Interview Questions

Q1. What is tokenization, and why is it important in LLMs? :

Tokenization is the process of splitting text into smaller units called tokens, which can be words, sub words, or even characters. For instance, the word "tokenization" might be broken down into smaller subwords like "token" and "ization." This step is crucial because LLMs do not understand raw text directly. Instead, they process sequences of numbers that represent these tokens.

Effective tokenization allows models to handle various languages, manage rare words, and reduce the vocabulary size, which improves both efficiency and performance.

Q2. What is LoRA and QLoRA?

LoRA and QLoRA are techniques designed to optimize the fine-tuning of Large Language Models (LLMs), focusing on reducing memory usage and enhancing efficiency without compromising performance in Natural Language Processing (NLP) tasks.

=> LoRA (Low-Rank Adaptation):
LoRA is a parameter-efficient fine-tuning method that introduces new trainable parameters to modify a model's behavior without increasing its overall size. By doing so, LoRA maintains the original parameter count, reducing the memory overhead typically associated with training large models. It works by adding low-rank matrix adaptations to the model's existing layers, allowing for significant performance improvements while keeping resource consumption in check.

This makes it ideal for environments where computational resourcesare limited, yet high model accuracy is still required.

=> QLoRA (Quantized LoRA):
QLoRA builds on LoRA by incorporating quantization to further optimize memory usage. It uses techniques such as 4-bit Normal Float, Double Quantization, and Paged Optimizers to compress the model's parameters and improve computational efficiency. By reducing the precision of model weights (e.g., from 16-bit to 4-bit)while retaining most of the model's accuracy, QLoRA allows for the fine-tuning of LLMs with minimal memory footprint. This method is particularly useful when scaling large models, as it maintains performance levels comparable to full-precision models while significantly reducing resource consumption.

Q3. What is beam search, and how does it differ from greedy decoding?

Beam search is a search algorithm used during text generation to find the most likely sequence of words. Instead of choosing the single highest-probability word at each step (as greedy decoding does), beam search explores multiple possible sequences in parallel, maintaining a set of the top k candidates (beams). It balances between finding high-probability sequences and exploring alternative paths. This leads to more coherent and contextually appropriate outputs, especially in long-form text generation tasks.

Greedy decoding:

At each step, pick the word with the highest probability.
Fast, but often suboptimal (because a locally best choice may not lead to the globally best sentence).

Beam Search:

Maintains a "beam width" k (e.g., 3, 5, 10).
At each decoding step:

1. Expand all current candidate sequences by one token.
2. Compute cumulative log-probabilities of each extended sequence.
3. Keep only the top $k$ sequences.

Continue until end-of-sequence (EOS) is reached or max length is exceeded.

Advantages:
Produces better results than greedy decoding.
Balances exploration (trying different continuations) and exploitation (keeping only the most promising ones).

Limitations:
Can still miss diverse or creative outputs (tends to produce "safe" and generic completions).
Larger beam width → more computation, but not always better quality (too large beams can favor overly common phrases).

Example:
Suppose a model generates the next word after "I love":
Probabilities: "you" = 0.4 ; "pizza" = 0.3 ; "dogs" = 0.2 ; "running" = 0.1

-> Greedy decoding (beam size = 1): picks "you" → sequence = "I love you".

-> Beam search (beam size = 2):
- Keeps both "you" (0.4) and "pizza" (0.3).
- Expands them in the next step.
- If "I love pizza" has a higher overall sequence probability than "I love you very much", beam search will return "I love pizza" instead of the greedy "I love you".

Q4. Explain the concept of temperature in LLM text generation.

