Interview Questions
Gen AI

1) What is attention dropout in Transformer models? Why is it important during training and how does it affect generalization?

Answer: Attention dropout is a regularization technique applied to the attention weights in Transformer models. During training, a portion of the attention values is randomly set to zero. This forces the model to not overly rely on specific tokens and promotes robustness. It helps prevent overfitting, especially in large models, by encouraging the model to learn distributed representations. Without attention dropout, models may memorize patterns leading to poor generalization on unseen data.

---

2) Compare LoRA (Low-Rank Adaptation) and Prefix Tuning. In what scenarios would one outperform the other?

Answer: LoRA introduces low-rank learnable matrices into attention layers, enabling efficient fine-tuning with fewer parameters. Prefix Tuning, on the other hand, prepends learnable vectors (prefixes) to the input sequence, influencing the model's behavior via soft prompts.

- **LoRA is ideal** for applications requiring high precision and where access to model internals (weights) is available.
- **Prefix Tuning is better** when fine-tuning access is limited and soft control via prompts is sufficient.

LoRA generally outperforms Prefix Tuning in terms of convergence and quality, but Prefix Tuning offers easier integration with API-only models.

---

3) What is the difference between cosine similarity and dot product in vector search? Why does the choice matter in RAG systems?

Answer:

- **Dot product** measures raw magnitude and directional alignment.
- **Cosine similarity** normalizes vectors to remove the impact of magnitude, focusing solely on direction.

In RAG (Retrieval-Augmented Generation) systems:

- Use **cosine similarity** when embedding vectors may have inconsistent magnitudes (e.g., across domains).
- Use **dot product** when embeddings are uniformly scaled (like BERT or dense passage retrieval embeddings) and speed is a priority.

Choice affects relevance ranking in retrieval and ultimately the quality of generation.

---

4). How do large models like GPT-4 handle context length beyond training limits using attention mechanisms like ALiBi or Position Interpolation?

Answer: ALiBi (Attention with Linear Biases) introduces fixed linear positional biases in attention scores, allowing extrapolation to longer sequences.

Position Interpolation scales learned positional embeddings non-linearly to extend context during inference.

These techniques allow pretrained models with limited context windows (e.g., 2K tokens) to generalize to longer contexts (e.g., 16K+ tokens) without full retraining, enabling long-form reasoning and document summarization tasks.

---

5). Explain Mixture of Experts (MoE) in LLMs. What are the benefits and challenges of using sparse activation in production models?

Answer: MoE involves splitting a model into multiple sub-models (experts), only activating a subset during each forward pass. This allows scaling to billions of parameters with lower computational cost.

Benefits:

- Lower inference cost per token
- Larger capacity models with fewer FLOPs

Challenges:

- Routing inefficiencies and imbalance
- Load balancing across GPUs
- Increased engineering complexity

Used in Switch Transformer, GShard, and DeepMind's GLaM.

---

6) How would you design a multi-tenant LLM API platform for enterprise clients to ensure security, isolation, and performance?

Answer: Key design components:

- **Authentication & Authorization:** Role-based access using OAuth or JWT
- **Request Isolation:** Use namespaces per tenant; containerize model instances or sessions

- **Rate Limiting & Quotas:** Per-tenant limits to avoid noisy neighbors
- **Monitoring:** Per-tenant logging and observability (e.g., Prometheus, Grafana)
- **Billing Hooks:** Usage-based cost tracking
- **Model Customization:** Enable tenant-specific prompts or adapters

---

7) Describe the trade-offs between GPU inference and using optimized CPU runtimes like ONNX Runtime or Intel OpenVINO for LLM deployment.

Answer:

- **GPU Inference:**
  - Pros: High throughput, parallelization, ideal for large models.
  - Cons: Expensive, may require batching to amortize cost.
- **CPU Inference (ONNX/OpenVINO):**
  - Pros: Lower cost, simpler infra, good for edge/local deployments.
  - Cons: Lower throughput, unsuitable for models >1B parameters unless quantized.

Choose GPU for real-time, high-load applications; CPU for cost-effective, on-device, or small-scale tasks.

