in the prompt showing reasoning steps; (2) Zero-shot CoT - append 'Let's think step by step' to trigger reasoning without examples. CoT is a single-prompt technique, not a multi-turn interaction pattern. It works best with larger models (100B+ parameters).

Q: How does Chain of Thought Prompt Engineering differ from traditional prompt engineering approaches?

A: Chain of Thought Prompt Engineering focuses on structuring prompts in a sequential manner to create a coherent narrative or logical progression of ideas, whereas traditional prompt engineering may involve standalone prompts for individual tasks.

Q: Can you provide an example of how Chain of Thought Prompt Engineering can be applied in content generation tasks?

A: Sure, for generating a story, the chain of prompts could start with an initial prompt like "Introduce the main character," followed by prompts like "Describe the setting," "Introduce the conflict," "Detail the character's actions," and "Resolve the conflict."

Q: What are some key considerations when designing a chain of thought prompts?

A: Considerations include maintaining coherence and consistency between prompts, ensuring a logical flow of ideas, and providing clear instructions for each step in the chain.

Q: How can Chain of Thought Prompt Engineering be leveraged to guide multi-turn conversations or interactions?

A: By structuring prompts as sequential steps, Chain of Thought Prompt Engineering can guide the model through a dialogue or interaction, ensuring that each turn builds upon the previous one to maintain coherence and relevance.

Q: What role does context play in Chain of Thought Prompt Engineering?

A: Contextual information provided in each prompt helps the model understand the progression of thoughts or actions and ensures that subsequent prompts are relevant and appropriate.

Q: How do you ensure that the chain of thought prompts leads to desired outcomes or objectives?

A: By carefully designing each prompt in the sequence to align with the overall goal or objective, and by iteratively refining the prompts based on model performance and user feedback.

Q: What are some challenges associated with implementing Chain of Thought Prompt Engineering?

A: Challenges may include maintaining coherence and relevance across multiple prompts, managing the complexity of the chain, and ensuring that each prompt effectively guides the model towards the desired outcome.

Hybrid Prompting

Q: What is Hybrid Prompting in the context of Generative AI?

A: Hybrid Prompting combines multiple prompt engineering techniques, such as using both structured prompts and open-ended prompts, to guide model behaviour and enhance output quality.

Q: How does Hybrid Prompting leverage the strengths of different prompt engineering approaches?

A: Hybrid Prompting integrates structured prompts for clarity and guidance with open-ended prompts for creativity and flexibility, allowing for a more nuanced and adaptable approach to content generation.

Q: Can you provide an example of how Hybrid Prompting can be applied in text generation tasks?

A: In a storytelling task, Hybrid Prompting could involve providing an initial structured prompt to set the scene and introduce characters, followed by open-ended prompts to allow the model to develop the narrative organically.

Q: What are the advantages of using Hybrid Prompting over individual prompt engineering techniques?

A: Hybrid Prompting offers the benefits of both structured and open-ended prompts, including clear guidance, context, and flexibility, leading to more diverse and high-quality outputs.

Q: How do you determine the optimal balance between structured and open-ended prompts in Hybrid Prompting?

A: The balance depends on the task requirements, desired output characteristics, and model capabilities. Experimentation and iterative refinement are key to finding the right balance.

Q: What role does user feedback play in refining Hybrid Prompting strategies?

A: User feedback helps identify areas where structured prompts may be too restrictive or open-ended prompts may lead to irrelevant outputs, guiding adjustments to the Hybrid Prompting approach.

Q: How can Hybrid Prompting be tailored to suit different types of content generation tasks?

A: By customizing the mix of structured and open-ended prompts based on the specific objectives, constraints, and characteristics of each task, such as adjusting the level of guidance or flexibility as needed.

Q: What are some challenges associated with implementing Hybrid Prompting?

A: Challenges may include designing prompts that effectively balance structure and flexibility, managing the complexity of hybrid prompt sequences, and ensuring coherence and relevance in model outputs.

Q: Can you explain how Hybrid Prompting can be used to address the trade-off between control and creativity in content generation?

A: Hybrid Prompting allows for a fine-tuned balance between providing guidance and allowing for creative exploration, enabling models to generate diverse and engaging content while maintaining control over the overall direction.

Q: How do you evaluate the effectiveness of Hybrid Prompting strategies in improving model performance?

A: Evaluation involves assessing the quality, diversity, and relevance of model outputs generated using Hybrid Prompting compared to other prompt engineering approaches, as well as gathering user feedback to inform further refinement.

ReAct Prompting

Q: What is the concept behind ReAct in Generative AI, and how does it leverage human cognitive processes?

A: ReAct is a method inspired by how humans learn and make decisions. It combines reasoning and action in AI models. ReAct prompts these models to think through a problem and take actions. It's like giving the AI a task and asking it to figure out the best way to solve it. It can also look up information from places like Wikipedia to help with its decision-making.

(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: `Search[Apple Remote]`
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: `Search[Front Row]`
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: `Search[Front Row (software)]`
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: `Finish[keyboard function keys]`

Advance Topics

Graph Retrieval Augmented Generation (GraphRAG)

Useful Blog : <https://github.com/microsoft/graphrag?tab=readme-ov-file>

Q: How would you integrate GraphRAG with a large language model (LLM) to improve the factual accuracy of generated responses in a customer support chatbot?