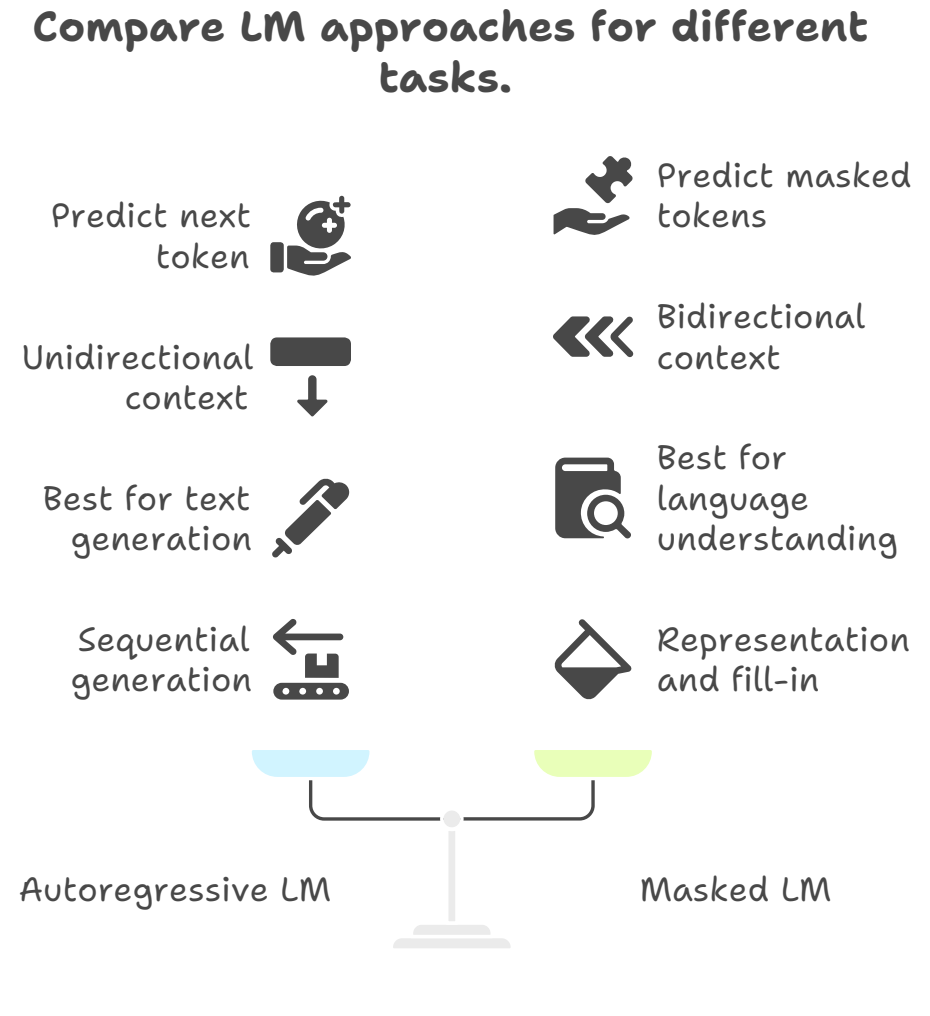
Temperature is a hyperparameter that controls the randomness of text generation by adjusting the probability distribution over possible next tokens. A low temperature (close to 0) makes the model highly deterministic, favoring the most probable tokens. Conversely, a high temperature (above 1) encourages more diversity by flattening the distribution, allowing less probable tokens to be selected. For instance, a temperature of 0.7 strikes a balance between creativity and coherence, making it suitable for generating diverse but sensible outputs

Q5. What is masked language modeling, and how does it contribute to model pretraining?

Masked language modeling (MLM) is a training objective where some tokens in the input are randomly masked, and the model is tasked with predicting them based on context. This forces the model to learn contextual relationships between words, enhancing its ability to understand language semantics. MLM is commonly used in models like BERT, which are pretrained using this objective to develop a deep understanding of language before fine-tuning on specific tasks.

Q6. How do autoregressive models differ from masked models in LLM training?

Autoregressive models, such as GPT, generate text one token ata time, with each token predicted based on the previously generated tokens. This sequential approach is ideal for tasks like text generation. Masked models, like BERT, predict randomly masked tokens within a sentence, leveraging both left and right context. Autoregressive models excel in generative tasks, while masked models are better suited for understanding and classification tasks.


Compare LM approaches for different tasks.

Q7. What role do embeddings play in LLMs, and how are they initialized?

Embeddings are dense, continuous vector representations of tokens, capturing semantic and syntactic information. They map discrete tokens (words or subwords) into a high-dimensional space, making them suitable for input into neural networks. Embeddings are typically initialized randomly or with pretrained vectors likeWord2Vec or GloVe. During training, these embeddings are fine-tuned to capture task-specific nuances, enhancing the model's performance on various language tasks.

Q8. What is next sentence prediction and how is useful in language modelling?

Next Sentence Prediction (NSP) is a key technique used in language modeling, particularly in training large models like BERT(Bidirectional Encoder Representations from Transformers). NSP helps a model understand the relationship between two sentences, which is important for tasks like question answering, dialogue generation, and information retrieval.

During pre-training, the model is fed two sentences:
50% of the time, the second sentence is the actual next sentence in the document(positive pairs).
50% of the time, the second sentence is a random sentence from the corpus(negative pairs).

The model is trained to classify whether the second sentence is the correct next sentence or not. This binary classification task is used alongside a masked language modeling task to improve the model's overall language understanding.