---

8) Explain how to implement a caching strategy for LLM outputs in a production pipeline to reduce latency and cost.

Answer: Use a combination of:

- **Prompt Hashing:** Hash prompts to use as keys
- **LRU Cache:** Store recent outputs for repeated queries
- **Chunk-level Caching:** Cache segments of responses for retrieval-based augmentation
- **Semantic Similarity Caching:** Use embeddings to reuse similar responses

Use Redis or in-memory caches. Helps with high-frequency Q&A and improves scalability.

---

9) How would you design a feedback loop system that lets human users improve the output of a generative AI model over time?

Answer: Key components:

- **UI for Feedback Collection**: Thumbs up/down, comments
- **Storage & Labeling**: Store inputs, outputs, feedback tags
- **Ranking Model (e.g., Reward Model)**: Fine-tune or reinforce model based on feedback

- **Retraining Pipeline**: Periodic updates based on labeled data
- **A/B Testing**: Validate changes before full deployment

---

10) What are the key considerations when using GenAI for multilingual support in a customer service chatbot?

Answer:

- **Language Coverage:** Ensure training data includes target languages
- **Code-Switching Support:** Handle mixed-language inputs
- **Translation Quality:** Balance between direct multilingual model use vs. translating to English before processing
- **Locale-specific Customization:** Cultural tone, idioms, formal/informal variants
- **Latency Considerations:** Real-time inference across multiple languages
- **Fallback Mechanisms:** When LLMs fail, integrate human-in-the-loop or canned responses

---

11) How do you implement knowledge distillation to compress a large LLM without losing much performance? Answer: Knowledge distillation involves training a smaller model (student) to replicate the behavior of a larger pre-trained model (teacher). In LLMs, this means feeding the same input to both models and minimizing the loss between their output distributions (typically logits). Key components include:

- **Teacher model**: A large LLM like GPT-3.
- **Student model**: A smaller transformer model.
- **Loss function**: Usually a combination of cross-entropy with the teacher's soft targets and hard labels.
- **Training**: Use temperature scaling and potentially intermediate layer supervision. Benefits include reduced inference time and memory usage. Challenges include ensuring that the student maintains reasoning and factual capabilities.

---

12) Can you walk through the challenges and solutions in integrating an LLM into a real-time production environment with low latency constraints? Answer: Key challenges include:

- **Inference speed**: Transformer-based models are computationally intensive.
- **Cold starts**: Dynamic scaling services may introduce latency.
- **Model size**: Large models strain memory and CPU/GPU budgets. **Solutions:**
- Use quantized or distilled models (e.g., int8, 4-bit quantization).
- Serve models using optimized runtimes (ONNX, TensorRT, vLLM).

- Deploy on GPU-enabled inference servers with autoscaling.
- Use caching strategies, approximate nearest neighbor (ANN) search, and prompt optimization to reduce calls.

---

13) How does the Mixture of Experts (MoE) model architecture optimize large-scale LLM inference? Answer: MoE introduces sparsity into LLMs by activating only a subset of the total model's "experts" (sub-networks) for each input. Benefits include:

- **Scalability**: Parameters scale with model size, but computational cost remains low.
- **Flexibility**: Route tokens to different experts based on input semantics.
- **Inference Efficiency**: Only 2–4 experts are active per forward pass, allowing massive models to be efficiently used. Key challenge: Load balancing and routing inefficiencies.

---

14) Describe your strategy for debugging and improving hallucinations in a production-grade RAG pipeline. Answer:

- **Content quality**: Improve the retrieval database with accurate, up-to-date documents.
- **Retriever tuning**: Use domain-specific embedding models.
- **Prompt design**: Explicitly condition the model to only answer based on retrieved docs.
- **Verification**: Include a post-response verification module or confidence score.
- **User feedback loop**: Collect and utilize feedback to refine retrieval and prompt policies.

