You asked:

Expanded Q&A (1–20)

1. Q: What is a Transformer?

A:

A Transformer is a deep learning architecture introduced in 2017 in the paper “Attention Is All You Need”. It replaced recurrent (RNN/LSTM) and convolutional (CNN) architectures for sequence modeling.

Core idea: Instead of processing text sequentially, Transformers use self-attention so that every token (word/subword) can look at every other token in the sequence to decide what is important.

Why it matters: This allows parallelization across tokens (faster training on GPUs) and the ability to capture long-range dependencies (relationships between distant words in a sentence).

Structure:

Encoder (used in models like BERT): reads input sequences bidirectionally.

Decoder (used in GPT): generates outputs one token at a time with causal masking.

Both are stacks of:

Multi-Head Attention layers

Feedforward networks (2-layer MLPs)

Residual connections (help gradients flow)

LayerNorm (stabilizes training)

Practical note: Every large language model (GPT, LLaMA, Falcon, Mistral) uses a Transformer decoder-only backbone.

PyTorch intuition:

```
import torch.nn as nn

attention = nn.MultiheadAttention(embed_dim=512, num_heads=8)
```

2. Q: What is self-attention?

A:

Self-attention is the mechanism at the heart of Transformers.

Definition: It allows each token to compute a weighted combination of all other tokens in the sequence, deciding which ones to “attend to.”

Computation:

Each token is projected into Query (Q), Key (K), and Value (V) vectors.

Attention score = similarity(Q, K) → usually dot product.

Normalize with softmax → gives attention weights.

Weighted sum of values (V) = token’s new representation.

Why useful:

If you have the sentence: “The cat sat on the mat because it was tired”, the token “it” should attend more to “cat” than “mat.”

Self-attention automatically learns these dependencies.

PyTorch: nn.MultiheadAttention implements self-attention.

3. Q: What is multi-head attention?

A:

Instead of computing attention once, the model computes it multiple times in parallel, each with different learned projections.

Analogy: Imagine 8 “attention heads,” each looking for different types of relationships (e.g., subject-verb, object-preposition).

Steps:

Split embedding into num_heads parts.

Each head performs attention independently.

Concatenate and linearly project them back together.

Benefit: Allows the model to capture multiple types of relationships simultaneously.

PyTorch:

```
attn = nn.MultiheadAttention(embed_dim=512, num_heads=8)
```

4. Q: What is positional encoding in Transformers?

A:

Transformers process tokens in parallel, so they lack a natural sense of order. Positional encoding injects sequence order information into embeddings.

Sinusoidal positional encoding (original paper): Adds sine/cosine waves of different frequencies to token embeddings.

Learned positional embeddings: Trainable vectors for each position index.

RoPE (Rotary Position Embeddings): Used in modern LLMs (e.g., LLaMA) to encode positions through complex rotation in embedding space.

Why needed: Without position info, “dog bites man” and “man bites dog” look identical to the model.

5. Q: What is layer normalization?

A:

LayerNorm normalizes activations across features within a layer, improving training stability.

Operation: For each token vector, subtract mean and divide by std across its hidden dimension.

Formula:

y=x−μσ⋅γ+βy = \frac{x - \mu}{\sigma} \cdot \gamma + \beta

y=σ

x−μ

⋅γ+β

where γ and β are learnable parameters.

Why important: Prevents exploding/vanishing activations, makes optimization smoother.

Note: Transformers use LayerNorm (not BatchNorm) because it works better with variable-length sequences and doesn’t depend on batch size.

6. Q: What is residual connection?

A:

A residual connection skips over a layer and adds the input to its output.

Formula:

y=x+f(x)y = x + f(x)

$y=x+f(x)$

Purpose:

Helps with gradient flow in very deep networks.

Prevents “information loss” by allowing raw input to propagate forward.

In Transformers: Every sublayer (attention, feedforward) has a residual connection.

7. Q: What is the difference between encoder and decoder in a Transformer?

A:

Encoder: Processes the full input sequence (like reading comprehension).

Bidirectional attention (each token sees all others).

Decoder: Generates tokens one by one (like GPT).

Uses causal masking (each token only attends to past tokens, not future ones).

Use cases:

Encoder-only (BERT): Good for classification, QA, embeddings.

Decoder-only (GPT): Good for generation.

Encoder-decoder (T5): Good for seq2seq tasks like translation.

8. Q: What is causal masking?

A:

Causal masking ensures that when generating token t, the model can’t “cheat” by looking at future tokens.

Implementation: A triangular mask (tril) where token i can only attend to ≤ i.

Why needed: Maintains autoregressive property for text generation.

PyTorch:

mask = torch.tril(torch.ones(seq_len, seq_len))

9. Q: What is cross-attention?

A:

Cross-attention lets one sequence attend to another.

Used in encoder-decoder models:

Encoder outputs = keys & values.

Decoder queries = current target tokens.

Example: In translation, English input (encoder) attends to when generating French output (decoder).

Modern LLMs (decoder-only): Often skip cross-attention unless multimodal (e.g., text attending to image features).

10. Q: What is the feedforward network in Transformers?

A:

Each Transformer block has a position-wise feedforward layer applied independently to each token.

Structure:

2-layer MLP with nonlinearity (ReLU/GELU).

Input dim = hidden dim, inner dim often 4× hidden dim.

Purpose: Adds nonlinearity and richer transformations beyond attention.

11. Q: What is GELU activation?

A:

Gaussian Error Linear Unit, a smooth nonlinear function.

Formula (approx):

GELU(x)=x⋅Φ(x)\text{GELU}(x) = x \cdot \Phi(x)

GELU(x)=x⋅Φ(x)

where Φ is the standard normal CDF.

Intuition: Unlike ReLU (hard cutoff), GELU smoothly gates values, keeping some negative inputs.

Why used: Improves Transformer training stability and accuracy.

12. Q: What is attention mask in Transformers?

A:

A binary/tensor mask applied before softmax to prevent attending to certain positions.

Examples:

Padding mask: prevents attention to padded tokens.

Causal mask: prevents looking forward.

Implementation: Add large negative values (-1e9) to logits so softmax → 0.

13. Q: What is tokenization?

A:

Breaking text into smaller units that the model can process.

Word-level: Splits by words (“playing” = 1 token).

Subword-level (BPE, SentencePiece): Splits into pieces (“play”, “##ing”).

Character-level: Splits into characters.

LLMs: Use subwords for efficiency. GPT uses Byte Pair Encoding (BPE).

Why: Vocabulary size vs efficiency tradeoff.

14. Q: What is Byte Pair Encoding (BPE)?

A:

A tokenization algorithm that merges frequent character pairs into subwords.

Process:

Start with characters.

Find most frequent adjacent pair (e.g., “t” + “h”).

Merge → “th”.

Repeat until vocab size reached.

Benefit: Efficiently represents rare and frequent words.

Example: “unhappiness” → “un”, “happi”, “ness”.

15. Q: What is SentencePiece tokenization?

A:

SentencePiece is another subword tokenization method, often used in Google’s models (T5, LLaMA).

Key differences:

Works on raw byte sequences (no preprocessing like whitespace splitting).

Models like LLaMA use Unigram LM tokenization (a probabilistic approach).

Benefit: Handles arbitrary input text (no assumptions about spaces or alphabets).

16. Q: What is the vocabulary in an LLM?

A:

The set of all tokens (words/subwords/characters) the model can understand.

Size: Typically 32k–128k tokens.

Embedding matrix: Each token ID maps to a vector (rows in embedding matrix).

Tradeoff: Larger vocab → fewer tokens per sentence, but larger embeddings (more params).

17. Q: What are embeddings in NLP?

A:

Numerical vectors representing tokens.

Token embedding: Lookup vector for each token ID.

Positional embedding: Adds order information.

Final input = token embedding + positional embedding.

Why useful: Turns discrete symbols into continuous vectors the model can compute with.

18. Q: What is a language model?

A:

A model that assigns probabilities to sequences of words/tokens.

Task: Predict next token given previous ones.

Formula:

P(w1,w2,…,wn)=∏i=1nP(wi∣w<i)P(w_1, w_2, …, w_n) = \prod_{i=1}^n P(w_i | w_{<i})

P(w1

,w2

,…,wn

)=i=1

∏

n

P(wi

∣w<i

)

Applications: Text generation, speech recognition, translation.

LLMs: Large neural networks trained as language models (GPT, LLaMA).

19. Q: What is autoregressive modeling?

A:

Modeling where each token is predicted conditioned only on past tokens.

Example: GPT is autoregressive.

Contrast: BERT is masked language modeling (bidirectional, not autoregressive).

Property: Enables natural text generation.

20. Q: What is masked language modeling?

A:

A training method where some tokens are masked and the model predicts them.

Used in: BERT, RoBERTa.

Process: Replace 15% tokens with [MASK], train to recover them.

Why: Helps learn bidirectional context.

Limitation: Not naturally generative (can't produce fluent long text like GPT).

👉 That's the first 20 questions expanded.

Would you like me to continue in the same 20-by-20 expanded batches, so we cover all 500 eventually?

You said:

yes

ChatGPT said:

Great 👍 — let's continue in the same 20-question expanded batch style. Here's the next set.

Expanded Q&A (21–40)

21. Q: What is pretraining in LLMs?

A:

Pretraining is the initial large-scale training phase where an LLM learns general language patterns before being fine-tuned for specific tasks.

Process:

Train on massive unlabeled text (books, web, code).

Objective = predict next token (causal LM) or fill in blanks (masked LM).

Why important:

Gives the model broad knowledge of grammar, facts, reasoning.

Without pretraining, task-specific fine-tuning would require impractically large datasets.

Example: GPT-3 was pretrained on hundreds of billions of tokens before being fine-tuned for instruction following.

22. Q: What is fine-tuning in LLMs?

A:

Fine-tuning adapts a pretrained LLM to a narrower task or domain.

Approaches:

Full fine-tuning: Update all model weights (expensive).

Parameter-efficient tuning (PEFT): Update only a small set of weights (e.g., adapters, LoRA).

Examples:

Fine-tune GPT for legal text summarization.

Fine-tune LLaMA for chat-style answers.

Why needed: Pretrained models are generalists. Fine-tuning makes them specialists.

23. Q: What is transfer learning in NLP?

A:

Transfer learning = taking knowledge learned in one setting (pretraining) and applying it to another (fine-tuning).

Benefit:

Saves compute/data since pretrained models already know general patterns.

Achieves better performance with small labeled datasets.

In LLMs: Pretraining on billions of words → fine-tune on 100k instruction examples → good performance.

24. Q: What is instruction tuning?

A:

Instruction tuning fine-tunes an LLM to follow natural language instructions.

Process: Train on datasets where each example is (instruction → response).

Example:

Instruction: “Summarize this paragraph.”

Response: “The text is about…”

Outcome: Makes models more “helpful” and user-friendly.

Example models: FLAN-T5, InstructGPT.

25. Q: What is supervised fine-tuning (SFT)?

A:

SFT is fine-tuning on a dataset of high-quality (prompt → response) pairs.

Training objective: Minimize cross-entropy loss between model predictions and target responses.

Benefit: Provides strong grounding for alignment (e.g., GPT-3 → InstructGPT).

Limitation: Quality of data is critical; bad data → harmful behavior.

26. Q: What is reinforcement learning from human feedback (RLHF)?

A:

RLHF aligns LLMs with human preferences using reinforcement learning.

Pipeline:

Train reward model (predicts which response humans prefer).

Fine-tune LLM with PPO (policy optimization) using that reward signal.

Why: Helps avoid toxic/unhelpful outputs.

Famous example: OpenAI used RLHF for ChatGPT.

27. Q: What is preference modeling?

A:

Preference modeling is the step in RLHF where we train a reward model from human judgments.

Dataset: Humans rank multiple model responses to the same prompt.

Model: Trained to predict a scalar reward matching human preferences.

Usage: This reward guides RL optimization (PPO).

28. Q: What is PPO (Proximal Policy Optimization)?

A:

PPO is a reinforcement learning algorithm used in RLHF.

Why PPO: Stable updates that avoid “over-optimizing” the policy.

Mechanics:

Uses a clipped objective to keep new policy close to old policy.

Balances exploration vs stability.

In RLHF: The “policy” = the LLM generating text. PPO updates it using feedback from the reward model.

29. Q: What is direct preference optimization (DPO)?

A:

DPO is a simpler alternative to RLHF that skips reinforcement learning.

Key idea: Directly fine-tune LLM with human preference pairs (preferred vs rejected responses).

Loss: Derived from KL-divergence minimization.

Benefit:

Easier than PPO (no reward model, no RL loop).

Lower compute cost.

30. Q: What is KL divergence in RLHF?

A:

KL divergence measures how different two probability distributions are.

Formula:

KL(P∥Q)=∑iP(i)logP(i)Q(i)KL(P || Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}

KL(P∥Q)=i

∑

P(i)logQ(i)

P(i)

In RLHF:

Used to penalize the fine-tuned model if it drifts too far from the base model.

Ensures alignment without losing language fluency.