A: To integrate GraphRAG with an LLM for improved factual accuracy:

- Build a knowledge graph of product information, FAQs, and customer interactions.
- Implement RAG to retrieve relevant subgraphs based on the customer query.
- Use graph embeddings to represent structural information in the knowledge graph.
- Combine retrieved graph information with the customer query as context for the LLM.
- Implement a fact-checking mechanism that compares LLM outputs with graph data.
- Fine-tune the LLM on graph-augmented data to improve coherence between graph and text.
- Use the graph to generate explanations for the LLM's responses.
- This approach would help the chatbot provide more accurate and contextually relevant responses while leveraging the LLM's natural language generation capabilities.
- Allow researchers to interactively explore connections through graph visualizations and LLM-generated summaries.
- This approach would leverage the LLM's language understanding while using the graph structure to uncover non-trivial relationships in the scientific literature.

Q: How would you implement a GraphRAG-enhanced LLM system for generating more accurate and diverse creative writing prompts?

A: To implement a GraphRAG-enhanced LLM for creative writing prompts:

- Build a knowledge graph of literary elements, genres, themes, and their relationships.
- Use RAG to retrieve relevant subgraphs based on user preferences or initial ideas.
- Implement graph traversal algorithms to find unique combinations of literary elements.
- Use the LLM to generate natural language prompts based on the retrieved graph information.

- Apply diversity-promoting techniques in both graph retrieval and LLM generation.
- Implement a feedback mechanism to update the graph based on user ratings of generated prompts.
- Use graph-based clustering to categorize and recommend similar prompts.
- This approach would combine the structured creativity of the graph with the natural language generation of the LLM to produce more diverse and inspiring writing prompts.

Q: In a financial analysis system, how would you use GraphRAG with an LLM to generate more comprehensive and insightful market reports?

A: To use GraphRAG with an LLM for financial market reports:

- Construct a knowledge graph of companies, financial indicators, market events, and news.
- Implement RAG to retrieve relevant financial data and relationships based on report requirements.
- Use graph algorithms to identify trends, correlations, and anomalies in financial data.
- Apply the LLM to generate natural language descriptions of graph-based insights.
- Implement a template system for report structure, using the graph to fill in specific data points.
- Use the LLM to generate explanations for complex financial concepts found in the graph.
- Incorporate a fact-checking mechanism to ensure LLM-generated text aligns with graph data.
- This approach would combine the data-driven insights from the graph with the LLM's ability to generate readable and contextually rich reports.

Q: How would you design a GraphRAG-enhanced LLM system for a personalized learning platform to generate tailored educational content?

A: To design a GraphRAG-enhanced LLM for personalized learning:

- Create a knowledge graph of educational concepts, prerequisites, and learning resources.

- Implement RAG to retrieve relevant educational materials based on the learner's profile and current topic.
- Use graph traversal to determine optimal learning paths and identify knowledge gaps.
- Apply the LLM to generate personalized explanations and examples based on graph data.
- Implement a difficulty scaling mechanism using graph-based complexity measures.
- Use the LLM to rephrase complex concepts retrieved from the graph for better understanding.
- Generate personalized quizzes and exercises by combining graph-based knowledge structure with LLM-generated questions.
- This approach would leverage the structured representation of knowledge in the graph with the LLM's ability to generate engaging and personalized educational content.

Q: In a drug interaction prediction system, how would you integrate GraphRAG with an LLM to improve the accuracy and explainability of predictions?

A: To integrate GraphRAG with an LLM for drug interaction prediction:

- Build a knowledge graph of drugs, their chemical properties, known interactions, and biological pathways.
- Implement RAG to retrieve relevant subgraphs for given drug combinations.
- Use graph neural networks to learn representations of drug interactions.
- Apply the LLM to generate natural language explanations of potential interactions based on graph data.
- Implement a mechanism to translate graph-based predictions into human-readable risk assessments.
- Use the LLM to generate detailed reports on the mechanism of predicted interactions.
- Incorporate a feedback loop where expert knowledge can be used to update both the graph and the LLM.

- This approach would combine the structured prediction power of graph-based models with the LLM's ability to generate understandable explanations for healthcare professionals.

Q: How would you use GraphRAG with an LLM to enhance a code generation system's ability to produce more context-aware and maintainable software?

A: To enhance code generation with GraphRAG and an LLM:

- Create a knowledge graph of programming concepts, design patterns, and code dependencies.
- Use RAG to retrieve relevant code snippets and architectural patterns based on the programming task.
- Implement graph-based static analysis to understand existing codebase structure.
- Apply the LLM to generate code while considering the retrieved graph context.
- Use graph algorithms to suggest optimal placement of new code within the existing structure.
- Implement an LLM-based system to generate meaningful variable names and comments based on graph context.
- Use the graph to check for potential conflicts or inefficiencies in the generated code.
- This approach would help the LLM generate code that is not only syntactically correct but also fits well within the larger software architecture and follows best practices.

Q: In a legal document analysis system, how would you implement GraphRAG with an LLM to improve contract review and risk assessment?

A: To implement GraphRAG with an LLM for legal document analysis:

- Build a knowledge graph of legal terms, clauses, precedents, and their relationships.
- Use RAG to retrieve relevant legal concepts and past cases based on contract content.

- Implement graph-based similarity measures to identify non-standard or high-risk clauses.
- Apply the LLM to generate plain-language summaries of complex legal language.
- Use graph traversal to identify potential conflicts or missing clauses in the contract.
- Implement an LLM-based system to generate suggested modifications for problematic clauses.
- Use the graph structure to provide context for the LLM when explaining legal implications.
- This approach would combine the structured analysis capabilities of the graph with the LLM's natural language understanding to provide more comprehensive and accessible legal document analysis.