---

15) How can you fine-tune a base model like LLaMA or Mistral on a specialized dataset using LoRA, and what are the trade-offs? Answer: LoRA (Low-Rank Adaptation) injects trainable low-rank matrices into specific layers of the transformer, enabling fine-tuning with fewer parameters. Steps:

- Load pre-trained LLaMA/Mistral.
- Insert LoRA modules into attention layers.
- Freeze base weights, train only LoRA layers.
- Merge LoRA into base weights post-training (optional). **Trade-offs:**
- **Pros**: Lower compute and memory cost, fast iteration.
- **Cons**: Might underfit highly domain-specific tasks compared to full fine-tuning.

---

16) In what situations would you use adapters or prompt-tuning instead of full fine-tuning of a large language model? Answer:

- **Adapters**: Ideal when updating large models for multiple tasks without altering the base model. Example: multi-task settings across departments.
- **Prompt-tuning**: Best for quick experimentation or when compute is limited. Use these approaches when:
- You need modularity.
- Limited computational resources.
- You want to maintain base model compatibility.

---

17) Explain token-level versus sentence-level perplexity and how these metrics impact GenAI evaluation. Answer:

- **Token-level perplexity**: Measures how well a model predicts the next token. Lower is better. Useful for language modeling.
- **Sentence-level perplexity**: Average perplexity across tokens in a sentence. More interpretable for sentence coherence. Implication: Lower perplexity doesn't always mean better quality—complement with metrics like BLEU, ROUGE, and human evaluation.

---

18) Describe a technique to make embeddings generated by transformer models more efficient and domain-specific. Answer:

- **Technique**: Domain-adaptive pretraining (DAPT) followed by contrastive learning.
- Use a domain corpus to continue pretraining.
- Apply contrastive loss to bring semantically similar samples closer in the embedding space.
- Use quantization or product quantization for efficient storage and ANN search.

---

19) How do you implement an RLHF (Reinforcement Learning with Human Feedback) pipeline practically and at scale? Answer:

- **Step 1**: Supervised fine-tune a base model using human-preferred responses.
- **Step 2**: Train a reward model from human ranking data.
- **Step 3**: Use PPO (Proximal Policy Optimization) to further fine-tune the LLM based on reward signals. Scaling tips:
- Use batch human feedback via labeling platforms.
- Optimize training loop using distributed training and checkpointing.
- Use scalable libraries like TRL (from Hugging Face).

---

20) How do you prevent prompt injection attacks in a deployed LLM-based application? Answer:

- **Sanitize inputs**: Remove suspicious patterns or meta-instructions.
- **Use prompt engineering**: Isolate system and user prompts with strict context windows.
- **Hard code boundaries**: Limit model access to external tools.
- **Use retrieval filters**: Prevent injection through vector store poisoning.
- **Post-processing**: Validate outputs using regex, LLM self-check, or human approval in critical apps.

---

21) What is a Bag-of-Words (BoW) model and what are its limitations in NLP? Answer: BoW represents text by the frequency of words, disregarding grammar and word order. It's simple and fast but fails to capture context, semantics, or word similarity. This often leads to sparse and high-dimensional vectors, making it ineffective for tasks needing word meaning or order.

---

22) How does TF-IDF improve upon Bag-of-Words? Answer: TF-IDF (Term Frequency-Inverse Document Frequency) weighs words based on how important they are to a document relative to a corpus. It reduces the influence of common words like "the" and highlights unique terms. However, it still ignores context and semantics, and suffers in synonym handling.

---

23)What are word embeddings and how do they address BoW limitations? Answer: Word embeddings like Word2Vec, GloVe, and FastText represent words in continuous vector space, capturing semantic similarity. Words with similar meanings have similar vectors. This enables context-aware similarity and downstream learning, but static embeddings still lack contextual understanding (e.g., “bank” in river vs finance).

---

24) Explain the architecture of Word2Vec and its training objective. Answer: Word2Vec has two main architectures: CBOW (predict center word from context) and Skip-gram (predict context from center word). It uses a shallow neural network to learn embeddings by maximizing the probability of observed word-context pairs. Negative sampling or hierarchical softmax aids efficient training.