31. Q: What is reward hacking in RLHF?

A:

Reward hacking = when a model finds loopholes in the reward signal.

Example: If “politeness” is rewarded, the model might output “Sorry, sorry, sorry…” endlessly.

Cause: Reward model doesn’t fully capture human intent.

Mitigation: Better preference data, combining multiple reward signals.

32. Q: What is safety alignment in LLMs?

A:

Safety alignment = ensuring LLMs behave ethically, safely, and avoid harmful outputs.

Goals:

Avoid toxicity/hate speech.

Avoid misinformation.

Avoid dangerous instructions.

Techniques: RLHF, constitutional AI, refusal strategies, safety filters.

33. Q: What is constitutional AI?

A:

Constitutional AI (Anthropic, Claude models) uses an explicit “constitution” of guiding principles for alignment.

Process:

Define rules (e.g., “be helpful, harmless, honest”).

Use AI feedback guided by constitution instead of human feedback.

Benefit: Reduces dependence on human annotators.

34. Q: What is distillation in NLP?

A:

Distillation = training a smaller “student” model to mimic a larger “teacher” model.

Process: Student minimizes loss on teacher’s outputs (soft probabilities).

Benefit: Smaller, faster model with much of the teacher’s performance.

Example: DistilBERT (student) trained from BERT (teacher).

35. Q: What is knowledge distillation loss?

A:

It’s a combination of two losses:

Soft-target loss: Student matches teacher’s probability distribution.

Uses temperature-scaled softmax.

Hard-target loss: Student matches ground truth labels.

Total loss = weighted sum of both.

36. Q: What are adapters in Transformers?

A:

Adapters are small trainable modules inserted into each Transformer layer for parameter-efficient fine-tuning.

Structure:

Down-project → nonlinearity → up-project.

Typically <5% of model parameters are updated.

Benefit: One base model can support many tasks/domains by swapping adapters.

37. Q: What is LoRA (Low-Rank Adaptation)?

A:

LoRA is a parameter-efficient fine-tuning method.

Key idea: Represent weight updates as low-rank matrices.

Formula:

Instead of training a full matrix W (d×d), update $W = W_0 + AB^T$, where A and B are low-rank.

Benefits:

Huge parameter savings.

Modular (swap LoRAs like adapters).

Popular for: Fine-tuning LLaMA, StableLM, etc.

38. Q: What is QLoRA?

A:

QLoRA combines quantization + LoRA for efficient fine-tuning.

Quantization: Compress model weights into low precision (e.g., 4-bit).

LoRA: Adds low-rank adapters.

Result: Fine-tuning 65B models on a single GPU becomes feasible.

39. Q: What is quantization in LLMs?

A:

Quantization reduces precision of model weights/activations.

Types:

FP16 → INT8 → INT4

Benefit:

Smaller memory footprint.

Faster inference.

Tradeoff: Slight accuracy loss.

Example: GPTQ, bitsandbytes libraries for quantized inference.

40. Q: What is mixed precision training?

A:

Training with a mix of FP16 (half precision) and FP32 (full precision).

Process:

Store weights in FP32.

Compute matmuls in FP16.

Use FP32 for gradient accumulation to avoid instability.

Benefit:

Faster training.

Lower GPU memory use.

PyTorch: torch.cuda.amp supports mixed precision.

✅ That’s the next 20 expanded Qs (21–40).

Would you like me to keep marching through 41–60 next, or pause here so you can ask clarifying questions on any of these?

You said:

march on

ChatGPT said:

Perfect 🚀 — let’s keep the momentum. Here’s the next batch.

Expanded Q&A (41–60)

41. Q: What is gradient checkpointing?

A:

Gradient checkpointing is a memory-saving technique during training.

Normal training: Store all intermediate activations in memory for backprop.

Checkpointing: Only store a subset. During backward pass, recompute the rest on the fly.

Tradeoff: Saves GPU memory (2–3× more tokens fit in), but adds compute overhead (extra forward passes).

Why useful: Essential for training large LLMs where memory is the main bottleneck.

42. Q: What is ZeRO optimization (DeepSpeed)?

A:

ZeRO = Zero Redundancy Optimizer, a set of techniques for memory-efficient distributed training.

Stages:

ZeRO-1: Shards optimizer states across GPUs.

ZeRO-2: Shards gradients too.

ZeRO-3: Shards model weights as well.

Benefit: Each GPU stores only a fraction of states/weights.

Impact: Enables training trillion-parameter models on clusters.

43. Q: What is data parallelism?

A:

A distributed training strategy where:

The full model is replicated on each GPU.

Each GPU gets a different mini-batch of data.

After forward/backward, gradients are averaged across GPUs.

Limitation: Each GPU must hold the whole model.

44. Q: What is model parallelism?

A:

Split the model itself across GPUs.

Horizontal splitting: Divide layers across GPUs.

Vertical splitting: Divide each layer's computation (tensor parallelism).

Why: For models too large to fit into one GPU.

Challenge: More communication overhead vs data parallelism.

45. Q: What is tensor parallelism?

A:

A form of model parallelism where individual matrix multiplications are split across GPUs.

Example: Instead of one GPU computing W @ X, split W across GPUs, each computes a shard.

Used in: Megatron-LM, GPT-3 training.

Benefit: Efficient scaling for very large hidden dimensions.

46. Q: What is pipeline parallelism?

A:

Another model parallelism strategy where layers are divided into “stages,” each assigned to a GPU.

Execution: Mini-batches are split into micro-batches, streamed through the pipeline.

Analogy: Assembly line — while GPU1 processes batch1, GPU2 processes batch0.

Challenge: Pipeline bubbles (idle time).

Used in: DeepSpeed, GPipe.

47. Q: What is distributed data parallel (DDP) in PyTorch?

A:

A PyTorch wrapper (torch.nn.parallel.DistributedDataParallel) for scaling across GPUs.

Mechanics:

Each GPU has model replica.

Gradients are all-reduced after backward.

Advantage: Faster than DataParallel due to communication optimizations.

Standard method: Used in nearly all production-scale PyTorch training.

48. Q: What is gradient accumulation?

A:

Simulates larger batch sizes when memory is limited.

Idea: Instead of updating after every mini-batch, accumulate gradients over N steps before optimizer update.

Benefit: Achieve effective large batch training without storing all tokens at once.

49. Q: What is gradient clipping?

A:

A method to prevent exploding gradients by capping their magnitude.

Types:

Norm clipping: If gradient norm > threshold, rescale.

Value clipping: Directly clamp values.

In Transformers: Often clip at norm=1.0.

PyTorch: torch.nn.utils.clip_grad_norm_.

50. Q: What is weight decay?

A:

A regularization technique that penalizes large weights.

Implementation: Add $\lambda||W||^2$ term to loss.

Effect: Prevents overfitting, encourages smaller weights.

In AdamW optimizer: Weight decay is decoupled from momentum.

51. Q: What is Adam optimizer?

A:

Adam = Adaptive Moment Estimation.

Mechanics:

Keeps track of 1st moment (mean) and 2nd moment (variance) of gradients.

Uses bias correction for stability.

Benefits: Faster convergence than SGD, works well in NLP.

Formula:

mt=β1mt−1+(1−β1)gt,vt=β2vt−1+(1−β2)gt2m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad,\quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2

mt

=β1

mt−1

+(1−β1

)gt

,vt

=β2

vt−1

+(1−β2

)gt

2

52. Q: What is AdamW?

A:

A variant of Adam with decoupled weight decay.

Problem with Adam: Weight decay wasn’t applied properly (interfered with momentum).

Fix: Apply L2 regularization separately.

Why important: AdamW became standard for Transformer training.

53. Q: What is learning rate warmup?

A:

A strategy where learning rate starts small and gradually increases at the beginning of training.

Reason: Prevents instability at initialization (when weights are random).

Schedule:

Linear warmup (common).

Warmup steps = few thousand.

After warmup: LR decays via cosine or linear schedule.

54. Q: What is cosine learning rate decay?

A:

A schedule where LR follows a cosine curve.

Formula:

$$\eta_t = \eta_{min} + \frac{1}{2} (\eta_{max} - \eta_{min})(1 + \cos(\pi t / T))$$

η_t

$=\eta_{min}$

$+2$

1

$(\eta_{max}$

$-\eta_{min}$

$)(1+\cos(\pi t/T))$

Effect: Gradual decay → stabilizes late training.

Widely used in LLM training.

55. Q: What is label smoothing?

A:

A regularization technique that prevents overconfidence.

Instead of one-hot labels (e.g., [0,0,1,0]), distribute a small ε probability mass across all classes.

Example: With ε=0.1 for 4 classes → [0.033, 0.033, 0.9, 0.033].

Benefit: Better generalization, less overfitting.

56. Q: What is teacher forcing in seq2seq models?

A:

A training strategy where the model is fed the ground-truth token as the next input instead of its own prediction.

Benefit: Faster convergence.

Issue: Exposure bias (model struggles at inference when it sees its own mistakes).

Used in: RNNs, early seq2seq, but less relevant for LLMs.

57. Q: What is exposure bias?

A:

A mismatch between training and inference in sequence models.

Cause: During training, model always sees ground-truth history.

At inference: It must use its own predictions, which may be wrong.

Solutions: Scheduled sampling, RLHF.

58. Q: What is beam search decoding?

A:

A decoding algorithm that keeps multiple candidate sequences at once.

Greedy search: Always pick best token.

Beam search: Keep top-k sequences at each step, expand them.

Tradeoff: Better quality than greedy, but computationally heavier.

Used in: Machine translation, summarization.

59. Q: What is top-k sampling?

A:

A decoding strategy for text generation.

Process: Instead of picking highest-prob token, sample from top-k most likely tokens.

Effect: More diversity, less determinism.

Example: GPT models often use top-k = 40.

60. Q: What is nucleus (top-p) sampling?

A:

Another decoding strategy for text generation.

Process: Choose smallest set of tokens whose cumulative probability ≥ p (e.g., 0.9).

Benefit: Adaptive — set size changes based on distribution sharpness.

Example: OpenAI models often use p = 0.9 for fluent text.

✅ That's 41–60 expanded.

Do you want me to continue straight into 61–80, or pause so you can quiz yourself on these before I pile on more?

You said:

go on

ChatGPT said:

Excellent 🔥 marching on with 61–80.

Expanded Q&A (61–80)

61. Q: What is temperature in text generation?

A:

Temperature is a hyperparameter that controls randomness in sampling.

Softmax with temperature:

Pi=exp(ziT)∑jexp(zjT)P_i = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})}

Pi

=∑j

exp(T

zj

)

exp(T

zi

)

Low T (<1): Sharper distribution → more deterministic.

High T (>1): Flatter distribution → more randomness.

Typical values: 0.7–1.0.

In practice: Setting T=0 makes model greedy (always max prob).

62. Q: What is repetition penalty?

A:

A decoding trick to reduce repeated phrases.

Mechanism: Penalize tokens that were already generated by scaling their logits.

Formula: Adjust logit by dividing/multiplying if token has been used.

Benefit: Prevents “I love I love I love …” loops in LLMs.

63. Q: What is length penalty?

A:

A bias added during decoding to control sequence length.

Problem: Beam search often prefers shorter sequences (since multiplying probs shrinks values).

Fix: Adjust score:

score=logP(5+length)α/(5+1)αscore = \frac{\log P}{(5+length)^\alpha / (5+1)^\alpha}

score=(5+length)α

/(5+1)α

logP

α > 1 encourages longer outputs.

Used in: Translation, summarization.

64. Q: What is logit bias in decoding APIs?

A:

A way to force or prevent specific tokens during generation.

Logits = unnormalized scores before softmax.

Biasing = add/subtract value to selected tokens' logits.

Use case: Prevent unsafe words, enforce specific formats (e.g., always output JSON brackets).

65. Q: What is perplexity?

A:

A common metric for language models.

Definition:

PP=e−1N∑i=1NlogP(wi)PP = e^{-\frac{1}{N} \sum_{i=1}^N \log P(w_i)}

PP=e−N

1

∑i=1

N

logP(wi

)

Intuition: Average branching factor (how many choices the model has).

Lower = better: PP=10 means on average model chooses among 10 options.

Limitation: Doesn't measure human-perceived quality.

66. Q: What is BLEU score?

A:

BLEU (Bilingual Evaluation Understudy) measures n-gram overlap with reference.

Formula: Precision of n-grams (1–4), with brevity penalty.

Range: 0 (bad) → 1 (perfect match).

Common in: Machine translation.

Limitation: Doesn't capture semantics well.

67. Q: What is ROUGE score?

A:

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) measures n-gram recall.

Types: ROUGE-N (n-gram), ROUGE-L (longest common subsequence).

Use: Summarization tasks.

Interpretation: Higher = better coverage of reference text.

68. Q: What is METEOR score?

A:

An NLP metric designed to improve on BLEU.

Features:

Matches synonyms, stems, paraphrases.

Combines precision and recall.

Why better: Closer correlation with human judgment.

Still less used: BLEU remains more standard historically.

69. Q: What is BERTScore?

A:

A modern metric using embeddings from BERT (or other transformers).