Q9. Explain the difference between top-k sampling and nucleus(top-p) sampling in LLMs.

Top-k sampling restricts the model's choices to the top k most probable tokens at each step, introducing controlled randomness. For example, setting k=10 means the model will only consider the 10most likely tokens. Nucleus sampling, or top-p sampling, takes amore dynamic approach by selecting tokens whose cumulative probability exceeds a threshold p (e.g., 0.9). This allows for flexible candidate sets based on context, promoting both diversity and coherence in generated text.

Q10. How can catastrophic forgetting be mitigated in large language models (LLMs)?

Happens when an LLM is fine-tuned on new data/tasks and forgets previously learned knowledge.
Critical in continual learning and domain adaptation.

Mitigating Catastrophic Forgetting in LLMs:

Regularization-based methods:
- EWC (Elastic Weight Consolidation) → penalizes changes to important weights. ☑ Retains old knowledge; ✗ computationally heavy.
- L2 / KL regularization → keeps fine-tuned weights close to original. ☑ Simple; ✗ may limit adaptation.

Replay-based methods:
- Experience replay → mix old data with new. ☑ Strong retention; ✗ storage & privacy concerns.
- Pseudo-rehearsal / Knowledge distillation → use synthetic data/logits from old model. ☑ No raw data needed; ✗ quality depends on teacher model.

Parameter isolation methods:
- Adapters / LoRA → add task-specific modules. ☑ Efficient & modular; ✗ parameter growth per task.
- Progressive networks → freeze old weights, add new ones. ☑ Prevents forgetting; ✗ large memory cost.

Multi-task & Continual Learning:
- Joint multi-task training → train on multiple domains/tasks together. ☑ Good generalization; ✗ requires simultaneous data.
- Curriculum learning → gradually introduce tasks. ☑ Smooth adaptation; ✗ task ordering is tricky.

Prompting & retrieval-based approaches:
- Prompt tuning / prefix-tuning / P-tuning → adapt via lightweight prompts. ☑ No weight overwrite; ✗ limited flexibility.
- Retrieval-Augmented Generation (RAG) → fetch knowledge externally. ☑ Avoids catastrophic forgetting; ✗ needs external KB & retrieval system.

Q11. What is model distillation, and how is it applied to LLMs?

Definition: Model distillation (knowledge distillation) is a process where a large model (teacher) transfers its knowledge to a smaller model (student). The aim is to reduce size, inference cost, and latency while keeping most of the performance.

How it works:

1. Train a large teacher model.
2. Use the teacher to produce "soft targets" (probability distributions over outputs).
3. Train a smaller student model to mimic the teacher's outputs.
4. The loss combines:

- • Cross-entropy with the true labels
- • KL divergence between teacher and student predictions

Applications in LLMs:
- • Compression → shrink huge models for faster inference
- • Deployment → make models efficient for edge/mobile use
- • Domain adaptation → distill domain-specific knowledge into smaller models
- • Ensemble distillation → compress multiple large teachers into one compact student

Examples:
- • DistilBERT retains \~97% of BERT's performance while being 40% smaller and 60% faster.
- • In modern LLM pipelines, distillation is often combined with LoRA/adapters for efficient specialization.

In short: Model distillation = teaching a smaller student to mimic a big teacher. In LLMs, it's applied for compression, efficiency, specialization, and deployment.

Q12. How do LLMs handle out-of-vocabulary (OOV) words?

Out-of-vocabulary words refer to words that the model did not encounter during training. LLMs address this issue through subword tokenization techniques like Byte-Pair Encoding (BPE) and Word Piece. These methods break down OOV words into smaller, known subword units. For example, the word "unhappiness" might be tokenized as "un," "happi," and "ness." This allows the model to understand and generate words it has never seen before by leveraging these subword components

Byte Pair Encoding (BPE)
Definition**: BPE is a subword tokenization algorithm that compresses text into smaller units (subwords) by repeatedly merging the most frequent pairs of characters or symbols.

Process:

1. Start with a sequence of characters (initial vocabulary = all characters).
2. Count frequency of all adjacent pairs.
3. Merge the most frequent pair into a new symbol.
4. Repeat until vocabulary reaches desired size.

Why it's used in LLMs:
- • Handles rare words by splitting them into subwords (e.g., "unhappiness" → "un", "happiness").
- • Reduces out-of-vocabulary (OOV) problems.
- • Balances between word-level and character-level tokenization.

Example:
Corpus: "low", "lowest", "newer", "wider"
- • Start: {l, o, w, e, s, t, n, ...}
- • Merge frequent pair "lo" → new token "lo"
- • Merge frequent pair "low" → new token "low"
- • Continue until vocab has desired size.