---

25) What are the limitations of Word2Vec and GloVe? Answer: Both generate a single vector per word, regardless of context. Thus, polysemy (same word, different meanings) is not

captured. Also, they require pre-processing and large corpora. GloVe is matrix-factorization-based and has challenges with rare word handling.

---

26) How does FastText improve over Word2Vec? Answer: FastText represents words as a sum of character n-grams, enabling the model to handle rare words and capture subword information (e.g., prefixes/suffixes). This improves handling of out-of-vocabulary (OOV) words and better generalization for morphologically rich languages.

---

27) What is the architecture of an RNN, and how was it used in early NLP models? Answer: Recurrent Neural Networks (RNNs) process sequences by maintaining a hidden state updated with each input. They were used in sequence labeling, text generation, and translation. However, they suffer from vanishing gradients, making long-term dependencies hard to learn.

---

28) How did LSTM and GRU address RNN limitations? Answer: LSTMs and GRUs introduce gating mechanisms to control information flow, allowing them to remember longer-term dependencies. LSTM has input, output, and forget gates; GRU simplifies this with update and reset gates. They significantly improved performance in tasks like translation and sentiment analysis.

---

29) What are the key drawbacks of LSTM-based language models that led to the development of Transformers? Answer: LSTMs are sequential, limiting parallelism and leading to longer training times. They also struggle with very long dependencies and scalability. Transformers addressed this with self-attention mechanisms, allowing full context visibility and parallel training.

---

30) Compare CNNs and RNNs for text classification before Transformers. Answer: CNNs capture local n-gram features via convolutional filters and are more parallelizable than RNNs. RNNs capture sequential dependencies but are slower. CNNs often outperform RNNs in classification but underperform in generative or sequence prediction tasks. Neither captures long-range dependencies as effectively as Transformers.

---

31) What is the role of positional encoding in transformer-based models, and why is it necessary?

Answer:
Transformers do not have an inherent sense of word order since they process tokens in parallel (unlike RNNs). To incorporate the sequence order, **positional encodings** are added to the input embeddings. These encodings are either **learned** (as in BERT) or **sinusoidal** (as in the original Transformer paper). Sinusoidal encodings use sine and cosine functions of different frequencies to capture relative and absolute positions. Without positional encoding, the model would treat the input tokens as a bag of words, losing syntactic and sequential relationships crucial for understanding context.

---

32) How do masked language models (e.g., BERT) differ from autoregressive models (e.g., GPT) in terms of training objectives?

Answer:

- **Masked Language Models (MLMs)** like BERT are **bidirectional** and trained to predict randomly masked tokens within a sentence. For example, in the sentence "The cat sat on the [MASK]," the model learns to predict "mat" by attending to both left and right contexts.

- **Autoregressive models** like GPT are **unidirectional** (usually left-to-right) and trained to predict the **next token** in a sequence, e.g., "The cat sat on the" → "mat."

- MLMs are better for understanding tasks (e.g., classification, QA), while AR models are better for generation (e.g., story writing, dialogue).

---

33) Explain the concept of multi-head attention. Why is it superior to single-head attention?

Answer:
Multi-head attention allows the model to focus on different parts of a sequence simultaneously from multiple representation subspaces. It consists of multiple attention heads, each with its own learnable projection matrices for queries, keys, and values. This:

- Improves the model's ability to capture **different types of relationships** (e.g., syntactic, semantic).

- Increases **representational capacity** without increasing computation dramatically.

- Helps with **gradient flow** and **training stability**.
  Single-head attention would be limited to a single focus and may not capture complex

dependencies.

---

34) Describe the significance of layer normalization in stabilizing transformer training.

Answer:
Layer normalization normalizes the inputs across the features (as opposed to across the batch in batch normalization), stabilizing training by:

- Reducing internal covariate shift.
- Ensuring consistent distribution of activations.
- Improving gradient flow, especially in deep architectures like transformers.
  Transformers typically apply layer norm before or after each sublayer (e.g., attention or feed-forward), enabling efficient convergence during training.

---

35) What are subword tokenization methods (e.g., Byte Pair Encoding), and how do they impact model performance?