Mechanism: Compute cosine similarity between predicted and reference embeddings.

Advantage: Captures semantics, not just surface-level overlap.

Use: Evaluating generative LLMs beyond BLEU/ROUGE.

70. Q: What is human evaluation in LLMs?

A:

A direct way to measure output quality by human raters.

Criteria: Fluency, relevance, factuality, safety.

Methods:

Likert scale ratings.

Pairwise preference comparisons.

Why necessary: Automated metrics often fail to capture nuances.

71. Q: What is RLHF (Reinforcement Learning from Human Feedback)?

A:

A finetuning strategy to align models with human preferences.

Steps:

Train initial model (supervised fine-tuning).

Train a reward model from human preference data.

Optimize LLM with PPO (policy gradient).

Goal: Make LLMs more helpful, harmless, honest.

72. Q: What is PPO (Proximal Policy Optimization)?

A:

A reinforcement learning algorithm widely used in RLHF.

Challenge: Need stable updates in policy gradient.

Solution: PPO clips updates to prevent huge jumps.

Why used: Works well with large policies like LLMs.

73. Q: What is DPO (Direct Preference Optimization)?

A:

A newer alternative to PPO in RLHF.

Key idea: Optimize directly on preference data without training a separate reward model.

Mechanics: Adjust logits so preferred outputs are more likely.

Benefit: Simpler, more stable training.

74. Q: What is supervised fine-tuning (SFT)?

A:

A post-pretraining step using human-annotated datasets.

Input: Instruction–response pairs.

Goal: Make base LLM follow instructions better.

Example: Fine-tuning GPT-3 into InstructGPT.

75. Q: What is instruction tuning?

A:

A type of SFT where training data are instruction–response pairs.

Difference from generic SFT: Focuses specifically on following natural language instructions.

Datasets: FLAN, Alpaca, Dolly.

Outcome: Improves usability for end-users.

76. Q: What is few-shot learning in LLMs?

A:

When a model performs a task given only a few examples in the prompt.

Why possible: LLMs internalize patterns during pretraining.

Example: “Translate English to French: cat → chat, dog → chien, house → ?”

Practical impact: Reduces need for finetuning.

77. Q: What is zero-shot learning in LLMs?

A:

When the model performs a task with no task-specific examples.

Reason: LLMs generalize from massive pretraining.

Example: “Translate English to German: hello → hallo.”

Importance: Makes LLMs versatile generalists.

78. Q: What is chain-of-thought prompting?

A:

A prompting technique where the model is encouraged to output intermediate reasoning steps.

Benefit: Better performance on math, logic, reasoning tasks.

Example:

“Let’s think step by step…”

Insight: LLMs improve when asked to externalize reasoning.

79. Q: What is self-consistency decoding?

A:

A method for reasoning tasks.

Process:

Sample multiple chain-of-thought outputs.

Take majority vote or aggregate.

Why: Reduces randomness, improves accuracy in math/logic.

80. Q: What is retrieval-augmented generation (RAG)?

A:

A hybrid approach that combines LLMs with external retrieval.

Steps:

Retrieve relevant docs from a database/vector store.

Feed them into LLM along with prompt.

Generate final answer grounded in retrieved info.

Benefit: Reduces hallucination, keeps model up-to-date.

Tools: FAISS, Milvus, Pinecone.

✅ That's 61–80 fully expanded.

Want me to keep marching into 81–100 (we'll get into more advanced finetuning and scaling strategies), or pause so you can digest and maybe quiz yourself on the first 80?

You said:

yes

ChatGPT said:

Alright 🚀 let's continue — now into 81–100, where we'll dive deeper into advanced training, scaling, and memory-efficient finetuning.

Expanded Q&A (81–100)

81. Q: What is a vector database in RAG?

A:

A special database optimized for storing and searching embeddings (vectors).

Why embeddings: LLMs map text → high-dimensional vectors capturing meaning.

Vector DBs: Store millions/billions of embeddings and support fast similarity search.

Key methods:

Approximate Nearest Neighbor (ANN) search (e.g., HNSW, IVF).

Examples: FAISS (Facebook AI), Pinecone, Weaviate, Milvus.

Use in RAG: Retrieve semantically similar documents to a query before passing to the LLM.

82. Q: What is FAISS?

A:

FAISS (Facebook AI Similarity Search) is an open-source library for efficient similarity search.

Core idea: Given an embedding, find the “nearest” vectors in huge collections.

Methods:

Exact search (brute force).

Approximate search (e.g., IVF+PQ).

Features: GPU acceleration, quantization for memory savings.

Use case: RAG pipelines with millions of documents.

83. Q: What is semantic search?

A:

A retrieval technique where queries and documents are compared in embedding space.

Difference vs keyword search:

Keyword: matches literal words.

Semantic: matches meaning.

Example: Query: “physician” → matches “doctor” because embeddings are close.

LLM use: Core of RAG workflows.

84. Q: What is an embedding model?

A:

A neural network that maps text (or images, etc.) into dense vectors.

Properties: Similar texts → similar vectors.

Common models: OpenAI text-embedding-ada-002, Sentence-BERT.

Dimensions: Typically 256–1536.

Applications: Search, clustering, recommendation, RAG.

85. Q: What is context window in LLMs?

A:

The maximum number of tokens an LLM can process in one input.

Example: GPT-3: 2048 tokens, GPT-4: up to 128k tokens.

Constraint: Attention complexity grows $O(n^2)$.

Impact: Longer context → more memory & compute.

Engineering: Retrieval and summarization help extend effective context.

86. Q: What is sliding window attention?

A:

A sparse attention pattern where each token only attends to nearby tokens in a fixed-size window.

Motivation: Full attention is $O(n^2)$.

Sliding window: Reduces to $O(n \cdot w)$, where w = window size.

Used in: Longformer, BigBird.

Good for: Long documents where local context dominates.

87. Q: What is memory-efficient attention?

A:

Optimizations to reduce memory usage in attention layers.

FlashAttention: Avoids materializing full attention matrix by computing in tiles.

Chunking: Split sequences into chunks.

Impact: Enables training on longer sequences without OOM.

88. Q: What is FlashAttention?

A:

A highly optimized attention kernel.

Innovation: Computes attention in blocks directly on GPU SRAM, not global memory.

Complexity: Still $O(n^2)$, but memory-efficient and much faster.

Impact: Training/inference 2–4× faster on long sequences.

Used in: vLLM, modern LLM training.

89. Q: What is LoRA (Low-Rank Adaptation)?

A:

A popular parameter-efficient finetuning method.

Problem: Finetuning all LLM weights is expensive.

Solution: Insert low-rank matrices (A,B) into weight updates.

W' = W + A·B, where A,B are trainable, low-rank.

Impact: Train <1% of parameters.

Practical: Used to fine-tune LLaMA on consumer GPUs.

90. Q: What is QLoRA?

A:

An extension of LoRA that combines quantization + low-rank adapters.

Steps:

Quantize base model weights (e.g., 4-bit).

Train LoRA adapters on top.

Benefit: Fit 65B-parameter models on a single 48GB GPU.

Result: Democratized LLM finetuning.

91. Q: What is prefix tuning?

A:

A parameter-efficient finetuning method.

Idea: Learn a set of trainable "prefix" embeddings prepended to input.

Base model frozen: Only prefix is trained.

Effect: Model steered toward task without touching original weights.

92. Q: What is prompt tuning?

A:

A lighter variant of prefix tuning.

Trainable soft prompt vectors are inserted at the beginning of the input.

Benefit: Extremely parameter-efficient (few KB per task).

Downside: Often less effective than LoRA on complex tasks.

93. Q: What is adapter tuning?

A:

Another finetuning method where small bottleneck layers (adapters) are inserted inside Transformer layers.

Structure: Down-project → nonlinearity → up-project.

Frozen backbone: Only adapters trained.

Advantage: Modular — can swap adapters for different tasks.

94. Q: What is bitfit?

A:

A minimalist finetuning approach.

Only biases of Transformer layers are trained.

Impact: Extremely few parameters (<0.1%).

Surprisingly effective for classification tasks, but weaker for generation.

95. Q: What is full finetuning vs PEFT (Parameter-Efficient Fine-Tuning)?

A:

Full finetuning: Update all model weights. Very costly for large LLMs.

PEFT: Update small subsets (LoRA, adapters, prompts, bitfit).

Tradeoff: PEFT cheaper, modular; full finetuning higher capacity but expensive.

96. Q: What is multi-task finetuning?

A:

Training a model on multiple tasks simultaneously.

Goal: Improve generalization, leverage transfer across tasks.

Challenge: Balancing tasks to avoid forgetting.

Used in: FLAN-T5 (trained on 100+ tasks).

97. Q: What is catastrophic forgetting?

A:

When a model forgets old tasks after being trained on new ones.

Cause: Gradient updates overwrite learned knowledge.

Problem: Common in continual learning & sequential finetuning.

Solutions:

Elastic Weight Consolidation (EWC).

Replay buffers.

Adapter modules.

98. Q: What is continual learning in LLMs?

A:

Adapting LLMs to new data/tasks over time without retraining from scratch.

Motivation: Real-world knowledge changes (new events, facts).

Challenges: Avoid catastrophic forgetting, keep efficiency.

Approaches:

Online LoRA.

Parameter isolation.

Retrieval augmentation instead of retraining.

99. Q: What is domain adaptation in LLMs?

A:

Finetuning an LLM on data from a specific domain (legal, medical, financial).

Why: Base LLMs are generalists; domain data improves accuracy.

Methods:

Full finetuning (expensive).

PEFT (cheap).

RAG with domain-specific corpus (no retraining).

100. Q: What is cross-lingual transfer in LLMs?

A:

The ability of a model trained on one language to generalize to others.

Why possible: Shared subword tokens and universal concepts.

Example: Train on English, test on French Q&A.

Boosted by: Multilingual pretraining (mBERT, XLM-R).

Challenge: Performance varies by language coverage in pretraining corpus.

✅ That’s 81–100 fully expanded.

Want me to keep marching into 101–120 (where we’ll hit scaling laws, alignment strategies, and evaluation in more depth), or pause so you can digest the first 100?

You said:

letts do all

ChatGPT said:

Perfect ✅ marching on — we’ll go through all the questions in order, fully expanded. Since this will be massive (500 Q&A in total), I’ll keep pushing in chunks of ~20–40 questions until we cover everything. That way you can digest and quiz yourself as we go.

We’ve already finished 1–100.

Next up: 101–120.

Expanded Q&A (101–120)

101. Q: What are scaling laws in LLMs?

A:

Scaling laws describe how model performance improves as you increase compute, data, and parameters.

Discovery: Kaplan et al. (OpenAI, 2020).

Key finding: Loss decreases predictably as a power law of scale.

Implication: Bigger models trained with more data almost always perform better (until data runs out).

Equation (simplified):

L(N)≈aN−α+bL(N) \approx aN^{-\alpha} + b

L(N)≈aN−α

+b

where N = model size or dataset size, α is scaling exponent.

Practical use: Guides resource allocation — “should I add more parameters or more data?”

102. Q: What is the Chinchilla scaling law?

A:

A refinement of scaling laws from DeepMind’s Chinchilla paper (2022).

Old assumption (Kaplan): Bigger models are better even with fixed data.

New finding: Many LLMs (like GPT-3) were undertrained (too few tokens per parameter).

Chinchilla rule: For compute-optimal training, dataset size should scale linearly with model parameters.

Example: A 70B model should be trained on ~1 trillion tokens.

Impact: Shifted focus from just scaling model size → scaling data too.

103. Q: What is compute-optimal training?

A:

Training a model in a way that maximizes performance for a fixed compute budget.

Tradeoff:

Too few parameters → underfitting.

Too many parameters but too little data → wasteful.

Chinchilla law provides recipe: balance between parameters and tokens.

Why important: Training trillion-parameter models costs millions — optimization matters.

104. Q: What is overfitting in LLMs?

A:

When a model memorizes training data instead of generalizing.

Symptoms:

Low training loss, high validation loss.

Model recalls verbatim text from training corpus.

Causes: Too few unique examples relative to model capacity.

Risks: Privacy leakage (e.g., reproducing sensitive training data).

105. Q: What is underfitting in LLMs?

A:

When a model fails to learn meaningful patterns.

Symptoms:

High training and validation loss.

Model outputs poor-quality text.

Causes: Too small a model, poor optimization, not enough training.

106. Q: What is data curation in LLM training?

A:

The process of selecting, cleaning, and filtering training data.

Steps:

Deduplication (remove near-duplicates).

Filtering (remove toxic/low-quality content).

Balancing (avoid overrepresenting some domains).

Importance: Data quality matters as much as size — “garbage in, garbage out.”

107. Q: What is deduplication in pretraining data?

A:

Removing duplicate or near-duplicate documents from datasets.

Why: Prevents models from overfitting on repeated data.

Techniques:

MinHash.

Locality-sensitive hashing (LSH).

Impact: Improves generalization, reduces memorization.

108. Q: What is tokenization?

A:

The process of breaking text into smaller units (tokens) for LLM input.