In practice:
- • GPT and many LLMs use BPE or variants (like SentencePiece, Unigram LM).
- • Enables efficient handling of large vocabularies.

In short:
BPE = iterative merging of frequent character pairs to form subwords
→ efficient, flexible tokenization for LLMs.

– Toy Example: Corpus: `"low lower new newer widest"`
1. Initial vocabulary (characters only): { l, o, w, e, r, n, i, d, s, t }

Words split into characters:
(l o w), (l o w e r), (n e w), (n e w e r), (w i d e s t)

2. Step 1: Count frequent pairs

(l, o) → 2; (o, w) → 2; (n, e) → 2; (e, w) → 2
(w, e) → 2; (e, r) → 2
others → 1

Most frequent pair: (l, o)
→ Merge into new token `lo`

3. Step 2: Update words
lo w
lo w e r
n e w
n e w e r
w i d e s t

4. Step 3: Count again

(lo, w) → 2
(n, e) → 2
(e, w) → 2
(w, e) → 2
(e, r) → 2

5. Step 4: Update words
low
low e r
n e w
n e w e r
w i d e s t

6. **Step 5: Continue merging**

Merge (n, e) → `ne`
Merge (ne, w) → `new`
Merge (w, e) → `we`
Merge (we, r) → `wer`
Merge (low, e) → `lowe`
Most frequent pair:
(lo, w)→ Merge into new token `low`
Final vocabulary(post few merges):
{ low, new, we, er, lo, ne, w, i, d, s, t }

☑ With more merges, the model learns frequent subwords like "low", "new", "er", etc. Rare words are split into smaller chunks, ensuring no word is truly "unknown."

Q13. How does the Transformer architecture overcome the challenges faced by traditional Sequence-to-Sequence models?

The Transformer architecture overcomes key limitations of traditional Seq2Seq models in several ways:

Parallelization: Seq2Seq models process sequentially, slowing training. Transformers use self-attention to process tokens in parallel, speeding up both training and inference.

Long-Range Dependencies: Seq2Seq models struggle with long-range dependencies. Transformers capture these effectively with self-attention, allowing the model to focus on any part of the sequence, regardless of distance

Positional Encoding: Since Transformers process the entire sequence at once, positional encoding is used to ensure the model understands token order.

Efficiency and Scalability: Seq2Seq models are slower to scale due to sequential processing. Transformers, with their parallelism, scale better for large datasets and long sequences.

Context Bottleneck: Seq2Seq uses a single context vector, limiting information flow. Transformers let the decoder attend to all encoder outputs, improving context retention

Q13. What are Generative and Discriminative models?

In NLP, generative and discriminative models are two key types of models used for various tasks.

Generative models learn the underlying data distribution and generate new samples from it. They model the joint probability distribution of inputs and outputs, aiming to maximize the likelihood of the observed data. A common example is a language model, which predicts the next word in a sequence based on previous words.

Discriminative models focus on learning a decision boundary between different classes in the input-output space. They model the conditional probability of outputs given inputs, aiming to accurately classify new examples. An example is a sentiment analysis model, which classifies text as positive, negative, or neutral based on its content.

Q14. What are positional encodings in the context of large language models?

Positional encodings are essential in Large Language Models(LLMs) to address the inability of transformer architectures to capture sequence order. Since transformers process tokens simultaneously through self-attention, they are unaware of token order. Positional encodings provide the necessary information to help the model understand the sequence of words

Mechanism:

Additive Approach: Positional encodings are added to input word embeddings, merging static word representations with positional data.

Sinusoidal Function: Many LLMs, such as the GPT series, use trigonometric functions to generate these positional encodings.

Positional Encoding (Sinusoidal)

For position = *pos*, embedding dimension = *i*, total dimension = *d*:

* PE(pos, 2i) = sin(pos / 10000^(2i / d))
* PE(pos, 2i+1) = cos(pos / 10000^(2i / d))

Final input to Transformer

InputEmbedding(pos) = WordEmbedding(pos) + PositionalEncoding(pos)

Q15. What is Multi-head attention?

Multi-head attention is an enhancement of single-head attention, allowing a model to attend to information from different representation subspaces simultaneously, focusing on various positions in the data. Instead of using a single attention mechanism, multi-head attention projects the queries, keys, and values into multiple subspaces (denoted as h times) through distinct learned linear transformations

This process involves applying the attention function in parallel to each of these projected versions of the queries, keys, and values, which generates multiple output vectors. These outputs are then combined to produce the final dv-dimensional result. This approach improves the model's ability to capture more complex patterns and relationships in the data.