Answer:
Subword tokenization (e.g., **Byte Pair Encoding**, **WordPiece**, **UnigramLM**) splits rare or unknown words into smaller, frequent sub-units. For example, “unhappiness” → “un”, “happi”, “ness”.
Advantages:

- Reduces **out-of-vocabulary (OOV)** issues.
- Balances between word-level and character-level models.
- Leads to a **smaller and more efficient vocabulary**.
- Enhances handling of **morphologically rich languages**.
  Subword tokenization allows for **robust generalization**, particularly for unseen or compound words during inference.

---

36) Explain the idea of contextual embeddings. How are they different from static embeddings like Word2Vec?

Answer:
Static embeddings (e.g., Word2Vec, GloVe) assign a single vector to each word, regardless of context. This means “bank” in “river bank” and “bank account” gets the same embedding.

Contextual embeddings, as generated by models like ELMo, BERT, or GPT, assign word vectors that change based on surrounding words. For example:

- “She sat on the bank.” → different vector for “bank” than in
- “He deposited money in the bank.”

These embeddings capture **semantic and syntactic nuances**, improving performance in tasks like NER, QA, and coreference resolution.

---

37) What is the role of the feed-forward neural network layer in a transformer block?

Answer:
Each transformer block has a **position-wise feed-forward network (FFN)** after the multi-head attention. It consists of two linear transformations with a ReLU or GELU non-linearity in between:

```
FFN(x) = max(0, xW1 + b1)W2 + b2
```

Its role:

- Adds **non-linearity** and transformation capacity.
- Operates **independently on each position**, enriching token representations.
- Helps the model learn **compositional patterns** not captured by attention alone.

---

38) Compare cross-attention and self-attention mechanisms. Where is each used?

Answer:

- **Self-attention**: The queries, keys, and values all come from the same sequence. Used in both encoder and decoder to capture intra-sequence dependencies.

- **Cross-attention**: In decoders (e.g., in encoder-decoder models like T5 or BART), queries come from the decoder's output so far, and keys/values come from the encoder. This allows the decoder to attend to the encoder's output, useful for translation or summarization.

Use cases:

- **Self-attention**: Language modeling, classification.

- **Cross-attention**: Sequence transduction tasks like translation, summarization.

---

39) How does causal masking work in transformer decoders?

Answer:
Causal masking (or autoregressive masking) ensures that a token at position *i* can only attend to tokens at positions ≤ *i*. This is done by masking out future tokens in the attention matrix using a triangular mask. It enforces **left-to-right** generation, critical for autoregressive models like GPT. Without it, the model would "cheat" by looking at future tokens during training or inference.

---

40) What are the challenges and methods in scaling LLMs beyond 100B parameters?

Answer:
Challenges:

- **Memory constraints** on GPUs or TPUs.

- **Slower training and inference**.

- **Communication bottlenecks** in distributed setups.

- **Environmental cost** (energy and CO2 footprint).

- **Alignment and controllability** issues.

Methods:

- **Model parallelism**: Splitting weights across devices.
- **Pipeline parallelism**: Splitting layers sequentially across devices.
- **Mixture of Experts (MoE)**: Activating sparse subsets of weights per input.
- **Gradient checkpointing**: Reduces memory usage by re-computing intermediate activations.
- **Quantization & Pruning**: Reduces memory and compute usage.
- **Efficient pretraining datasets** (e.g., curated, deduplicated).
- **Specialized hardware** like H100s or custom LLM accelerators.

41) What is the difference between generative and discriminative models in classical NLP?

Answer:

- **Generative models** (e.g., Naive Bayes, HMMs) model the **joint probability** `P(X, Y)` and can generate data by sampling from the distribution.
- **Discriminative models** (e.g., Logistic Regression, Conditional Random Fields) model the **conditional probability** `P(Y | X)`, focusing on the decision boundary.
- Generative models are more flexible in modeling data distributions, while discriminative models generally perform better on prediction tasks.

---

42) Explain the concept and limitations of Hidden Markov Models (HMMs) in NLP.