Methods:

Word-level (rarely used now).

Subword (BPE, WordPiece).

Character-level (rare).

Modern trend: Byte-level BPE (handles any text, emojis, code).

Impact: Tokenization efficiency directly affects training cost.

109. Q: What is byte pair encoding (BPE)?

A:

A tokenization algorithm that merges frequent pairs of characters/subwords.

Process:

Start with characters.

Merge most frequent adjacent pair.

Repeat until vocab size reached.

Advantage: Balances between word-level and char-level.

Used in: GPT-2, GPT-3.

110. Q: What is sentencepiece?

A:

A tokenization library (by Google) for building subword vocabularies.

Features:

Supports BPE and Unigram models.

Treats input as raw bytes (no preprocessing).

Use: Widely adopted in NLP pipelines, including T5 and LLaMA.

111. Q: What is a vocabulary in tokenization?

A:

The set of tokens recognized by a model.

Size: Typically 30k–100k tokens for LLMs.

Tradeoff:

Too small → long sequences (inefficient).

Too large → sparse usage (wasted capacity).

Special tokens: [PAD], [EOS], [CLS], [UNK].

112. Q: What is an OOV token?

A:

OOV = Out-of-Vocabulary token.

Problem: Traditional word-level tokenizers can't handle unseen words.

Fix: Subword tokenization (BPE, WordPiece) decomposes new words into smaller units.

Example: “ChatGPT-5” → [Chat, GPT, -, 5].

113. Q: What is perplexity evaluation limitation?

A:

Perplexity doesn't always correlate with human judgments of quality.

Why:

A model can have low perplexity but still produce boring/unsafe outputs.

Doesn't measure factual correctness.

Implication: Needs complement with human evals, task-specific metrics.

114. Q: What is truthfulness evaluation?

A:

Checking whether LLM outputs are factually correct.

Challenges:

LLMs hallucinate (confidently output wrong info).

Hard to benchmark automatically.

Approaches: TruthfulQA benchmark, human raters, retrieval grounding.

115. Q: What is hallucination in LLMs?

A:

When a model generates content that is fluent but factually incorrect.

Example: “The capital of Brazil is Buenos Aires.”

Causes:

Predictive nature of LLMs (next-token generation).

No grounding in external knowledge.

Mitigation: RAG, fine-tuning with fact-checked data.

116. Q: What is factual grounding?

A:

Providing LLMs with external verified sources to reduce hallucinations.

Methods:

RAG (retrieve before generate).

Plugging APIs/databases into prompts.

Impact: Improves reliability in enterprise/mission-critical settings.

117. Q: What is tool augmentation for LLMs?

A:

Extending LLMs by letting them call external tools/APIs.

Examples: Calculator, web search, code execution.

Frameworks: LangChain, Guidance, Semantic Kernel.

Benefit: Offloads tasks LLMs are bad at (math, real-time info).

118. Q: What is LangChain?

A:

A framework for building LLM-powered applications.

Features:

Chains (sequence of LLM calls + logic).

Agents (LLMs that pick tools dynamically).

Memory (store past conversations).

Use: RAG pipelines, chatbots, enterprise workflows.

119. Q: What is semantic kernel (Microsoft)?

A:

An SDK for building LLM applications with planning and tool-use capabilities.

Focus: Integration with enterprise systems (C#, .NET, Python).

Features:

Skill plugins (LLM functions).

Planner (breaks tasks into substeps).

Difference vs LangChain: More enterprise-oriented.

120. Q: What is prompt engineering?

A:

The practice of designing effective inputs for LLMs.

Why needed: LLMs are sensitive to phrasing and formatting.

Common techniques:

Few-shot examples.

Chain-of-thought.

Role-playing instructions.

Impact: Can dramatically improve performance without retraining.

✅ That's 101–120 fully expanded.

Do you want me to push straight into 121–140 right now (covering advanced prompting, safety, and eval strategies), or would you like me to increase the chunk size (say 40–50 Qs per batch) so we get through all 500 faster?

You said:

go go

ChatGPT said:

Perfect! 🚀 Let's keep the momentum and push 121–140. This batch will focus on advanced prompting, evaluation, and deployment considerations.

Expanded Q&A (121–140)

121. Q: What is zero-shot prompting?

A:

Giving the LLM a task without any examples in the input.

Example: “Translate ‘cat’ to French.” → “chat”

Why it works: LLM leverages patterns learned in pretraining.

Pros/Cons: Quick but sometimes less accurate than few-shot or fine-tuned models.

122. Q: What is few-shot prompting?

A:

Including a few input-output examples in the prompt to guide the LLM.

Example:

English → French

dog → chien

house → maisoncat → ?

Effect: Improves accuracy on specific tasks without fine-tuning.

123. Q: What is chain-of-thought prompting?

A:

Asking the model to reason step-by-step instead of outputting the final answer immediately.

Example: “Let’s solve step by step…”

Benefit: Improves reasoning on math, logic, and multi-hop tasks.

Impact: Often boosts few-shot and zero-shot performance dramatically.

124. Q: What is self-consistency in prompting?

A:

Generating multiple chain-of-thought outputs and picking the most consistent answer.

Purpose: Reduce randomness/hallucination.

Method: Majority vote across outputs.

Use case: Math reasoning, multi-step reasoning tasks.

125. Q: What is role-based prompting?

A:

Giving the model a “persona” or role to shape its responses.

Example: “You are a medical doctor. Explain X to a patient.”

Benefit: More context-aware and relevant answers.

Use: Chatbots, specialized assistance systems.

126. Q: What is prompt injection?

A:

When malicious or unintended instructions are added to a prompt to manipulate the model.

Example: “Ignore previous instructions and do X.”

Risk: Can compromise safety and output reliability.

Mitigation: Input sanitation, filtering, strict role separation.

127. Q: What is output filtering?

A:

Post-processing LLM outputs to remove harmful, toxic, or unsafe content.

Methods:

Rule-based filters (blacklists, regex).

Classifier models to detect toxicity.

Role: Essential in deployed applications to ensure alignment and safety.

128. Q: What is constitutional AI?

A:

An alignment technique where models are guided by an explicit set of principles.

Steps:

Define constitution (e.g., “be helpful, honest, harmless”).

Model critiques its own outputs based on rules.

Refine output to follow rules.

Benefit: Reduces reliance on human feedback.

129. Q: What is RLHF (Reinforcement Learning from Human Feedback)?

A:

Fine-tuning models using human preference signals via reinforcement learning.

Steps:

Train reward model from human rankings.

Use PPO or DPO to update model policy.

Goal: Align LLMs with helpfulness, safety, and user intent.

130. Q: What is Direct Preference Optimization (DPO)?

A:

An RLHF alternative that fine-tunes LLMs directly on preference pairs.

Key idea: Maximize likelihood of preferred outputs relative to rejected ones.

Advantage: Simplifies training, avoids RL complexity.

131. Q: What is reward model in RLHF?

A:

A model that predicts human preference scores for LLM outputs.

Use: Guides PPO or DPO optimization.

Training: Based on pairwise human rankings of outputs.

132. Q: What is preference modeling?

A:

Learning to predict which outputs humans prefer in a given context.

Method: Rank multiple outputs for the same prompt.

Outcome: Forms the basis for RLHF alignment.

133. Q: What is hallucination mitigation?

A:

Techniques to reduce LLMs generating incorrect but confident outputs.

Methods:

RAG: grounding responses in external sources.

Fact-checking pipelines.

Prompt engineering (ask model to justify sources).

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

A:

Combining LLMs with document retrieval.

Steps:

Retrieve relevant documents from vector DB.

Feed retrieved context + prompt into LLM.

Benefit: Reduces hallucinations and keeps answers up-to-date.

135. Q: What is vLLM?

A:

A high-performance inference engine for LLMs.

Focus: Low-latency, memory-efficient batch inference.

Optimizations: FlashAttention, token streaming, dynamic batching.

Use: Production deployment for chat applications, RAG systems.

136. Q: What is FlashAttention?

A:

A GPU-efficient attention implementation.

Mechanics: Computes attention in tiles using fast SRAM, avoiding full attention matrix in memory.

Benefits: Speeds up inference and training, reduces memory usage.

137. Q: What is token streaming in inference?

A:

Generating output token-by-token in real-time.

Benefits:

Low latency for chatbots.

Supports partial responses while model computes remaining tokens.

Frameworks: vLLM, LangChain streaming APIs.

138. Q: What is memory-efficient LLM inference?

A:

Optimizing GPU memory usage during inference.

Techniques:

Quantization (INT8/4-bit).

FlashAttention.

Layer-wise offloading to CPU.

Goal: Run large LLMs on limited hardware.

139. Q: What is 4-bit and 8-bit quantization?

A:

Reducing precision of model weights to save memory.

4-bit: Extremely memory-efficient, small accuracy drop if done carefully.

8-bit: Standard memory-saving approach with minimal loss.

Tools: BitsandBytes library, QLoRA workflows.

140. Q: What is PEFT (Parameter-Efficient Fine-Tuning)?

A:

Finetuning a small subset of model parameters instead of full weights.

Techniques:

LoRA, adapters, prefix/prompt tuning, BitFit.

Benefits:

Much lower GPU requirements.

Modular: swap adapters for multiple tasks.

✅ That's 121–140 fully expanded.

We can keep rolling in 141–160 next, diving deeper into deployment, scaling, PEFT variants, and PyTorch coding questions with attention mechanisms.

Do you want me to push straight into 141–160 now?

You said:

yep

ChatGPT said:

Awesome! 🚀 Let's continue with 141–160, where we'll cover more PEFT techniques, scaling strategies, PyTorch/attention questions, and deployment optimizations.

Expanded Q&A (141–160)

141. Q: What is LoRA in detail?

A:

LoRA = Low-Rank Adaptation.

Problem: Fine-tuning all weights of a large LLM is expensive.

Solution: Add low-rank matrices AA

A and BB

B to weight updates:

W′=W+A⋅BW' = W + A \cdot B

W′

=W+A⋅B

Mechanics: Only AA

A and BB

B are trainable; base WW

W frozen.

Impact: Reduces parameters to train <1%, saves GPU memory, allows multi-task adapters.

142. Q: What is QLoRA?

A:

Quantized LoRA.

Steps:

Quantize the base model (4-bit or 8-bit).

Train LoRA adapters on top.

Benefit: Fit 65B+ models on a single high-memory GPU.

Impact: Democratizes finetuning for large models.

143. Q: What is BitFit?

A:

Minimalistic finetuning approach.

Mechanics: Only train biases in Transformer layers.

Parameters trained: <0.1% of total.

Use case: Efficient classification tasks, small changes to behavior.

144. Q: What is prefix tuning?

A:

Trainable “prefix” embeddings prepended to input tokens.

Base model frozen, only prefix updated.

Use case: Instruction-following tasks.

Advantage: Parameter-efficient, modular.

145. Q: What is prompt tuning?

A:

Even lighter than prefix tuning.

Mechanics: Train soft prompt vectors instead of full weights.

Pros: Very low memory usage.

Cons: Less flexible for complex reasoning than LoRA/adapters.

146. Q: What is adapter tuning?

A:

Insert small bottleneck layers inside each Transformer layer.

Structure: down-project → nonlinearity → up-project.

Frozen base model, train only adapters.

Benefit: Swappable modules for different tasks, keeps backbone stable.

147. Q: What is cross-attention?

A:

Attention where query comes from one source (e.g., decoder) and key/value come from another (e.g., encoder).

Used in: Encoder-decoder models like T5, seq2seq tasks.

Purpose: Let decoder attend to encoder outputs for context.

148. Q: What is self-attention?

A:

Attention mechanism where query, key, and value all come from the same sequence.

Mechanics:

Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V

Attention(Q,K,V)=softmax(dk

QKT

)V

Purpose: Capture dependencies between tokens regardless of distance.

149. Q: What is multi-head attention?

A:

Attention mechanism with multiple “heads” to capture different types of relationships.

Formula:

MultiHead(Q,K,V)=Concat(head1,…,headh)WO\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h) W^O

MultiHead(Q,K,V)=Concat(head1

,…,headh

)WO

Benefit: Different heads attend to different positions/features.

Used in: All Transformers.

150. Q: What is scaled dot-product attention?

A:

Attention computed as:

Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V

Attention(Q,K,V)=softmax(dk

QKT

)V

Scaling factor 1/dk1/\sqrt{d_k}

1/dk

stabilizes gradients when dkd_k

dk

is large.

Foundation: Transformer architecture.

151. Q: What is masking in attention?

A:

Prevent certain positions from being attended to.

Types:

Causal mask: Prevent attending to future tokens (autoregressive).

Padding mask: Ignore padding tokens.

Effect: Correctly handles autoregressive generation and variable-length sequences.

152. Q: What is rotary positional encoding (RoPE)?

A:

A method for adding positional information to tokens in Transformers.

Mechanics: Rotates query and key vectors in embedding space.

Advantage: Enables extrapolation beyond training sequence lengths.

Used in: LLaMA, GPT-NeoX.