Q16. How is the dot product used in self-attention, and what are its implications for computational efficiency?

Cross-entropy loss measures the difference between the predicted probability distribution and the true distribution (one-hot encoding of the correct token). It is defined as:

$$Attention(\underset{\text{from}}{q}, \underset{\text{to}}{k}, v) = \overbrace{softmax\left(\frac{qk^T}{\underset{\text{vector dimensionality of K, V}}{\sqrt{d_k}}}\right)}^{\text{Attention weights}} v$$

Where d_k is the dimensionality of the key vectors. The dot product measures alignment between tokens, helping the model decide which tokens to focus on. While effective, the quadratic complexity(O(n^2)) in sequence length can be a challenge for long sequences, prompting the development of more efficient approximations.

Q17. Explain cross-entropy loss and why it is commonly used in language modeling.

In self-attention, the dot product is used to calculate the similarity between query (Q) and key (K) vectors. The attention scores are computed as:

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Where yi is the true label and ŷi is the predicted probability. Cross-entropy loss penalizes incorrect predictions more heavily, encouraging the model to output probabilities that are closer to 1 for the correct class. In language modeling, it ensures the model predicts the correct token in a sequence with high confidence.

Q18. Explain the concept of eigenvalues and eigenvectors in the context of matrix factorization for dimensionality reduction.

1. Intuition:
Suppose we have a square matrix A.
An eigenvector v of A is a non-zero vector such that multiplying A by v just stretches or shrinks v, but does not change its direction.

Equation: A v = λ v
where: v = eigenvector
λ = eigenvalue (the amount of stretching or shrinking)
So eigenvalues tell us how much the eigenvector is scaled when transformed by A.

2. Why are they useful in dimensionality reduction?

In dimensionality reduction, like **Principal Component Analysis (PCA)**:

1. We start with data points in high dimensions.
2. We compute the **covariance matrix Σ** of the data.
3. The eigenvectors of Σ give the **principal directions** (axes along which the data varies the most).
4. The eigenvalues tell us **how much variance lies in each eigenvector direction**.

Large eigenvalue → direction of high variance → important for capturing information.
Small eigenvalue → direction of low variance → can often be discarded.

3. Connection to Matrix Factorization

If X is the data matrix (rows = samples, columns = features):

The covariance matrix is -> Σ = (1/n) XT X
We perform eigen-decomposition -> Σ = V Λ VT

where:
V = matrix of eigenvectors (principal components)
Λ = diagonal matrix of eigenvalues (variances along components)

To reduce dimensions:

Keep only the top *k* eigenvectors corresponding to the largest eigenvalues.
Project data onto this lower-dimensional subspace.

4. Intuition

Think of eigenvectors as **the most informative directions in your data space**.
Eigenvalues tell you **how much information (variance) is in each direction**.
By keeping only the top ones, you compress the data while retaining most of its structure.

☑ In short:

Eigenvectors = new axes (directions of maximum variance).
Eigenvalues = importance of those axes.
Matrix factorization (like PCA) uses these to project high-dimensional data into a lower-dimensional subspace.

Q19. Word Hallucination in Language Models.

Word hallucination occurs when a language model generates incorrect, misleading, or fabricated content that is not present in the input data. This can be a significant issue in applications where accuracy and factual correctness are crucial.

? Causes of Word Hallucination
Training Data Bias: The model learns patterns from a large dataset, which may contain biases or inconsistencies.
Overgeneralization: The model tries to generate coherent responses by filling in missing information, leading to plausible but incorrect details.
Next-Token Prediction Limitation: Since LLMs predict the most likely next word, they may fabricate information to maintain fluency.
Lack of Context Retention: Long-range dependencies in text can be difficult for models, leading to incorrect responses when context is lost.
Reinforcement Learning Trade-offs: RLHF (Reinforcement Learning from Human Feedback) improves alignment but may introduce biases by favoring certain outputs.
Dataset Contamination: If the training data contains incorrect facts or synthetic text, the model may generate hallucinated outputs.

↗ How to Reduce Word Hallucination?
Use Retrieval-Augmented Generation (RAG): Instead of relying solely on pre-trained data, integrate real-time retrieval from verified knowledge bases (e.g., Wikipedia, internal databases).
Fact-Checking Mechanisms: Implement post-processing methods to verify model outputs against trusted sources.
Improve Training Data Quality: Curate datasets with verified and diverse sources to minimize biases and inaccuracies.
Fine-Tuning with Domain-Specific Data: Fine-tune models using industry-specific knowledge to reduce misinformation.
Confidence Scoring: Assign confidence scores to model outputs to help users assess reliability.
Human-in-the-Loop (HITL): Implement human feedback loops to correct model errors and refine responses.
Prompt Engineering: Use structured prompts to guide model responses and reduce ambiguity.
Use External Verification Models: Cross-check generated text with another AI model trained for verification tasks.