Q: How would you design a GraphRAG-enhanced LLM system for generating more accurate and diverse marketing campaign ideas?

A: To design a GraphRAG-enhanced LLM for marketing campaign ideation:

- Create a knowledge graph of marketing channels, audience segments, past campaigns, and performance metrics.
- Implement RAG to retrieve relevant marketing strategies and performance data.
- Use graph algorithms to identify successful patterns and untapped market segments.
- Apply the LLM to generate campaign ideas based on retrieved graph data and user input.
- Implement a diversity-promoting mechanism in both graph retrieval and LLM generation.
- Use the graph to fact-check and ground the LLM's creative ideas in historical performance data.
- Generate explanations for each campaign idea, linking it to relevant nodes in the knowledge graph.
- This approach would combine data-driven insights from the graph with the LLM's creative language generation to produce innovative yet grounded marketing ideas.

Q: In a multi-lingual information retrieval system, how would you use GraphRAG with an LLM to improve cross-language query understanding and result generation?

A: To use GraphRAG with an LLM for multi-lingual information retrieval:

- Build a multi-lingual knowledge graph linking concepts across languages.
 - Implement RAG to retrieve relevant information regardless of the language of the source.
 - Use graph embeddings to capture semantic relationships across languages.
 - Apply the LLM for query translation and expansion using graph-based context.
 - Implement cross-lingual graph traversal to find relevant information in other languages.
 - Use the LLM to generate coherent summaries of retrieved information in the user's preferred language.
 - Implement a mechanism to explain cross-lingual connections found in the graph.
 - This approach would leverage the language-agnostic nature of the graph structure with the LLM's multi-lingual capabilities to provide more comprehensive cross-language information retrieval and presentation.
-

Agentic AI / AI Agents

- [Agentic AI Interview Q&A](#)
- [GitHub](#)

Q: What is an Agentic AI?

A: An Agentic AI is an AI framework that uses large language models to perform tasks autonomously or semi-autonomously. These systems can understand natural language instructions, make decisions, and take actions to achieve specific goals.

Q: How does an LLM Agent differ from a standard LLM?

A: While a standard LLM primarily focuses on generating text based on prompts, an LLM Agent can interact with its environment, make decisions, and take actions. It often integrates with external tools and APIs to perform tasks beyond just text generation.

Q: What are the key components of an LLM Agent system?

A: Key components typically include:

- The LLM itself
- A prompt engineering system

- A memory or context management system
- A decision-making/planning module
- Tool integration for performing actions
- A feedback loop for learning and improvement

Q: Can you explain the concept of "prompt engineering" in the context of Agentic AI?

A: Prompt engineering involves crafting specific instructions or queries that guide the LLM's behaviour and output. For agents, this often includes defining the agent's role, goals, constraints, and available actions in a way that the LLM can understand and act upon.

Q: How do AI Agents handle long-term memory and context?

A: AI Agents often use techniques like vector databases or other persistent storage methods to maintain context over long interactions. They may also summarize past interactions or use retrieval techniques to access relevant information when needed.

Q: What are some common challenges in developing Agentic AI systems?

A: Challenges include:

- Ensuring consistent behaviour and adherence to goals
- Managing context and memory effectively
- Integrating with external tools and APIs securely
- Handling errors and unexpected situations
- Balancing autonomy with safety and control

Q: How can Agentic system be made more reliable and less prone to hallucination?

A: Strategies include:

- Implementing fact-checking mechanisms
- Using retrieval-augmented generation (RAG) to ground responses in verified information
- Employing multiple agents for cross-verification
- Implementing robust error handling and uncertainty quantification

Q: What are some potential applications of Agentic AI systems?

A: Applications include:

- Personal assistants
- Automated customer service
- Research and data analysis assistants
- Code generation and debugging agents
- Task planning and execution systems
- Educational tutors

Q: How do AI Agents make decisions?

A: LLM Agents typically use a combination of:

- Pre-defined rules and constraints
- The LLM's understanding of the task and context
- Heuristics encoded in their prompts
- Feedback from previous actions
- Some advanced systems may also incorporate reinforcement learning or other AI decision-making techniques.

Q: What ethical considerations should be taken into account when developing Agentic systems?

A: Key ethical considerations include:

- Ensuring transparency about the system's capabilities and limitations
- Protecting user privacy and data security
- Avoiding bias and ensuring fairness in decision-making
- Implementing safeguards against misuse or harmful actions
- Considering the potential impact on employment and human roles

Q: How can the performance of an Agentic AI system be evaluated?

A: Evaluation methods may include:

- Task completion rates and quality
- User satisfaction metrics
- Adherence to defined constraints and goals
- Efficiency in resource use (time, computational resources, etc.)

- Robustness in handling various scenarios, including edge cases

Q: What's the difference between single-task and multi-task Agents?

A: Single-task agents are designed to perform a specific function, like customer support for a particular product. Multi-task agents are more versatile and can handle a variety of tasks, often deciding which actions to take based on user input or environmental cues.

Latest Q & A (Based on question asked in different companies)

Q: How do LLMs balance between fluency and factual accuracy?

A: LLMs often prioritize fluency due to their training on language patterns, which can lead to confident-sounding but inaccurate responses. This tendency is sometimes referred to as "hallucination." Strategies to improve accuracy while maintaining fluency include:

- **External grounding:** Using verified information sources during generation.
- **Reinforcement learning:** Fine-tuning with accuracy-focused rewards.
- **Few-shot learning:** Providing accurate examples in prompts.
- **Uncertainty quantification:** Training models to express doubt when unsure.
- **Factual knowledge injection:** Training on curated factual datasets.
- **Constrained decoding:** Limiting improbable text generation.
- **Prompt engineering:** Crafting prompts to elicit accurate responses.