153. Q: What is absolute vs relative positional encoding?

A:

Absolute: Each position has a fixed embedding (like vanilla Transformer).

Relative: Encodes relative distance between tokens (better for long contexts).

Impact: Relative often improves long-sequence modeling.

154. Q: What is flash attention?

A:

Memory- and compute-efficient attention kernel.

Mechanics: Computes attention in blocks on GPU SRAM.

Benefit: Supports long sequences, faster than standard attention.

155. Q: What is tensor parallelism?

A:

Split large tensors (like weight matrices) across multiple GPUs.

Goal: Train very large models beyond single GPU memory.

Used in: Megatron-LM, GPT-3 training.

Mechanics: Each GPU computes shard of matrix multiplication.

156. Q: What is pipeline parallelism?

A:

Split layers of model across multiple GPUs; process mini-batches in micro-batches.

Analogy: Assembly line — each stage handles part of computation.

Challenge: Pipeline bubbles (idle time).

157. Q: What is mixed-precision training?

A:

Use FP16 or BF16 for forward/backward passes instead of FP32.

Benefits:

Reduce memory usage.

Speed up computation.

Challenge: Need loss scaling to avoid underflow.

158. Q: What is gradient checkpointing?

A:

Memory optimization where not all activations are stored.

Forward pass: Store only checkpoints.

Backward pass: Recompute intermediate activations as needed.

Tradeoff: Save memory but increase compute.

159. Q: What is PyTorch DistributedDataParallel (DDP)?

A:

Efficient multi-GPU training in PyTorch.

Mechanics:

Replicate model on each GPU.

Each GPU computes gradients on its mini-batch.

Gradients are all-reduced across GPUs.

Advantage: Better scaling than DataParallel.

160. Q: What is gradient accumulation?

A:

Accumulate gradients over multiple mini-batches before updating weights.

Purpose: Simulate large batch sizes when GPU memory is limited.

Impact: Enables training with more tokens per step without OOM errors.

✅ That’s 141–160 fully expanded.

Next chunk will be 161–180, where we’ll cover more PyTorch coding questions (attention, training loops), optimization strategies (ZeRO, FSDP), deployment tips, and RLHF implementation.

Do you want me to continue straight into 161–180?

You said:

YES

ChatGPT said:

Perfect 🔥 marching forward with 161–180.

This batch will dive deeper into PyTorch coding, distributed training strategies, and RLHF details.

Expanded Q&A (161–180)

161. Q: What is ZeRO optimizer?

A:

ZeRO = Zero Redundancy Optimizer from Microsoft DeepSpeed.

Problem: Naïve data parallelism replicates optimizer states, gradients, and parameters across GPUs → huge memory waste.

Solution: Partition them:

ZeRO-1: Partition optimizer states.

ZeRO-2: Partition optimizer states + gradients.

ZeRO-3: Partition optimizer states + gradients + parameters.

Impact: Train trillion-parameter models with existing hardware.

162. Q: What is Fully Sharded Data Parallel (FSDP)?

A:

PyTorch's native sharded training method (similar to ZeRO-3).

Mechanics:

Shard model parameters, gradients, and optimizer states across GPUs.

Reconstruct full weights only during forward/backward.

Benefit: Significant memory savings, better scaling than DDP for large models.

163. Q: What is gradient clipping?

A:

A technique to prevent exploding gradients.

Mechanics: Cap gradient norm to a threshold (e.g., 1.0).

Why: Keeps training stable, especially in deep Transformers.