Answer:
HMMs model sequences where the system is assumed to be a Markov process with hidden (unobserved) states.

- Used in POS tagging, NER, and speech recognition.
- Assumes:
    - The probability of the current state depends only on the previous state (Markov assumption).
    - The observed word depends only on the current hidden state.

Limitations:

- Assumes conditional independence of observations.
- Struggles with long-range dependencies.
- Requires manual feature engineering.

---

43) What are Conditional Random Fields (CRFs) and how do they improve upon HMMs?

Answer:
CRFs are **discriminative probabilistic models** used for **structured prediction** (e.g., sequence labeling).
Unlike HMMs:

- CRFs model `P(Y|X)` directly without making strong independence assumptions.
- Allow incorporation of arbitrary, overlapping features (e.g., prefixes, suffixes, word shapes).
- Provide better performance in tasks like POS tagging and NER.

---

44) Describe Pointwise Mutual Information (PMI) and its application in NLP.

Answer:
PMI measures the association between two events (e.g., word pairs):

PMI(x,y)=logP(x,y)P(x)P(y)\text{PMI}(x, y) = \log \frac{P(x, y)}{P(x) P(y)}PMI(x,y)=logP(x)P(y)P(x,y)

Applications:

- **Word similarity:** High PMI indicates strong semantic association.
- **Collocation detection:** Identifies meaningful word pairs (e.g., "New York").

Limitations:

- Biased toward infrequent word pairs.
- Can be unstable with sparse data.

---

45) How does the Viterbi algorithm work in sequence labeling tasks?

Answer:
The Viterbi algorithm is a **dynamic programming** method used to find the most probable sequence of hidden states in HMMs or CRFs.

Steps:

1. Initialize probabilities for each state at time t=0.
2. Recursively compute the most likely previous state for each state at time t.
3. Backtrack from the final state with the highest score to recover the full sequence.

Used in: POS tagging, speech recognition, bioinformatics.

---

46) What is a dependency grammar and how does dependency parsing work in classical NLP?

Answer:
Dependency grammar represents syntactic structure by linking words via **binary relations** (dependencies), where each word depends on a head.

Classical methods:

- **Transition-based parsers** (e.g., shift-reduce): Greedy or beam search-based decisions.
- **Graph-based parsers**: Find the highest scoring parse tree using MST algorithms.

Used for: syntactic analysis, semantic role labeling, relation extraction.

---

47) Explain Maximum Entropy (MaxEnt) models and their role in NLP.

Answer:
MaxEnt models (a.k.a. multinomial logistic regression) predict the probability distribution over outcomes that maximize entropy under given constraints.

Used in:

- POS tagging, NER, and text classification.
- Handles overlapping features well.

Advantages:

- Does not assume independence between features.
- Performs well with sparse but informative feature sets.

---

48) What is a confusion matrix, and how is it used in evaluating NLP classifiers?

Answer:
A **confusion matrix** is a tabular summary showing the performance of a classification model.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

From this, we calculate:

- **Precision** = TP / (TP + FP)
- **Recall** = TP / (TP + FN)
- **F1 Score** = 2 * (Precision * Recall) / (Precision + Recall)

Useful in NLP tasks like spam detection, sentiment analysis, and NER.

---

49) What are edit distance metrics, and how are they used in NLP?

Answer:
Edit distance measures how many changes are required to convert one string into another.

Types:

- **Levenshtein Distance:** Inserts, deletes, substitutes.
- **Jaccard Similarity:** Used for sets (e.g., word n-grams).

Applications:

- Spell correction.
- Fuzzy matching.
- Plagiarism detection.

- Record linkage.

---

50) What is the Zipf’s Law, and how does it apply to NLP corpus analysis?

Answer:
Zipf’s Law states that the frequency of a word is **inversely proportional to its rank** in a frequency list.

f(r)≈1rkf(r) \approx \frac{1}{r^k}f(r)≈rk1

Implications in NLP:

- Small number of words appear very frequently (e.g., "the", "is").
- Long tail of rare words — crucial for vocabulary management.
- Affects decisions on tokenization, stop word removal, and embedding coverage.