Q20. Comprehensive Evaluation of an LLM Model in a RAG Pipeline

Evaluating a Large Language Model (LLM) built using a Retrieval-Augmented Generation (RAG) pipeline involves assessing two core components:

Retrieval Performance: Ensuring that the correct documents are retrieved.
Generation Quality: Ensuring that the model produces factually accurate, coherent, and relevant responses.
▶ 1. Evaluating the Retrieval Component
The retrieval component is responsible for fetching relevant documents from a knowledge base. Its effectiveness is evaluated using:

◆ Key Metrics:
- • Recall@K: Measures whether the correct document appears in the top-K retrieved results.

Example: If a user asks about "heart disease symptoms" and the relevant document is in the top 3 retrieved documents, Recall@3 is high.
- • Precision@K: Evaluates how many of the retrieved documents are actually relevant.
- • MRR (Mean Reciprocal Rank): Determines how early the most relevant document appears in the ranked list.
- • NDCG (Normalized Discounted Cumulative Gain): Assigns a score based on the ranking and relevance of retrieved documents.

▶ 2. Evaluating the Generation Component
Once the relevant documents are retrieved, the LLM generates responses based on them. Evaluation metrics focus on:

◆ Quality Metrics:
- •BLEU (Bilingual Evaluation Understudy): Measures text similarity to reference responses (useful for structured responses).
- • ROUGE (Recall-Oriented Understudy for Gisting Evaluation): Evaluates overlap between generated text and ground truth.
- • ROUGE-N: Measures n-gram overlap.
- • ROUGE-L: Measures longest common subsequence overlap.
- • METEOR: Accounts for synonyms and stemming variations in generated text.
- • Faithfulness Score: Checks if the generated response correctly represents the retrieved facts.
- • Hallucination Rate: Measures the frequency of incorrect or fabricated information.

◆ Error Analysis:
Fact Verification: Comparing generated output with external knowledge sources.
Contradictions: Identifying if the response contradicts retrieved facts.
Bias Analysis: Checking for model-generated biases in responses.

▶ 3. Evaluating Speed & Latency
Performance evaluation is crucial to ensure real-time usability. Key factors include:

- • Response Time: Measuring the time taken from query input to final response.
- • Token Efficiency: Ensuring the model does not generate unnecessary text.
- • Compute Cost: Measuring the resource consumption of retrieval and generation.

▶ 4. Human-in-the-Loop (HITL) Evaluation
While automated metrics are useful, human evaluation remains essential for subjective assessments:

- • Expert Review: Domain experts verify factual correctness.
- • User Feedback: Collecting ratings from real users.
- • A/B Testing: Comparing different model versions.

▶ 5. Debugging & Continuous Improvement
- • Failure Case Analysis: Studying incorrect outputs to refine retrieval and generation.
- • Bias Detection: Ensuring the model remains neutral and unbiased.
- • Edge Case Testing: Checking performance on ambiguous or adversarial queries.

▶ Conclusion
To evaluate an LLM model in a RAG pipeline effectively, a multi-layered approach combining retrieval accuracy, generation quality, latency, and human validation should be used. This ensures reliability, factual correctness, and efficiency in real-world applications.

Q21. What Happens If You Don't Use an Activation Function in a Neural Network?

Role of Activation Functions
Activation functions introduce non-linearity into a neural network, allowing it to learn complex patterns. Without activation functions, a neural network behaves like a simple linear function.

▶ Consequences of Not Using an Activation Function
1️⃣ The Entire Neural Network Becomes a Linear Model
If we remove activation functions from all layers, the output of each layer is just a weighted sum of the previous layer:

Output = W2 * (W1 * X) = W_final * X
This collapses the network into a single-layer linear model, no matter how many layers it has.

2️⃣ Inability to Learn Complex Patterns
Neural networks with activation functions can model non-linear relationships. Without them, the network can only learn linear mappings, which limits its usefulness for real-world problems like image recognition and NLP.

3️⃣ No Representation Power in Deep Networks
A deep network without activation functions is equivalent to a single-layer perceptron, making the depth meaningless.

4️⃣ No Universal Approximation Capability
With activation functions, neural networks can approximate any function (Universal Approximation Theorem). Without them, this capability is lost.