Q: What is an embedding in the context of machine learning and natural language processing?

A: An embedding is a dense vector representation of data (such as words, sentences, or documents) in a continuous vector space. It captures semantic meaning and relationships between data points, allowing for efficient computation and comparison.

Q: How do embedding models handle out-of-vocabulary words?

A: Embedding models typically handle out-of-vocabulary words through techniques like:

- Subword tokenization (e.g., WordPiece, [Byte-Pair Encoding](#))
- Character-level embeddings
- Using a special token for unknown words
- Fallback to nearest known words.
- The specific method depends on the model architecture and training approach.

Q: What is the "[curse of dimensionality](#)" and how does it relate to embeddings?

A: The curse of dimensionality refers to various phenomena that arise when analysing data in high-dimensional spaces. With embeddings, it can lead to:

- Increased computational complexity
- Sparsity of data in the embedding space
- Reduced effectiveness of distance metrics To mitigate this, techniques like dimensionality reduction or using specialized index structures for high-dimensional spaces are often employed.

Q: Can you explain the concept of "semantic search" and how it relates to embeddings?

A: Semantic search refers to search techniques that aim to understand the intent and contextual meaning of the query, rather than just matching keywords. Embeddings enable semantic search by:

- Representing both **queries** and **documents** in the same vector space
- Allowing for similarity comparisons based on meaning, not just exact matches
- Capturing nuanced relationships between concepts This results in more relevant search results, especially for complex or ambiguous queries.

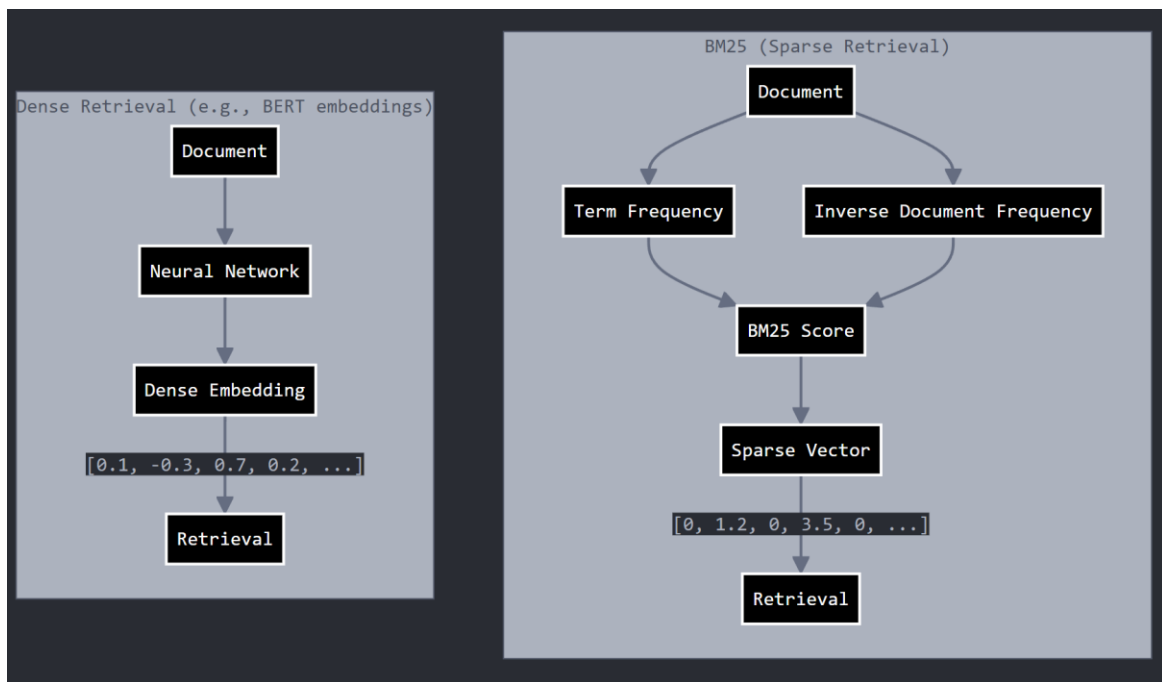
Q: What is the difference between sparse and dense embeddings?

A: Sparse embeddings:

- Have many zero values
- Are typically high-dimensional
- Often used in traditional information retrieval (e.g., TF-IDF vectors)
- Efficient for exact matching

Dense embeddings:

- Have few or no zero values
- Are typically lower-dimensional
- Used in modern neural network-based models
- Better at capturing semantic similarity
- More computationally efficient for many operations



Q: What is the main difference between dense search and sparse search?

A: The main difference lies in their vector representations. Dense search uses continuous, real-valued vectors where each dimension contributes some meaning, while sparse search uses high-dimensional vectors where most elements are zero, typically corresponding to specific features or keywords.

Q: In the context of text search, what might a sparse vector look like compared to a dense vector?

A: For the phrase "quick brown fox":

- Sparse vector: `[0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, ...]` where 1s represent the presence of "quick", "brown", and "fox" in a large vocabulary
- Dense vector: `[0.1, -0.3, 0.7, 0.2, -0.1, ...]` where each number represents some learned semantic feature

Q: In which scenarios would you prefer to use dense search over sparse search?

A: Dense search is preferable in scenarios where:

- Semantic understanding is crucial
- Dealing with natural language queries
- Handling synonyms or related concepts
- Working with out-of-vocabulary words
- Searching for similar items based on meaning rather than exact matches

Q: What are the advantages of sparse search?

A: Advantages of sparse search include:

- Efficiency in large-scale retrieval tasks
- Excellent performance for exact matching and Boolean queries
- Easy interpretability and explainability of results
- Lower computational requirements compared to dense search

Q: How do dense vectors capture semantic meaning?

A: Dense vectors capture semantic meaning by:

- Using machine learning models (often neural networks) to learn representations
- Encoding contextual information into the vector
- Representing words or items in a continuous space where similar concepts are closer together
- Allowing for complex relationships to be captured in the multidimensional space

Q: Can you explain a use case where combining dense and sparse search might be beneficial?

A: A common use case is in e-commerce search engines. Here's how a combined approach could work:

- Sparse search can quickly filter products based on exact attribute matches (e.g., brand, colour, size)
- Dense search can then rank the filtered results based on semantic similarity to the user's query
- This combination provides both the efficiency of sparse search for filtering and the semantic understanding of dense search for ranking

Q: How do you evaluate the quality of embeddings?

A: Embedding quality can be evaluated through various methods:

- **Intrinsic evaluation:** Using tasks like word similarity or analogy tests
- **Extrinsic evaluation:** Performance on downstream tasks (e.g., classification, clustering)
- **Visualization techniques:** t-SNE or UMAP to examine relationships in 2D or 3D

Similarity analysis: Checking if semantically related items are close in the embedding space

Q: If we embed information using one embedding model and query using a different embedding model, will we be able to fetch accurate responses?

A: No, this approach would likely result in poor or inconsistent retrieval performance.

Q: Why wouldn't this work effectively?

A: Different embedding models create vector representations in distinct "embedding spaces" that are not directly comparable. It's akin to trying to compare words in two different languages without translation.

Q: What's the best practice for effective retrieval using embeddings?

A: For effective retrieval, it's crucial to use the same embedding model for both encoding the information and generating the query embeddings. This ensures that both the stored information and the queries are represented in the same vector space.

Q: What happens if you use mismatched embeddings?

A: Using mismatched embeddings would lead to unreliable or meaningless similarity scores. The semantic relationships preserved by each embedding model wouldn't align, making accurate comparisons impossible.

Q: If I need to switch to a new embedding model, what should I do with my existing embedded information?

A: If you need to switch embedding models, you would typically need to re-embed all your stored information using the new model. This maintains consistency and ensures continued retrieval effectiveness.

Q: Why is it important to have both stored information and queries in the same vector space?

A: Having both in the same vector space allows for meaningful similarity comparisons. It ensures that the relationships and distances between vectors are consistent and interpretable, which is essential for accurate information retrieval.

Q: Let's say I am storing different Indian mutual fund scheme names in a vector database. If I want to access a particular scheme like "HDFC Top 100 Fund - Direct Plan - Growth Option", what should be my approach to get the exact result without getting any other results like "HDFC Top 100 Fund - Regular Plan - Growth Option" or "HDFC Mid-Cap Opportunities Fund - Direct Plan - Growth Option"?

A: To retrieve the exact match "HDFC Top 100 Fund - Direct Plan - Growth Option" without getting similar results, you should use a hybrid approach:

- Perform a vector similarity search for "HDFC Top 100 Fund - Direct Plan - Growth Option".
- Apply an exact string match filter on the results of the vector search.
- Use a separate metadata field for the precise mutual fund scheme name for filtering.
- Utilize any database-specific exact matching features provided by your vector database system.

Q: Why is a simple vector similarity search not sufficient for this task in the Indian mutual fund context?

A: A simple vector similarity search is not sufficient because:

- It's designed to find semantically similar items, which could include other HDFC funds or similar funds.
- Vector representations may not distinguish between minor differences in scheme names that are crucial for exact identification (e.g., Direct Plan vs Regular Plan).
- It might return related schemes with similar investment objectives instead of the exact scheme we're looking for.
- In the Indian mutual fund context, even small differences in scheme names can represent significantly different investment products.

Q: How does adding an exact string match improve the search results in this mutual fund scenario?

A: Adding an exact string match:

- Filters out similar but non-identical mutual fund scheme names.
- Ensures that only the precise scheme "HDFC Top 100 Fund - Direct Plan - Growth Option" is returned.
- Eliminates false positives like "HDFC Top 100 Fund - Regular Plan - Growth Option" or "HDFC Mid-Cap Opportunities Fund - Direct Plan - Growth Option".
- Prevents confusion between direct and regular plans, or between growth and dividend options of the same scheme.

Q: How does this system handle updates to mutual fund schemes, such as changes in names or mergers?

A: To handle updates to mutual fund schemes:

- Implement a versioning system to track changes in scheme details over time.

- Use the SEBI-assigned scheme code as a persistent identifier, even if the scheme name changes.
- Maintain a mapping of old scheme names to new ones for historical searches.
- Update vector embeddings when significant changes occur to ensure accurate semantic search.
- Implement a regular update process to sync with source.

Q: As a developer using a third-party LLM API for your e-commerce chatbot, how would you protect against prompt injection attempts?

A: To protect against prompt injection as an LLM API consumer:

- Implement robust input validation to filter out suspicious commands before sending to the API.
- Use a prompt prefix that reinforces the chatbot's role and limitations, e.g., "You are an e-commerce assistant. Do not perform actions outside of product inquiries and purchases."
- Sanitize user inputs to remove potential injection phrases like "ignore previous instructions."
- Implement output filtering to catch unexpected or harmful responses.
- Use API features like content moderation if available.
- Set up monitoring to flag suspicious interactions for human review.