▶ Example Comparison

With Activation Function	Without Activation Function
Can model non-linear decision boundaries..	Can only model linear relationships.
Capable of deep learning and hierarchical feature learning.	Becomes equivalent to a linear regression model.
Used in complex tasks like image classification and NLP..	Fails at complex tasks due to lack of non-linearity.

☑ Conclusion
Activation functions are crucial for deep learning. Without them, neural networks lose their ability to learn and model complex patterns, effectively reducing them to simple linear models.

Q22. Dimensionality Reduction: PCA and LDA.

What is Dimensionality Reduction?
Dimensionality reduction is the process of reducing the number of input variables (features) while preserving important information. This helps in:

Reducing computational complexity
Eliminating redundant features
Improving model performance and interpretability
Handling the "curse of dimensionality"
1️⃣. Principal Component Analysis (PCA)
PCA is an unsupervised technique used to find new axes (principal components) that maximize variance in the data.

--> How PCA Works:
Computes the covariance matrix of the data.
Finds eigenvectors and eigenvalues of the covariance matrix.
Selects the top k eigenvectors (principal components) with the highest variance.
Projects data onto these new axes.
--> Example Use Cases:
Image compression (reducing pixel dimensions)
Feature extraction for machine learning models
Noise reduction in datasets
--> Key Points:
PCA is used for feature extraction, not classification.
Works best when features are correlated.
Does not consider class labels (unsupervised).

2️⃣ Linear Discriminant Analysis (LDA)
LDA is a supervised technique used to find a lower-dimensional space that best separates different classes.

--> How LDA Works:
Computes the mean of each class.
Finds the within-class and between-class scatter matrices.
Solves the eigenvalue problem to maximize class separability.
Projects data onto the new feature space.
--> Example Use Cases:
Facial recognition
Spam classification
Medical diagnosis
--> Key Points:
LDA is used for classification, unlike PCA.
Works best when classes are well-separated.
Uses class labels (supervised).
☑ Comparison Table

Feature	PCA (Principal Component Analysis).	LDA (Linear Discriminant Analysis)
Type	Unsupervised	Supervised
Objective	Maximizes variance	Maximizes class separability
Works on	Feature extraction	Classification problems
Uses class labels?	No	Yes

☑ Conclusion
PCA is used for dimensionality reduction without considering class labels, while LDA is used for reducing dimensions while maximizing class separability.

Q23. Docker and Its Role in Deployment.

-> What is Docker?
Docker is an open-source platform that enables developers to build, package, and distribute applications using containers. Containers ensure that applications run consistently across different environments.

-> Key Features of Docker:
Containerization: Packages applications and dependencies together.
Portability: Runs consistently across various operating systems and cloud providers.
Scalability: Easily deploys multiple instances for load balancing.
Isolation: Ensures applications do not interfere with each other.

-> Why is Docker Used in Deployment?
Docker simplifies the deployment process by ensuring that applications work the same way in different environments.

-> Key Benefits:
Eliminates "It Works on My Machine" Issues: Containers package everything, making deployments consistent.
Faster Deployment: Containers start quickly compared to virtual machines.
Efficient Resource Utilization: Uses fewer resources than traditional VMs.
Easy Rollbacks: Can revert to previous container versions easily.
Cloud & CI/CD Integration: Seamlessly integrates with AWS, Azure, Kubernetes, and CI/CD pipelines.

-> Docker in Deployment Workflow
Developer writes code and creates a Dockerfile (blueprint for the container).
Builds a Docker image using docker build.
Pushes the image to a container registry (Docker Hub, AWS ECR, etc.).
Deploys the container using docker run or an orchestration tool like Kubernetes.

-> Example: Simple Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.9

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container
COPY . .