Q: Users are attempting to jailbreak the LLM-powered application you've built. As someone without access to modify the underlying model, how would you address this?

A: To address jailbreak attempts as an LLM API consumer:

- Analyse patterns in jailbreak attempts to identify common techniques.
- Implement stricter input validation based on observed patterns.
- Use system prompts that reinforce ethical boundaries and the intended use of the application.
- Implement a multi-stage filtering process for both inputs and outputs.
- Utilize API parameters for safety and content filtering if provided by the LLM service.
- Develop a list of forbidden prompts or keywords associated with jailbreak attempts.
- Implement user action limits or cooldown periods to prevent rapid-fire attempts.
- Set up logging and alerting for potential jailbreak attempts to enable quick responses.

Q: You're using an LLM API to create an AI study aid. How would you prevent students from manipulating it to do their homework?

A: Strategies to protect an AI study aid as an LLM API consumer:

- Design prompts that encourage explanations rather than direct answers.

- Implement input classification to detect homework-like questions and adjust responses accordingly.
- Use follow-up prompts that ask students to show their work or explain their understanding.
- Set character limits on inputs to prevent entire homework problems from being submitted.
- Develop a bank of educational prompts to guide the LLM towards tutoring rather than problem-solving.
- Implement user sessions with usage patterns analysis to detect potential misuse.
- Create custom output templates that format responses as hints or study guides rather than complete answers.
- Include automated messages about academic integrity and proper use of AI study aids.

Q: You're integrating an LLM into a banking chatbot for customer service. How would you protect against prompt injection or jailbreak attempts that could compromise financial data or operations?

A: To secure a banking chatbot using an LLM API:

- **Strict input validation:** Implement rigorous filters to catch and block potential injection attempts before they reach the LLM.
- **Financial context reinforcement:** Begin each interaction with a strong prompt (system prompt) that reinforces the chatbot's role and limitations in handling financial information.
- **Multi-factor authentication:** Require additional verification for any actions involving sensitive operations or data access.
- **Sensitive information masking:** Develop a system to automatically detect and mask account numbers, SSNs, and other sensitive data in both inputs and outputs.
- **Transaction isolation:** Ensure that the LLM has no direct access to perform financial transactions. Use it for information lookup and guidance only.
- **Response templating:** Create pre-approved response templates for common financial queries to minimize unexpected outputs.
- **Anomaly detection:** Implement real-time monitoring to flag unusual patterns of interaction or requests for sensitive information.
- **Regulatory compliance checks:** Incorporate checks to ensure all responses comply with financial regulations (e.g., GDPR, CCPA).
- **Audit logging:** Maintain detailed logs of all interactions for security audits and potential forensic analysis.
- **Fallback mechanisms:** Implement automatic escalation to human operators for any detected anomalies or high-risk queries.

- **Regular security testing:** Conduct frequent penetration testing and security audits specific to LLM-based financial interactions.
- **Customer education:** Provide clear guidelines to users about the chatbot's capabilities and limitations, especially regarding financial transactions and data handling.

Latest top LLM Leaderboard

We can find latest top LLMs at following leaderboard.

LMSYS Leaderboard: <https://lmarena.ai/?leaderboard>

Modern LLM Topics (2024-2026)

Essential topics for current GenAI interviews not covered in earlier sections.

Q: What is Mixture of Experts (MoE) and why is it important for modern LLMs?

A: Mixture of Experts is an architecture where the model contains multiple specialized sub-networks called 'experts', with a gating/router network that routes each input token to only a subset (e.g., top-2 out of N experts).

Only selected experts are activated per token, sparse activation, meaning the effective compute per forward pass is much less than the total parameter count.

Key benefits:

(1) Larger total parameter count without proportionally larger compute per token

(2) Experts can specialize in different types of content or language. Examples: Mixtral 8x7B (8 experts, 2 active, 13B active out of 47B total), GPT-4 (reported MoE), DeepSeek-V3 (671B total, 37B active per token).

Challenge: MoE models require more GPU memory to hold all expert weights at inference time, even though only a fraction are used per step.

Q: What is function calling (tool use) in LLMs, and how does it work?

A: Function calling (tool use) is a native LLM capability where the model can decide to invoke external tools or APIs as part of generating a response.

Workflow:

- (1) Define available tools with their schemas (name, parameters, descriptions)
- (2) The LLM generates a structured tool call when needed
- (3) Your code executes the tool and returns the result
- (4) The LLM uses the result to produce the final response. This enables LLMs to access real-time data (search, databases), perform calculations, interact with external systems, and is the foundation of modern AI agents. Supported natively by OpenAI, Anthropic, Google, and others. Parallel function calling allows invoking multiple tools simultaneously within a single response.

Q: What is a KV cache in LLMs, and why is it important for inference efficiency?

A: The KV (Key-Value) cache stores the key and value tensors computed during self-attention for all previously processed tokens. During autoregressive text generation, each new token must attend to all prior tokens.

Without caching, this requires recomputing K/V for the entire sequence at every step - $O(n)$ cost per step, $O(n^2)$ total. The KV cache stores computed K/V pairs so only the new token's attention needs computation - $O(1)$ incremental cost.

Trade-off: KV cache memory grows linearly with sequence length and batch size, becoming a major GPU memory bottleneck for long sequences.

Techniques like Grouped Query Attention (GQA) and Multi-Query Attention (MQA) reduce KV cache size by sharing key/value heads across query heads.