# Install dependencies
RUN pip install -r requirements.txt

# Run the application
CMD ["python", "app.py"]
```

☑ Conclusion
Docker simplifies deployment by packaging applications and dependencies into isolated containers. This ensures consistency, portability, and scalability in software development and cloud environments.

Q24. TF-IDF, Word Embeddings, and BM25 Encoder

1️⃣ TF-IDF (Term Frequency-Inverse Document Frequency)
TF-IDF is a statistical method used to evaluate the importance of a word in a document relative to a collection (corpus). It helps in text retrieval and feature extraction.

◆ How TF-IDF Works:
Term Frequency (TF): Measures how often a word appears in a document.
Inverse Document Frequency (IDF): Reduces the weight of common words and boosts rare words.
Formula: TF-IDF = TF * IDF
◆ Example:
If "machine" appears 3 times in a document of 100 words, its TF is 3/100 = 0.03. If "machine" appears in 10 out of 1000 documents, its IDF is:

IDF = log(1000/10) = 2
TF-IDF score = 0.03 * 2 = 0.06

◆ Limitations of TF-IDF:
Does not capture word meaning or relationships.
Sensitive to long documents.
Works well for keyword-based retrieval but not for contextual understanding.

2️⃣ Word Embedding Techniques
Word embeddings represent words as dense numerical vectors, capturing semantic meanings and relationships.

◆ Popular Word Embedding Models:
- Word2Vec: Uses Skip-gram & CBOW models to learn word relationships.
- GloVe: Captures co-occurrence patterns across a large corpus.
- FastText: Handles out-of-vocabulary (OOV) words by using subword information.
- BERT Embeddings: Contextualized embeddings that consider the surrounding words.

◆ Example of Word2Vec:
If "king" is similar to "queen", Word2Vec can learn relationships like:

Vector("king") - Vector("man") + Vector("woman") ≈ Vector("queen")
◆ Advantages:
Captures semantic meaning.
Works well for NLP tasks like sentiment analysis, machine translation.
◆ Limitations:
Requires large training data.
Context-independent (except BERT and transformer-based embeddings).
3️⃣ BM25 Encoder
BM25 (Best Matching 25) is an advanced ranking function used in information retrieval. It improves upon TF-IDF by considering term saturation and document length.

◆ How BM25 Works:
Modifies TF-IDF by normalizing term frequency.
Uses a free parameter k1 to control term saturation.
Applies b to adjust for document length.
◆ BM25 Formula:
BM25 = IDF * [(TF * (k1 + 1)) / (TF + k1 * (1 - b + b * (doc length / avg doc length)))]

◆ BM25 vs. TF-IDF

Feature.	TF-IDF.	BM25
Scoring.	Simple multiplication of TF & IDF.	Adjusts for term saturation & document length
Normalization.	No normalization.	Length normalization improves ranking
Use Case.	Basic keyword search.	More accurate document retrieval

☑ When to Use BM25?
When ranking search results in a retrieval system.
For document ranking in a search engine.
When document length varies significantly.
☑ Conclusion
- TF-IDF is a simple text-weighting technique used for keyword-based retrieval.
- Word Embeddings capture deep semantic relationships and are useful for deep learning models.
- BM25 is an advanced ranking function that improves upon TF-IDF for search engines and document retrieval.

Q25. Evaluation Metrics for NLP Models

1️⃣ BLEU (Bilingual Evaluation Understudy)
BLEU measures n-gram precision with a brevity penalty.

Example Calculation:

Reference: "The cat sits on the mat."
Predicted: "The cat is sitting in the mat."
Unigram Matches: 4/7 = 0.57
Bigram Matches: 2/6 = 0.33
Final BLEU Score: 0.43 (43%)

2️⃣ ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
ROUGE measures recall, checking how much of the reference text appears in the generated text.

ROUGE-1 (Unigram Recall): 4/6 = 0.67
ROUGE-2 (Bigram Recall): 2/5 = 0.4
Final ROUGE Scores: ROUGE-1: 0.67, ROUGE-2: 0.4

3️⃣ METEOR (Metric for Evaluation of Translation with Explicit ORdering)
METEOR improves BLEU by considering stemming, synonyms, and word order.

Final METEOR Score: 0.55 (55%)

4️⃣ Perplexity (PPL)
Perplexity measures how well a language model predicts a sequence of words. Lower is better.

Formula:

PPL = exp(- (1/N) * Σ log P(wi))

Example Score: PPL = 50 (lower is better)

5️⃣ BERTScore
BERTScore compares embeddings using cosine similarity.

Final BERTScore: 0.89 (89%)

6️⃣ Frechet Inception Distance (FID)
FID compares text embeddings to measure realism.

Lower FID = Better generated text.

FID Score: 4.5

Final Summary Table

Metric.	Measures.	Best Score.	Example Value
BLEU	N-gram precision	High.	0.43
ROUGE	Recall-based match	High.	0.67
METEOR	Semantic similarity	High.	0.55
Perplexity (PPL)	Language model confidence	Low.	50
BERTScore	Meaning similarity	High.	0.89
FID	Text realism	Low.	4.5

☑ Key Takeaways
Use BLEU for translation but supplement it with METEOR.
ROUGE is great for summarization.
BERTScore is best for semantic similarity.
Perplexity is key for evaluating fluency.
FID helps in generative text evaluation.

Made with Napkin