Q: How are LLMs evaluated? What are the key benchmarks and evaluation approaches?

A: LLM evaluation uses multiple approaches:

(1) **Automated benchmarks** - MMLU (57-subject knowledge test), HumanEval/MBPP (code generation), GSM8K/MATH (math reasoning), HellaSwag/ARC (commonsense), BIG-Bench, HELM

(2) **LLM-as-judge** - a strong LLM (e.g., GPT-4) evaluates open-ended response quality on MT-Bench and AlpacaEval

(3) **Human preference evaluation** - side-by-side comparisons in Chatbot Arena (LMSYS)

(4) **Task-specific metrics** - ROUGE for summarization, BLEU for translation, Exact Match/F1 for QA, pass@k for code

(5) **Safety/alignment benchmarks** - TruthfulQA (hallucination), BBQ (bias), red-teaming datasets. No single benchmark captures full capability - always evaluate across multiple dimensions relevant to your use case.

Q: What is hallucination in LLMs and how can it be mitigated?

A: Hallucination refers to LLMs generating content that is factually incorrect, fabricated, or unsupported by source documents, yet presented with confident fluency.

Types:

(1) **Intrinsic hallucination** - output contradicts the provided source document

(2) **Extrinsic hallucination** - output cannot be verified from the source (may or may not be correct).

Causes: training data biases, optimization for fluency over factual accuracy, knowledge cutoff limitations.

Mitigation: RAG (grounding outputs in retrieved verified documents), RLHF with accuracy-focused rewards, uncertainty quantification (model expresses doubt when unsure), factual

consistency checking, constrained decoding, chain-of-thought prompting for multi-step reasoning, and citation-based generation where the model must cite supporting passages.

Q: What is DPO (Direct Preference Optimization) and how does it differ from RLHF?

A: DPO is a simpler, more stable alternative to PPO-based RLHF for aligning LLMs with human preferences. Traditional RLHF requires three stages: supervised fine-tuning (SFT), training a separate reward model on preference pairs, then PPO-based reinforcement learning.

DPO eliminates the reward model entirely - it directly optimizes the language model on preference data (chosen vs. rejected response pairs) using a single binary cross-entropy objective derived from the Bradley-Terry preference model. Advantages: simpler pipeline, more training stability, lower compute cost, no reward model needed. DPO is now widely used in open-source models including Llama 3-Instruct, Mistral Instruct, and Zephyr.

Limitation: DPO is an offline method and can be less effective than online RL for tasks requiring active exploration.

Q: What is quantization in LLMs, and what are the main techniques?

A: Quantization reduces the numerical precision of model weights and/or activations to lower memory requirements and speed up inference.

Key formats: FP32 (full precision, 4 bytes/param), BF16/FP16 (16-bit, standard for GPU inference), INT8 (8-bit, ~2x memory reduction, minimal quality loss), INT4 (4-bit, ~4x reduction, used for consumer hardware).

Key techniques:

(1) **GPTQ** - post-training quantization designed specifically for LLMs using Hessian-based weight compression

(2) **AWQ (Activation-Aware Weight Quantization)** - protects salient weights from quantization

(3) **QLoRA** - combines 4-bit quantization with LoRA for efficient fine-tuning on consumer GPUs

(4) **GGUF/llama.cpp** - quantized format for CPU/GPU inference on consumer hardware.

Trade-off: lower bit-width reduces memory but risks accuracy degradation, especially on smaller models and knowledge-intensive tasks.

Q: What are multimodal LLMs (Vision-Language Models) and how do they work?

A: Multimodal LLMs process and reason over multiple modalities - text, images, audio, and video. Architecture: a vision encoder (e.g., CLIP, SigLIP, ViT) converts images into embedding vectors that are projected into the same token space as text, allowing the LLM decoder to reason over both.

Examples: GPT-4o (text+image+audio), Claude 3 (text+image), Gemini 1.5 Pro (text+image+video+audio), LLaVA, PaliGemma, InternVL.

Key capabilities: image captioning, visual Q&A, document understanding (charts, tables, PDFs), OCR, and image-based reasoning. Training typically follows two stages:

(1) pre-train the vision-language projector with a frozen LLM backbone

(2) instruction fine-tune the full model on multimodal task datasets.

Q: How do you handle documents or conversations exceeding an LLM context window?

A: Several strategies exist:

(1) **RAG** - chunk the document, embed chunks in a vector store, retrieve only the most relevant chunks at query time

(2) **Summarization chains** - use map-reduce or refine patterns to progressively summarize large documents before passing to LLM

(3) **Sliding window** - process text in overlapping chunks

(4) **Long-context models** - Gemini 1.5 Pro (1M tokens) and Claude 3.5 (200K tokens) can process many documents natively

(5) **Hierarchical processing** - create multi-level summaries. Important consideration: the 'Lost in the Middle' problem - research shows LLMs attend more strongly to information at the beginning and end of long contexts, performing worse on information in the middle. RAG is often preferred for cost efficiency and precision, while long-context models are better for tasks requiring holistic understanding of large documents.

Research Papers

Important research papers.

2025

1. [DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning](#) - DeepSeek introduces its first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, trained solely through reinforcement learning (RL), exhibits strong reasoning abilities but faces challenges with readability and language mixing. To address these issues, DeepSeek-R1 incorporates multi-stage training and cold-start data, achieving reasoning performance comparable to OpenAI-o1-1217. DeepSeek has open-sourced both models along with six distilled versions (1.5B–70B) based on Qwen and Llama.
2. <https://ai.meta.com/research/publications/brain-to-text-decoding-a-non-invasive-approach-via-typing/> - Brain2Qwerty, a deep learning model, decodes sentences from brain activity using EEG or MEG while participants type. In a study of 35 volunteers, MEG achieved a 32% character-error-rate (CER), outperforming EEG (67%). Top participants reached 19% CER, with some sentences perfectly decoded. Errors suggest both motor and cognitive influences. This non-invasive approach brings brain-computer interfaces closer to aiding non-communicating patients safely.
3. [The Future of Software Engineering in an AI-Driven World](#) - AI-driven software development is transforming engineering, with LLMs enhancing productivity. This trend will grow, fostering a human-AI partnership. The research community must tackle key challenges in AI integration. This paper outlines our vision and the critical issues to address for the future of AI-driven software engineering.

2024

1. [Mixtral of Experts](#) - Mixtral 8x7B is a Sparse Mixture of Experts (SMoE) model based on Mistral 7B, featuring 8 feedforward experts per layer, with two selected per token. Each token accesses 47B parameters but uses only 13B during inference. Trained with a 32k context, Mixtral outperforms Llama 2 70B and GPT-3.5, excelling in math, coding, and multilingual tasks. The fine-tuned Mixtral 8x7B-Instruct surpasses GPT-3.5 Turbo, Claude-2.1, Gemini Pro, and Llama 2 70B-chat. Both models are released under Apache 2.0.
2. [DeepSeek-V3 Technical Report](#) - DeepSeek-V3 is a 671B-parameter Mixture-of-Experts (MoE) model with 37B active per token, optimized for efficiency using Multi-head Latent Attention

(MLA) and DeepSeekMoE. It introduces an auxiliary-loss-free strategy for load balancing and a multi-token prediction objective for improved performance. Trained on 14.8T tokens with SFT and RL, it rivals top closed-source models while requiring only 2.788M H800 GPU hours. Its training was highly stable, with no irrecoverable loss spikes or rollbacks.

3. [The Llama 3 Herd of Models \(Meta, 2024\)](#) - This paper introduces Llama 3, a new set of foundation models supporting multilinguality, coding, reasoning, and tool usage. The largest model, with 405B parameters and a 128K token context window, shows comparable performance to GPT-4 across various tasks. Llama 3 models, including Llama Guard 3 for safety, are publicly released. Additionally, the paper explores integrating image, video, and speech capabilities, demonstrating competitive results, though these models are still in development.
4. [NeedleBench: Can LLMs Do Retrieval and Reasoning in 1 Million Context Window? \(Mo Li et al., 2024\)](#) - This paper introduces NeedleBench, a framework for evaluating the long-context capabilities of large language models (LLMs) across various text lengths and depths. It assesses how well models retrieve and reason with critical information in bilingual long texts. The Ancestral Trace Challenge (ATC) is also proposed to test LLMs on complex logical reasoning. Results indicate that current LLMs have significant room for improvement in handling real-world long-context tasks.
5. [PaliGemma: A versatile 3B VLM for transfer \(Lucas Beyer et al., 2024\)](#) - PaliGemma is an open Vision-Language Model (VLM) that is based on the SigLIP-So400m vision encoder and the Gemma-2B language model. It is trained to be a versatile and broadly knowledgeable base model that is effective to transfer. It achieves strong performance on a wide variety of open-world tasks. We evaluate PaliGemma on almost 40 diverse tasks including standard VLM benchmarks, but also more specialized tasks such as remote-sensing and segmentation.
6. [Internet of Agents: Weaving a Web of Heterogeneous Agents for Collaborative Intelligence \(Weize Chen et al., 2024\)](#) - This paper introduces the Internet of Agents (IoA), a new framework for LLM-based multi-agent collaboration. IoA overcomes the limitations of existing frameworks by offering flexible agent integration, dynamic teaming, and scalable communication, inspired by the Internet's architecture. Experiments show IoA outperforms current baselines in various tasks, enabling seamless collaboration among diverse agents for enhanced intelligence and capabilities.

2023

1. [GPT-4 Technical Report \(OpenAI, 2023\)](#) - Detailed the capabilities and limitations of the GPT-4 language model.
2. [LLaMA: Open and Efficient Foundation Language Models \(Touvron et al., 2023\)](#) - Introduced a collection of open foundation language models with high performance and efficiency.
3. [PaLM 2 Technical Report \(Anil et al., 2023\)](#) - Described the architecture and performance of Google's PaLM 2 language model.
4. [Reinforcement Learning with Human Feedback: Learning Dynamic Choices via Pessimism \(Zihao Li et al., 2023\)](#) - This research paper explores offline Reinforcement Learning with Human Feedback (RLHF) to understand human rewards and optimal policies in a Markov Decision Process (MDP).

2022

1. [DALL·E 2: Hierarchical Text-Conditional Image Generation with CLIP Latents \(Ramesh et al., 2022\)](#) - Improved upon the original DALL-E model with higher resolution and more accurate text-to-image generation.
2. [InstructGPT: Training language models to follow instructions with human feedback \(Ouyang et al., 2022\)](#) - Described the process of fine-tuning language models to better follow human instructions.

2020

1. [Language Models are Few-Shot Learners \(Brown et al., 2020\)](#) - Introduced GPT-3, a large language model with few-shot learning capabilities.

2017

1. [Attention Is All You Need \(Vaswani et al., 2017\)](#) - Introduced the Transformer architecture, which revolutionized natural language processing and foundation for current Large Language Models (GPT, Claude, Gemini, Llama etc).