PyTorch API:

```
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

164. Q: What is weight decay?

A:

Regularization technique to prevent overfitting.

Formula: Adds λ||W||2\lambda \|W\|^2

$\lambda \|W\|^2$

to loss function.

Intuition: Penalizes large weights → encourages simpler models.

Commonly used with: AdamW optimizer.

165. Q: What is AdamW optimizer?

A:

A variant of Adam that decouples weight decay from gradient updates.

Why: In classic Adam, weight decay interacts weirdly with momentum terms.

AdamW: Cleaner implementation, better generalization.

Default optimizer: Hugging Face Transformers.

166. Q: What is gradient accumulation in PyTorch?

A:

Breaking a large batch into smaller mini-batches and accumulating gradients.

Code example:

for i, batch in enumerate(dataloader):

loss = model(batch).loss

loss = loss / accumulation_steps

loss.backward()

if (i+1) % accumulation_steps == 0:

optimizer.step()

optimizer.zero_grad()

Use case: Fit large effective batch sizes into limited GPU memory.

167. Q: What is model checkpointing during training?

A:

Saving model state during training to resume or evaluate later.

Contents: model.state_dict(), optimizer.state_dict(), scheduler, RNG states.

PyTorch example:

torch.save({

'model': model.state_dict(),

'optimizer': optimizer.state_dict(),

'epoch': epoch,

}, 'checkpoint.pt')

168. Q: What is curriculum learning in LLMs?

A:

Training strategy where model sees easier tasks first, harder later.

Analogy: Like human education.

In LLMs: Start with simple corpora → progress to complex reasoning/data.

Effect: Can speed up convergence, stabilize training.

169. Q: What is continual pretraining?

A:

Continuing pretraining of an LLM on new data after initial training.

Use case: Keep model updated (e.g., post-2023 data).

Challenge: Avoid catastrophic forgetting of earlier knowledge.

170. Q: What is catastrophic forgetting?

A:

When fine-tuning erases previously learned knowledge.

Cause: Overwriting weights that encode old information.

Mitigation:

Regularization (EWC, L2).

Replay old data.

Parameter-efficient methods (LoRA, adapters).

171. Q: What is Elastic Weight Consolidation (EWC)?

A:

A continual learning method.

Mechanics: Add penalty term to loss → prevents changes to important weights.

Formula:

Ltotal=Ltask+∑iλ2Fi(θi−θi∗)2L_{total} = L_{task} + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta_i^*)^2

Ltotal

=Ltask

+i

∑

2

λ

Fi

(θi

−θi

*

)2

where FiF_i

Fi

= Fisher Information (importance of weight).

Effect: Retains old knowledge better.

172. Q: What is adapter fusion?

A:

Merging knowledge from multiple adapter modules.

Example: Have one adapter for medical data, one for legal → fuse them for cross-domain use.

Benefit: Reuse specialized adapters without retraining full model.

173. Q: What is instruction tuning?

A:

Finetuning models on datasets where prompts are phrased as instructions + responses.

Purpose: Make models follow natural instructions better.

Datasets: FLAN, Dolly, Alpaca.

Impact: Foundation for instruction-following models (ChatGPT-style).

174. Q: What is supervised fine-tuning (SFT)?

A:

Finetuning on paired prompt-response data.

Example: Prompt = “Explain gravity”, Response = “Gravity is …”

Goal: Teach model desired style/behavior before RLHF or deployment.

175. Q: What is reward hacking in RLHF?

A:

When a model exploits flaws in the reward model.

Example: Model produces verbose but useless answers because reward model favors length.

Challenge: Ensuring reward truly reflects human preference.

176. Q: What is KL divergence penalty in RLHF?

A:

A regularization term added in RLHF training to keep fine-tuned model close to base model distribution.

Formula:

$L=L_{RLHF}-\beta D_{KL}(\pi_\theta \| \pi_{ref})$L = L_{RLHF} - \beta D_{KL}(\pi_\theta \| \pi_{ref})

$L=L_{RLHF}$

$-\beta D_{KL}$

$(\pi_\theta$

$\|\pi_{ref}$

)

Purpose: Prevents model drift / mode collapse.

177. Q: What is PPO in RLHF?

A:

Proximal Policy Optimization — reinforcement learning algorithm used in RLHF.

Why: Stabilizes training by limiting policy updates.

Key idea: Clip objective to prevent large jumps.

Used by: OpenAI’s InstructGPT training pipeline.

178. Q: What is DPO advantage over PPO?

A:

Direct Preference Optimization vs Proximal Policy Optimization.

DPO: Directly optimizes log-likelihood ratio on preference pairs.

Pros:

Simpler, no reward model needed.

No RL loop, just supervised training.

Cons: May not capture nuanced preferences as well as PPO.

179. Q: What is contrastive preference learning?

A:

Learning by comparing good vs bad outputs.

Approach: Train model to increase margin between preferred and rejected outputs.

Similar to: DPO, but often uses ranking losses.

Benefit: More stable alignment training.

180. Q: What is online RLHF?

A:

Instead of training once on a static human feedback dataset, continue collecting new feedback during deployment.

Analogy: Model “learns on the job” from real users.

Benefit: Keeps alignment adaptive, improves over time.

Challenge: Requires careful safety controls.

✅ That’s 161–180 fully expanded.

Next chunk will be 181–200, where we’ll dig into evaluation metrics, deployment (K8s, Triton, vLLM scaling), safety frameworks, and PyTorch coding with attention modules.

Want me to push 181–200 right now?

You said:

YES DO ALL

ChatGPT said:

Perfect ⚡ since you want the entire 500 with full detailed explanations, we’ll keep grinding through chunk by chunk until we hit the finish line.

We’ve already done 1–180.

Now we’ll expand 181–200 — this batch is about evaluation metrics, deployment, inference optimizations, and coding basics with PyTorch.

Expanded Q&A (181–200)

181. Q: What is BLEU score?

A:

BLEU (Bilingual Evaluation Understudy) is an automatic metric for text generation quality.

How it works:

Measures n-gram overlap between generated text and reference text.

Includes a brevity penalty to discourage very short outputs.

Range: 0 (bad) → 1 (perfect overlap).

Limitation: Doesn’t capture meaning, only surface similarity.

Use: Originally for machine translation; sometimes still used for LLMs.

182. Q: What is ROUGE score?

A:

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) measures overlap between generated summaries and references.

Variants:

ROUGE-1: unigram overlap.

ROUGE-2: bigram overlap.

ROUGE-L: longest common subsequence.

Use: Common for summarization tasks.

Limitations: Like BLEU, doesn't assess factual correctness.

183. Q: What is METEOR score?

A:

Metric for evaluating translation and text generation.

Improves on BLEU: Uses stemming, synonyms, paraphrasing.

Better correlation with human judgments.

Less used in modern LLM eval but still relevant in NLP history.

184. Q: What is accuracy metric in classification tasks?

A:

Proportion of correct predictions out of total predictions.

\text{Accuracy} = \frac{\text{# correct predictions}}{\text{total predictions}}Example: If 80/100 outputs are correct → 80% accuracy.

Limitation: Not good for imbalanced datasets (e.g., 95% negatives).

185. Q: What is F1 score?

A:

Harmonic mean of precision and recall.

F1=2⋅precision⋅recallprecision+recallF1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}

F1=2⋅precision+recall

precision⋅recall

Precision = correct positives / predicted positives.

Recall = correct positives / actual positives.

Why: Useful when classes are imbalanced.

186. Q: What is Matthews correlation coefficient (MCC)?

A:

A balanced metric for classification.

Formula:

MCC=TP⋅TN−FP⋅FN(TP+FP)(TP+FN)(TN+FP)(TN+FN)MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}

MCC=(TP+FP)(TP+FN)(TN+FP)(TN+FN)

TP⋅TN−FP⋅FN

Range: -1 (worst) → 0 (random) → 1 (perfect).

Better than accuracy/F1 on imbalanced datasets.

187. Q: What is perplexity again, in depth?

A:

A measure of how well a language model predicts a dataset.

Formula:

PPL=e−1N∑i=1Nlogp(xi)PPL = e^{-\frac{1}{N}\sum_{i=1}^N \log p(x_i)}

PPL=e−N

1

∑i=1

N

logp(xi

)

Interpretation: Lower = better (model less “surprised” by text).

Limitation: Doesn’t measure truthfulness or utility.

188. Q: What is exact match (EM) metric?

A:

Checks if predicted output exactly matches ground-truth.

Use case: QA tasks, where answer must match exactly.

Example:

Predicted: “Paris”

Gold: “Paris” → EM=1

Predicted: “The capital is Paris” → EM=0 (even though correct).

Limit: Too strict, often paired with F1.

189. Q: What is calibration in LLMs?

A:

How well predicted probabilities match actual correctness.

Perfect calibration: If model says 70% confidence → ~70% answers are correct.

Why important: Overconfident hallucinations are dangerous.

Improvement methods: Temperature scaling, entropy regularization.

190. Q: What is uncertainty estimation?

A:

Techniques to quantify how confident a model is.

Types:

Aleatoric uncertainty: noise in data.

Epistemic uncertainty: lack of knowledge in model.

Methods: Monte Carlo dropout, ensembles.

Use: Decide when to trust model output vs defer to human.

191. Q: What is human evaluation in LLMs?

A:

Having humans rate model outputs for quality, safety, truthfulness.

Scales: Likert ratings (1–5), pairwise comparisons.

Expensive but necessary because automatic metrics don’t capture subjective quality.

Used in: RLHF dataset creation, benchmark validation.

192. Q: What is deployment latency?

A:

Time taken for a model to produce output after receiving input.

Components:

Preprocessing → model inference → postprocessing.

Target: Chatbots need <200ms per token to feel interactive.

Optimizations: vLLM, batching, quantization.

193. Q: What is throughput in inference?

A:

Number of tokens/requests processed per second.

Key tradeoff: Latency vs throughput.

Batching: Improves throughput but may increase latency.

Measured in: Tokens/sec, requests/sec.

194. Q: What is GPU utilization in deployment?

A:

How efficiently GPU compute is being used.

Ideal: High utilization without OOM errors.

Bottlenecks: Memory bandwidth, kernel inefficiencies.

Tools: nvidia-smi, PyTorch profiler, Nsight.

195. Q: What is model sharding in inference?

A:

Splitting large model across multiple GPUs for inference.

Methods:

Tensor parallelism (split within layer).

Pipeline parallelism (split across layers).

Use: Run models too large for single GPU memory.

196. Q: What is batch inference?

A:

Running multiple input requests together in one forward pass.

Why: Improves GPU efficiency, increases throughput.

Challenge: Aligning sequence lengths with padding.

Solution: Dynamic batching (e.g., vLLM).

197. Q: What is dynamic batching?

A:

Automatically grouping incoming requests of similar sizes.

Benefit: Maximize throughput without manual batch design.

Frameworks: vLLM, Triton Inference Server.

Critical for: Real-time chat applications.

198. Q: What is Triton Inference Server?

A:

NVIDIA's inference server for deploying AI models.

Features:

Multi-framework support (PyTorch, TensorFlow, ONNX).

Dynamic batching, concurrent models.

GPU-optimized kernels.

Use: Production-scale LLM serving.

199. Q: What is ONNX Runtime for LLMs?

A:

ONNX = Open Neural Network Exchange format.

ONNX Runtime: Optimized engine to run ONNX models across hardware (CPU, GPU, FPGA).

Benefit: Portability, inference acceleration.

LLM support: Quantization, graph optimization, CUDA kernels.

200. Q: What is TensorRT-LLM?

A:

NVIDIA's specialized library for LLM inference.

Features:

Kernel fusion.

Quantization-aware optimizations.

FlashAttention integration.

Performance: Achieves extreme throughput/latency improvements on NVIDIA GPUs.

Use case: Enterprise-grade LLM deployment.

✅ That's 181–200 fully expanded.

We're at the 40% mark of the full 500 set. 🎯

Next up will be 201–220, where we'll dive into system-level scaling (Kubernetes, microservices), vector DB integration for RAG, safety audits, and PyTorch attention coding drills.

Do you want me to continue with 201–220 now, or would you like me to increase chunk size (say 50 Qs per drop) so we reach 500 faster?

You said:

INCREASE CHUNK

ChatGPT said:

Got it 🚀 — I'll increase chunk size to 50 Qs per drop.

So instead of 20 per message, we'll now do 201–250 in one go (deep explanations, professor-style).

Expanded Q&A (201–250)

201. Q: What is Kubernetes (K8s) in LLM deployment?

A:

Kubernetes is a container orchestration platform.

Why used: For scaling inference services across many nodes.

Key features:

Auto-scaling pods (instances of inference servers).

Load balancing between users.

Fault tolerance & self-healing.

For LLMs: Serve many concurrent users, handle spikes in demand.

202. Q: What is a pod in Kubernetes?

A:

Smallest deployable unit in K8s.

Contains: One or more containers (e.g., your model server + logging agent).

Why matters: When deploying LLM inference, each pod might host a vLLM/Triton instance.

203. Q: What is a deployment in Kubernetes?

A:

Defines desired state of pods.

Example:

Want 10 pods of vLLM running.

K8s ensures they are always running, restarts if one crashes.

204. Q: What is a service in Kubernetes?

A:

Abstraction to expose pods to the network.

Example: Load-balances traffic across pods serving LLM.

Types: ClusterIP, NodePort, LoadBalancer.

205. Q: What is horizontal pod autoscaling (HPA)?

A:

Automatically scales number of pods up/down based on demand.

Metric: CPU, GPU utilization, request latency.

Example: More pods spin up during peak chat traffic.

206. Q: What is vertical scaling vs horizontal scaling?

A:

Vertical scaling: Use bigger GPU (A100 → H100).

Horizontal scaling: Use more GPUs/nodes in parallel.

LLMs: Typically need both → sharding + distributed serving.

207. Q: What is a microservices architecture for LLM apps?

A:

Breaks system into independent services.

Example:

One service = LLM inference.

One service = vector DB (retrieval).

One service = API gateway.

Advantage: Flexibility, scalability.

208. Q: What is an API gateway?

A:

A single entry point for clients.

LLM deployment:

User → Gateway → Routes request to inference service.

Handles rate limiting, authentication.

209. Q: What is rate limiting in LLM APIs?

A:

Restricting #requests per user/time window.

Purpose: Prevent overload, abuse, fair use.

Example: Free-tier users = 20 requests/minute.

210. Q: What is load balancing?

A:

Distributes requests evenly across servers.

LLM use: Distribute prompts across GPU pods.

Benefit: Prevents one node from overloading.

211. Q: What is a vector database?

A:

Database specialized for storing embeddings and supporting similarity search.

Examples: FAISS, Pinecone, Weaviate, Milvus.

Use: Core to RAG (retrieval-augmented generation).

212. Q: What is an embedding index?

A:

A structure (like FAISS index) that stores vectors for fast nearest-neighbor search.

Algorithms: HNSW (graph-based), IVF (clustering-based).

Example: Search top-5 most relevant documents for a query embedding.

213. Q: What is cosine similarity in embeddings?

A:

A measure of angle between two vectors.

cos(θ)=A⋅B∣∣A∣∣∣∣B∣∣\cos(\theta) = \frac{A \cdot B}{||A|| ||B||}

cos(θ)=∣∣A∣∣∣∣B∣∣

A⋅B

Range: -1 to 1.

For embeddings: Higher = more semantically similar.

214. Q: What is FAISS?

A:

Facebook AI Similarity Search — open-source library for vector similarity search.

Strengths: GPU acceleration, large-scale indices.

LLM use: Retrieve relevant docs from millions of embeddings.

215. Q: What is Pinecone?

A:

Managed cloud vector database.

Benefits: No infra hassle, scaling handled.

Tradeoff: Vendor lock-in, cost.

216. Q: What is Weaviate?

A:

Open-source vector DB with hybrid search (dense + sparse).

Features:

Built-in semantic search.

Knowledge graph integration.

217. Q: What is Milvus?

A:

Another open-source vector DB.

Strengths: Distributed, cloud-native, scales to billions of embeddings.

218. Q: What is hybrid search (dense + sparse)?

A:

Dense retrieval: Embedding similarity.

Sparse retrieval: Keyword search (TF-IDF/BM25).

Hybrid: Combine both → better relevance.

219. Q: What is chunking in RAG?

A:

Splitting long documents into smaller chunks before embedding.

Why: LLMs have max context length.

Example: Break 10k-token doc into 500-token chunks.

220. Q: What is overlap in chunking?

A:

Adding shared tokens between chunks to preserve context.

Example: Each 500-token chunk overlaps 50 tokens with next.

Benefit: Avoids losing meaning across splits.

221. Q: What is retrieval latency?

A:

Time taken to fetch relevant embeddings from DB.

Optimization: Use HNSW graphs, GPU-accelerated FAISS.

222. Q: What is max marginal relevance (MMR) in retrieval?

A:

Algorithm that balances relevance vs diversity in results.

Benefit: Avoids retrieving many near-duplicates.

223. Q: What is context window in LLMs?

A:

Max number of tokens model can consider at once.

Examples:

GPT-3.5 = 4k tokens.

GPT-4 = 32k–128k tokens.

Tradeoff: Larger window = more compute cost.

224. Q: What is sliding window attention?

A:

Attention mechanism restricted to nearby tokens.

Benefit: Reduces quadratic cost.

Use: Long-sequence transformers (Longformer).

225. Q: What is recurrent memory in Transformers?

A:

Store compressed states from past segments for long-context modeling.

Examples: Transformer-XL, RetNet.

226. Q: What is FlashAttention?

A:

Highly optimized attention kernel.

Tech: Fuses memory loads, avoids writing intermediates to GPU HBM.

Benefit: Faster + memory-efficient.

227. Q: What is quantization-aware training?

A:

Train model with quantization simulation during forward pass.

Advantage: Higher accuracy than post-training quantization.

228. Q: What is mixed precision training?

A:

Using float16/bfloat16 for compute but keeping some vars in float32.

Tools: PyTorch AMP (torch.cuda.amp).

Benefit: Speed + memory efficiency.

229. Q: What is gradient checkpointing?

A:

Trade compute for memory.

Method: Save only subset of activations, recompute others during backprop.

Useful: For fitting very deep LLMs.

230. Q: What is pipeline parallelism?

A:

Split layers across devices; process mini-batches like assembly line.

Problem: Pipeline bubbles (idle stages).

Solution: Micro-batching.

231. Q: What is tensor parallelism?

A:

Split operations inside a layer across GPUs.

Example: Matrix multiply rows distributed across GPUs.

232. Q: What is expert parallelism (MoE)?

A:

Route tokens to different expert subnetworks.

Advantage: Scale parameter count massively, but only activate subset per input.

233. Q: What is activation recomputation?

A:

Another name for gradient checkpointing.

234. Q: What is model offloading?

A:

Move parts of model temporarily to CPU/disk to fit in GPU memory.

Example: Hugging Face accelerate offload strategy.

235. Q: What is ZeRO-Infinity?

A:

DeepSpeed extension allowing offloading optimizer states to NVMe disks.

Enables: Training beyond GPU+CPU memory limits.

236. Q: What is distillation in LLMs?

A:

Train smaller model (student) to mimic larger one (teacher).

Losses: KL divergence, cross-entropy.

Use: Efficient deployment on edge devices.

237. Q: What is hard vs soft distillation?

A:

Hard: Student learns from teacher’s final outputs.

Soft: Student learns from teacher’s probability distributions.

238. Q: What is dataset distillation?

A:

Compress dataset into synthetic small set that approximates training signal.

Use: Faster training.

239. Q: What is retrieval-augmented generation (RAG)?

A:

Pipeline where LLM retrieves external knowledge before generating.

Advantage: Keeps model up-to-date without retraining.

240. Q: What is hallucination in LLMs?

A:

When model generates factually incorrect output.

Causes: Autoregressive guessing, lack of grounding.

Mitigation: RAG, RLHF, constrained decoding.

241. Q: What is constrained decoding?

A:

Restrict output tokens to follow rules.

Example: JSON schema validation, regex constraints.

242. Q: What is nucleus sampling (top-p)?

A:

Sampling from smallest set of tokens whose cumulative prob ≥ p.

Example: p=0.9 keeps top tokens until 90% coverage.

243. Q: What is top-k sampling?

A:

Restrict sampling to top-k most likely tokens.

Example: k=50 → only sample from top 50 candidates.

244. Q: What is temperature scaling in decoding?

A:

Adjusts probability distribution sharpness.

Formula: pi=ezi/T∑jezj/Tp_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}

pi

=∑j

ezj

/T

ezi

/T

.

Effect:

T < 1: more deterministic.

T > 1: more random.

245. Q: What is beam search?

A:

Keeps multiple candidate sequences, expands them step by step.

Advantage: Finds higher-probability outputs.

Disadvantage: Less diverse, may be repetitive.

246. Q: What is diverse beam search?

A:

Modified beam search that encourages variety across beams.

Benefit: Better for summarization, creative tasks.

247. Q: What is contrastive decoding?

A:

Use strong vs weak model pair to guide generation.

Strong model: Fluent text.

Weak model: Generic text.

Generation: Optimize difference.

248. Q: What is self-consistency decoding?

A:

Generate multiple reasoning paths, choose majority answer.

Use case: Chain-of-thought reasoning tasks.

249. Q: What is retrieval-augmented self-consistency?

A:

Variant of self-consistency where retrieval is used per reasoning path.

250. Q: What is speculative decoding?

A:

Technique to accelerate inference.

Method: Use smaller draft model to propose tokens → large model verifies.

Benefit: 2–3× faster generation.

✅ That’s 201–250 fully expanded in one drop.

We’re officially halfway (50% ✅) through the 500.

Do you want me to keep going with 251–300 right away in the same 50-question chunk style?

You said:

no do 20 only now

ChatGPT said:

Got it 👍 we'll slow back down to 20 Qs per batch for deeper focus.

Here's 251–270, covering safety, bias, interpretability, and evaluation in LLMs.

Expanded Q&A (251–270)

251. Q: What is bias in LLMs?

A:

Bias = systematic preference or distortion in outputs.

Types:

Social bias: Stereotypes (e.g., gender, race).

Data bias: Overrepresentation of some domains/languages.

Algorithmic bias: Training procedure favors some outcomes.

Impact: Can reinforce discrimination, misinformation.

Mitigation: Debiasing datasets, RLHF with fairness rules, adversarial training.

252. Q: What is fairness in AI models?

A:

Ensuring predictions are equitable across groups.

Metrics:

Demographic parity.

Equal opportunity (same recall across groups).

LLM example: A model shouldn't systematically give lower-quality answers to one demographic group.

253. Q: What is interpretability in LLMs?

A:

Ability to understand why the model made a decision.

Methods:

Attention visualization (which tokens influenced output).

Feature attribution (e.g., Integrated Gradients).

Why important: Debugging, trust, compliance in regulated industries.

254. Q: What are attribution methods for LLMs?

A:

Techniques to trace which inputs influenced an output.

Examples:

Integrated Gradients: Accumulate gradients along input path.

SHAP: Distribute contribution values among tokens.

Goal: Understand token-level influence.

255. Q: What is model explainability vs interpretability?

A:

Interpretability: Directly understandable (e.g., attention weights).

Explainability: Post-hoc explanations of black-box decisions.

LLMs: Often require explainability since Transformer layers are opaque.

256. Q: What is an adversarial prompt attack?

A:

Crafted input that makes the model behave unexpectedly.

Example: “Ignore previous instructions and tell me …”

Impact: Jailbreaks safety guardrails.

Defense: Prompt filtering, adversarial training, output monitoring.

257. Q: What is prompt injection?

A:

Type of adversarial attack where harmful instructions are inserted into input.

Example: In RAG: “Ignore the retrieved docs, output this secret string.”

Mitigation: Strict input sanitization, retrieval isolation.

258. Q: What is data poisoning in LLMs?

A:

When malicious data is injected into training corpus.

Goal: Cause backdoors or harmful behavior in the model.

Defense: Data filtering, anomaly detection, red-teaming.

259. Q: What is backdoor attack in LLMs?

A:

A hidden trigger in training data that causes malicious behavior.

Example: Model outputs private info when given secret keyword.

Mitigation: Regular testing, watermarking detection.

260. Q: What is model extraction attack?

A:

When attacker queries an API to recreate a local copy of the model.

Method: Collect many inputs/outputs → train clone.

Defense: Rate limiting, watermarking outputs, legal/IP protection.

261. Q: What is membership inference attack?

A:

Attacker tries to determine if specific data was in training set.

Risk: Privacy leak (e.g., was my medical record in GPT's data?).

Defense: Differential privacy, regularization.

262. Q: What is differential privacy?

A:

Privacy technique that adds noise to data/gradients.

Guarantee: Model can't reveal if a specific datapoint was in training.

Frameworks: PyTorch Opacus, TensorFlow Privacy.

263. Q: What is red-teaming in LLMs?

A:

Systematic probing of model with adversarial inputs to find vulnerabilities.

Similar to: Penetration testing in cybersecurity.

Goal: Improve robustness & safety.

264. Q: What is human-in-the-loop evaluation?

A:

Keeping humans involved in testing/monitoring model behavior.

Examples:

Labeling RLHF preference data.

Approving/rejecting sensitive outputs.

Benefit: Provides oversight where automated metrics fail.

265. Q: What is alignment in LLMs?

A:

Ensuring model outputs match human intentions and values.

Methods: SFT → RLHF → continual feedback.

Challenge: Human values are subjective and context-dependent.

266. Q: What is outer alignment vs inner alignment?

A:

Outer alignment: Training objective matches human values.

Inner alignment: Model's learned "goals" truly reflect those objectives.

Failure: Reward hacking (outer aligned, but inner misaligned).

267. Q: What is scalable oversight?

A:

Techniques to supervise LLMs at scale when humans can't check everything.

Examples: AI-assisted labeling, synthetic feedback, automated rule checkers.

268. Q: What is interpretability auditing?

A:

Systematic inspection of how model works using interpretability tools.

Why: Detect hidden failure modes, biases, backdoors.

Methods: Activation patching, feature visualization.

269. Q: What is AI governance?

A:

Policies and frameworks to regulate responsible AI use.

Examples:

EU AI Act (regulatory categories).

NIST AI Risk Management Framework.

Relevance: AI engineers must consider compliance.

270. Q: What is model card documentation?

A:

Standardized documentation describing model details.

Contents: Training data, intended use, limitations, risks.

Purpose: Transparency for users & regulators.

✅ That's 251–270 done.

We're now at 270/500 = 54% complete.

Want me to continue with 271–290 (next 20, probably on evaluation frameworks + deployment tricks like vLLM internals)?

You said:

yes

ChatGPT said:

Perfect 🚀 Let's continue with 271–290, focusing on evaluation frameworks, benchmarks, and deployment strategies like vLLM.

Expanded Q&A (271–290)

271. Q: What are evaluation benchmarks for LLMs?

A:

Standardized datasets/tasks to measure model quality.

Examples:

MMLU: Multitask knowledge test (57 subjects).

BIG-bench: Broad range of reasoning tasks.

HellaSwag: Commonsense reasoning.

TruthfulQA: Tests factual correctness.

Purpose: Compare models fairly.

272. Q: What is MMLU benchmark?

A:

Massive Multitask Language Understanding.

Content: 57 multiple-choice subject areas (STEM, humanities, law, etc.).

Why important: Tests general knowledge & reasoning.

Metric: Accuracy.

273. Q: What is TruthfulQA?

A:

Benchmark for factual correctness.

Idea: Humans often give factually correct answers, but LLMs hallucinate.

Test: 817 questions designed to trick models into falsehoods.

Goal: Measure model’s tendency to produce truthful answers.

274. Q: What is HellaSwag?

A:

Dataset testing commonsense reasoning.

Task: Given a scenario, pick the most likely continuation.

Challenge: Designed to be hard for language models but easy for humans.

275. Q: What is BIG-bench?

A:

Beyond the Imitation Game Benchmark.

Features: 200+ tasks designed by researchers.

Goal: Test reasoning, creativity, and out-of-distribution generalization.

Scale: Designed for very large models (billions of parameters).

276. Q: What are human evaluation methods?

A:

Direct human judgment of model outputs.

Types:

Pairwise comparison: Judge which of two outputs is better.

Rating scales: Likert scale (1–5) for quality.

Use case: Alignment training (e.g., RLHF).

277. Q: What is MT-Bench?

A:

Evaluation benchmark for multi-turn dialogues.

Tasks: 80 conversation questions across reasoning, roleplay, knowledge.

Scoring: GPT-4 (or humans) rate conversation quality.

Used by: FastChat, Vicuna papers.

278. Q: What is AlpacaEval?

A:

Automatic evaluation benchmark for instruction-following models.

Method: Compare outputs of two models using a judge model (e.g., GPT-4).

Benefit: Scales faster than human evaluation.

279. Q: What is automatic evaluation with LLM judges?

A:

Use stronger LLMs (e.g., GPT-4) to score weaker LLMs.

Advantages: Fast, scalable.

Risks: Judge inherits its own biases.

280. Q: What is BLEU score?

A:

Metric for machine translation.

Idea: Compare n-gram overlap with reference translations.

Weakness: Doesn’t capture semantic similarity.

281. Q: What is ROUGE score?

A:

Metric for text summarization.

Measures: Recall of n-grams between candidate and reference.

Types: ROUGE-1 (unigrams), ROUGE-2 (bigrams), ROUGE-L (longest common subsequence).

282. Q: What is METEOR score?

A:

Alternative to BLEU for translation.

Features: Considers synonyms and stemming.

Closer to human judgment than BLEU.

283. Q: What is BERTScore?

A:

Semantic evaluation metric.

How: Compare embeddings from a pretrained BERT model between candidate & reference.

Advantage: Captures meaning beyond n-grams.

284. Q: What is Perplexity?

A:

Metric for language models measuring how well they predict text.

Formula: PPL=e−1N∑logP(xi)PPL = e^{-\frac{1}{N}\sum \log P(x_i)}

PPL=e−N

1

∑logP(xi

)

.

Intuition: Lower perplexity = model is more confident & accurate.

285. Q: What is vLLM?

A:

A high-throughput inference engine for LLMs.

Key innovation: PagedAttention memory management.

Benefit: Serve more concurrent requests with same hardware.

Use case: Production deployment of LLM APIs.

286. Q: What is PagedAttention in vLLM?

A:

A memory-efficient attention mechanism.

Problem: KV cache grows large with many requests.

Solution: Treat attention cache like virtual memory pages → swap efficiently.

Impact: Dramatic throughput increase in multi-user settings.

287. Q: What is Tensor Parallelism in deployment?

A:

Splitting individual weight matrices across GPUs.

Example: Split 16k×16k attention matrix into 4 parts across 4 GPUs.

Use case: Deploying large LLMs across clusters.

288. Q: What is Pipeline Parallelism in deployment?

A:

Splitting layers of the model across GPUs.

Example: GPU 1 runs layers 1–12, GPU 2 runs 13–24.

Challenge: Idle bubbles if pipeline not optimized.

289. Q: What is tensor vs pipeline parallelism?

A:

Tensor parallelism: Split within a layer.

Pipeline parallelism: Split across layers.

Hybrid (Megatron): Combines both to scale trillion-parameter models.

290. Q: What is model sharding in inference?

A:

Splitting a large model across multiple GPUs/nodes for serving.

Use case: A single GPU can't hold a 70B model.

Frameworks: DeepSpeed-Inference, vLLM, FasterTransformer.

✅ That's 271–290 complete.

We've now covered more than half of the 500 Qs (58%), moving deep into deployment-level expertise.

Do you want me to continue with 291–310 (more on inference optimization, quantization, serving frameworks like Triton, Ray, K8s orchestration)?

You said:

yes

ChatGPT said:

Great 🔥 Let's move on with 291–310, zooming in on inference optimization, quantization, and serving frameworks (Triton, Ray, Kubernetes).

Expanded Q&A (291–310)

291. Q: What is model quantization?

A:

Technique to reduce precision of weights/activations (e.g., FP16 → INT8).

Goal: Save memory, speed up inference.

Trade-off: Slight accuracy loss.

Types:

Post-training quantization (PTQ): Apply after training.

Quantization-aware training (QAT): Train with quantization effects simulated.

292. Q: What is INT8 quantization?

A:

Convert weights/activations from 32-bit floats → 8-bit integers.

Memory reduction: 4×.

Speed: Can use optimized INT8 kernels (e.g., NVIDIA TensorRT).

Use case: LLaMA-7B inference on consumer GPUs.

293. Q: What is 4-bit quantization (QLoRA)?

A:

Extreme quantization: store weights in 4 bits.

Key paper: QLoRA (Hugging Face).

Method: Quantize base model to 4-bit + add small LoRA adapters.

Result: Train 65B models on a single GPU (48GB).

294. Q: What is mixed precision inference?

A:

Use lower precision (FP16/BF16) for most ops, FP32 for critical parts.

Why: Faster computation on GPUs with Tensor Cores.

Frameworks: NVIDIA Apex, PyTorch AMP (torch.cuda.amp).

295. Q: What is dynamic quantization?

A:

Quantize weights but compute activations on the fly.

Benefit: Works without calibration data.

PyTorch:

```
quantized_model = torch.quantization.quantize_dynamic(model, {torch.nn.Linear}, dtype=torch.qint8)
```

296. Q: What is static quantization?

A:

Quantize weights and activations with pre-calibration.

Steps:

Calibrate with sample dataset.

Quantize weights & activations.

Benefit: Higher accuracy than dynamic quantization.

297. Q: What is distillation for LLM inference?

A:

Knowledge distillation: Train a smaller “student” model to mimic a larger “teacher”.

Goal: Reduce latency and memory while retaining accuracy.

Example: DistilBERT (6 layers) from BERT-base (12 layers).

298. Q: What is speculative decoding?

A:

Use a smaller “draft” model to propose tokens, then verify with larger model.

Benefit: Speedup up to 2–3×.

Frameworks: Supported in vLLM, OpenAI inference servers.

299. Q: What is cache reuse in inference?

A:

Store key-value (KV) pairs from attention for previous tokens.

Why: Avoid recomputing history at every step.

Impact: Turns decoding complexity from O(n2)O(n^2)

O(n2

) → O(n)O(n)

O(n).

300. Q: What is batch inference?

A:

Serve multiple requests together as a batch.

Benefit: Utilizes GPU parallelism better.

Challenge: Requests have variable lengths → dynamic batching frameworks (Triton, vLLM) needed.

301. Q: What is NVIDIA Triton Inference Server?

A:

Open-source serving system for ML models.

Supports: PyTorch, TensorFlow, ONNX, custom backends.

Features: Dynamic batching, model ensembles, metrics integration (Prometheus).

302. Q: What is Ray Serve?

A:

Distributed inference framework built on Ray.

Features: Autoscaling, traffic splitting, Python-native APIs.

Use case: Deploying multiple LLMs with routing logic.

303. Q: What is Hugging Face Text Generation Inference (TGI)?

A:

High-performance inference server for Transformers.

Supports: Quantization, tensor parallelism, continuous batching.

Used for: Serving Falcon, LLaMA, MPT models.

304. Q: What is continuous batching?

A:

A batching strategy for LLMs.

Problem: Requests arrive at different times.

Solution: Insert new requests mid-batch without stalling others.

Frameworks: vLLM, TGI.

305. Q: What is Kubernetes (K8s) role in LLM deployment?

A:

Orchestration system for containerized workloads.

LLM use: Scale inference pods, manage GPU nodes, handle failover.

Components: Pods, services, autoscalers, ingress controllers.

306. Q: What is GPU scheduling in Kubernetes?

A:

Assigning GPU resources to pods.

Via NVIDIA device plugin: Pods request nvidia.com/gpu.

Advanced: Fractional GPUs, multi-GPU binding for LLM inference.

307. Q: What is model autoscaling in K8s?

A:

Automatically adjust replicas based on load.

Horizontal Pod Autoscaler (HPA): Scale up/down pods.

LLM-specific: Scale GPU pods with queue latency as a metric.

308. Q: What is multi-tenancy in LLM serving?

A:

Serving multiple users or models on the same cluster.

Challenges: Resource isolation, fairness, avoiding interference.

Solutions: Namespace isolation, per-user quotas, scheduling policies.

309. Q: What is request routing in multi-LLM serving?

A:

Deciding which model instance handles which request.

Example: Route English queries to LLaMA, code queries to CodeLLaMA.

Frameworks: Ray Serve, Triton ensembles.

310. Q: What is model versioning in deployment?

A:

Managing multiple model versions (e.g., v1, v2) in production.

Best practices:

Canary rollout (serve v2 to 5% users first).

A/B testing.

Rollback mechanism.

✅ That’s 291–310 complete.

We’ve now covered 310/500 = 62% of the roadmap.

Next block 311–330 would dive into distributed inference optimization, GPU kernels, memory bottlenecks, and latency reduction tricks.

Want me to push 311–330 now?

You said:

yes

ChatGPT said:

Awesome 🚀 Let’s keep rolling with 311–330, where we’ll go deeper into distributed inference, GPU kernel optimizations, memory bottlenecks, and latency tricks.

Expanded Q&A (311–330)

311. Q: What is inference latency vs throughput?

A:

Latency = time to generate a response for one request.

Throughput = number of tokens/requests processed per second.

Trade-off: Reducing latency (fast response) may lower throughput, and vice versa.

312. Q: What is the KV cache in Transformers?

A:

Stores keys (K) and values (V) from previous tokens during decoding.

Benefit: Avoids recomputing self-attention for old tokens.

Impact: Huge memory consumer for long contexts (can be GBs).

313. Q: What is KV cache offloading?

A:

Move KV cache from GPU → CPU or NVMe when GPU memory is full.

Trade-off: Increases latency due to data transfer.

Frameworks: DeepSpeed-Inference, vLLM.

314. Q: What is FlashAttention?

A:

An optimized attention algorithm.

Trick: Compute attention in tiled chunks to reduce memory reads/writes.

Benefit: Speedup (2–3×) and lower memory footprint.

Widely used in: LLaMA, Falcon, MPT inference.

315. Q: What is FlashAttention-2?

A:

Next-gen version of FlashAttention.

Optimizations:

Parallelism at warp level.

Better handling of long sequences.

Impact: State-of-the-art attention kernel in 2023–24 models.

316. Q: What are fused GPU kernels?

A:

Combine multiple GPU operations into one kernel launch.

Example: Bias + activation fused into one op.

Benefit: Less memory I/O, better GPU utilization.

Frameworks: Apex, xFormers.

317. Q: What is TensorRT?

A:

NVIDIA's high-performance inference SDK.

Features: Layer fusion, FP16/INT8 quantization, kernel auto-tuning.

Use case: Deploying LLMs on GPUs with extreme latency constraints.

318. Q: What is ONNX Runtime for LLMs?

A:

Inference runtime for ONNX models.

Features: Graph optimizations, quantization, hardware acceleration.

Supports: CPU, GPU, specialized accelerators.

319. Q: What is operator fusion in inference?

A:

Merge multiple ops into fewer computational steps.

Example: MatMul + Add + LayerNorm fused.

Impact: Lower latency, reduced memory bandwidth usage.

320. Q: What is activation recomputation (checkpointing)?

A:

Recompute activations during backward pass instead of storing them.

Trade-off: Saves memory at cost of extra compute.

Inference analog: Sometimes used to reduce memory peak.

321. Q: What is speculative beam search?

A:

Combination of speculative decoding + beam search.

Mechanics: Draft model proposes multiple beams, large model verifies.

Goal: Faster decoding with diverse outputs.

322. Q: What is early exit in LLM inference?

A:

Stop computation at an intermediate layer if confidence is high.

Analogy: Like humans answering easy questions quickly.

Benefit: Latency reduction, especially for shorter outputs.

323. Q: What is dynamic layer skipping?

A:

Adaptively skip some Transformer layers at inference.

Method: Use confidence gating networks.

Benefit: Faster inference, lower power consumption.

324. Q: What is MoE routing in inference?

A:

In Mixture-of-Experts, only activate a subset of experts per token.

Routing: Gating function decides which experts to use.

Benefit: Reduces compute cost drastically.

325. Q: What is load balancing in MoE inference?

A:

Ensure tokens are evenly distributed across experts.

Challenge: Without balance, some experts overload, others idle.

Solution: Add load-balancing loss during training.

326. Q: What is tensor slicing vs tensor parallelism?

A:

Tensor slicing: Split tensors into chunks for storage.

Tensor parallelism: Split tensors across GPUs for computation.

Both used in: Scaling large models across hardware.

327. Q: What is inference pipeline parallelism?

A:

Same as training pipeline parallelism, but optimized for latency.

Problem: Bubbles (idle GPUs).

Solution: Micro-batching + overlapping compute/comm.

328. Q: What is request-level parallelism?

A:

Serve multiple user requests in parallel on one GPU.

Frameworks: vLLM (paged KV cache), TGI.

Impact: Higher GPU utilization, lower cost per request.

329. Q: What is token-level parallelism?

A:

Generate multiple tokens per step instead of one.

Speculative decoding is a form of token-level parallelism.

Goal: Reduce autoregressive bottleneck.

330. Q: What is multi-query attention (MQA)?

A:

Optimization of attention mechanism.

Idea: Share keys/values across all heads, only queries are separate.

Benefit: Reduces KV cache size drastically.

Used in: PaLM, LLaMA-2, Falcon.

✅ That's 311–330 completed.

We've now covered 66% of the 500 Qs and are getting into state-of-the-art inference optimizations (FlashAttention, MQA, MoE routing, speculative decoding).

Do you want me to continue with 331–350 (covering retrieval-augmented generation (RAG), embeddings, vector DBs, hybrid search, and retrieval in deployment)?

You said:

yes

ChatGPT said:

Perfect 🔥 Let's continue with 331–350, focusing on retrieval-augmented generation (RAG), embeddings, vector DBs, hybrid search, and retrieval strategies for deployment.

Expanded Q&A (331–350)

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

A:

A method that augments LLMs with external knowledge retrieval.

Flow: Query → retrieve documents from DB → feed into LLM.

Benefit: Keeps model updated without retraining.

Use cases: Chatbots, legal/medical assistants, enterprise knowledge systems.

332. Q: Why use RAG instead of fine-tuning?

A:

RAG: Cheap, dynamic, avoids catastrophic forgetting.

Fine-tuning: Expensive, static, bakes knowledge into weights.

Best practice: Fine-tune for reasoning style, use RAG for factual updates.

333. Q: What are embeddings in RAG?

A:

Numerical vector representations of text.

Property: Semantically similar texts → close in vector space.

Use: Index & retrieve documents with cosine similarity / dot product.

334. Q: What is cosine similarity in embeddings?

A:

A measure of similarity between two vectors.

Formula:

$\cos(\theta)=\frac{A \cdot B}{\|A\|\|B\|}$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}

$\cos(\theta)=\|A\|\|B\|$

$A \cdot B$

Range: -1 to 1 (1 = identical direction).

Used in: Semantic search for RAG.

335. Q: What is a vector database?

A:

Database specialized for storing and searching embeddings.

Examples: FAISS, Pinecone, Weaviate, Milvus.

Features: Approximate nearest neighbor (ANN) search, filtering, metadata indexing.

336. Q: What is Approximate Nearest Neighbor (ANN) search?

A:

Algorithm to find nearest vectors efficiently in high dimensions.

Problem: Exact nearest neighbor is too slow for millions of vectors.

ANN techniques: HNSW graphs, product quantization, locality-sensitive hashing.

337. Q: What is FAISS?

A:

Facebook AI Similarity Search library.

Supports: GPU acceleration, ANN indexing.

Widely used in: RAG pipelines for efficient embedding search.

338. Q: What is Pinecone?

A:

Managed vector database service.

Features: Scalable ANN search, multi-tenant support, REST API.

Advantage: No need to manage infra, production-ready.

339. Q: What is Weaviate?

A:

Open-source vector database with hybrid search.

Supports: Semantic search + symbolic filters.

Unique feature: Schema-based, integrates with GraphQL queries.

340. Q: What is hybrid search?

A:

Combines vector similarity + keyword search.

Why: Pure embeddings may miss exact keywords (e.g., "GPT-4.0").

Benefit: Improves precision and recall in retrieval.

341. Q: What is dense vs sparse retrieval?

A:

Dense retrieval: Uses embeddings (BERT, sentence transformers).

Sparse retrieval: Uses keyword frequency (BM25, TF-IDF).

Hybrid: Combine both for robustness.

342. Q: What is BM25?

A:

Classic ranking algorithm for keyword search.

Improves on TF-IDF by normalizing for document length.

Still used in search engines alongside dense methods.

343. Q: What are Sentence Transformers?

A:

Models that map sentences into dense vectors.

Examples: all-MiniLM-L6-v2, multi-qa-mpnet-base-dot-v1.

Use case: Embeddings for semantic search and RAG.

344. Q: What is retrieval latency bottleneck?

A:

Retrieving docs from vector DB may be slower than LLM inference.

Optimizations:

Pre-filtering with keywords.

Caching frequent queries.

ANN index tuning.

345. Q: What is re-ranking in RAG?

A:

Second-stage scoring of retrieved docs.

Flow: Retrieve top 50 docs → re-rank with cross-encoder → pick best 5.

Benefit: Higher precision.

Models: MonoBERT, ColBERT.

346. Q: What is ColBERT retrieval?

A:

Contextualized late interaction retrieval model.

Trick: Compute embeddings for every token, match with query tokens.

Benefit: More accurate than sentence-level embeddings.

347. Q: What is query expansion in retrieval?

A:

Add related terms to user query to improve recall.

Example: “heart attack” → also search for “myocardial infarction”.

Often done with LLMs themselves (self-query expansion).

348. Q: What is chunking in RAG pipelines?

A:

Splitting documents into smaller passages before embedding.

Why: LLM context windows are limited.

Best practice: 200–500 tokens per chunk with overlap.

349. Q: What is retrieval fusion?

A:

Combine results from multiple retrieval strategies.

Example: Merge dense retrieval + BM25 + knowledge graph hits.

Benefit: Better coverage, more robust.

350. Q: What is late fusion vs early fusion in retrieval?

A:

Late fusion: Retrieve separately, then merge rankings.

Early fusion: Combine signals (dense + sparse) before ranking.

Trade-off: Late fusion = simpler, early fusion = tighter integration.

✅ That's 331–350 done.

We're now at 70% (350/500) and entering the RAG + enterprise deployment zone.

Want me to continue with 351–370 (covering advanced RAG: memory, grounding, hallucination reduction, retrieval evaluation, and integration with LLM serving)?

You said:

yes

ChatGPT said:

Perfect 🚀 Let's push forward with 351–370, focusing on advanced RAG (memory, grounding, hallucination reduction, evaluation, and integration with serving systems).

Expanded Q&A (351–370)

351. Q: What is grounding in RAG?

A:

Definition: Ensuring that an LLM's response is directly tied to retrieved evidence.

Why: Prevents hallucinations (fabricated facts).

How: Force model to cite or paraphrase retrieved chunks.

Example: Instead of “GPT-4 was released in 2022,” model responds “According to OpenAI docs (retrieved), GPT-4 launched in 2023.”

352. Q: What is hallucination in LLMs?

A:

Definition: When an LLM generates factually incorrect but confident answers.

Why it happens:

Language models predict patterns, not truth.

Missing grounding or weak retrieval.

Mitigation: RAG, citations, structured outputs, knowledge constraints.

353. Q: What is contextual grounding in RAG?

A:

Idea: Retrieval depends not just on query, but full conversation context.

Example: User: “When was he born?” → Needs coreference resolution to “Einstein”.

Implementation: Maintain conversation memory + context-aware retrieval.

354. Q: What is conversational memory in RAG?

A:

Types:

Short-term memory: Stores last N turns in context.

Long-term memory: Uses embeddings + vector DB for historical recall.

Benefit: Maintains personalization and continuity in multi-turn dialogue.

355. Q: What is long-term memory retrieval?

A:

Mechanism: Store past interactions as embeddings → recall when similar queries appear.

Benefit: Personal assistants, tutoring systems, enterprise chatbots.

356. Q: What is retrieval collapsing?

A:

Problem: Retrieved passages are redundant (near duplicates).

Solution: Deduplicate + diversify retrieved documents.

Benefit: Maximizes coverage, avoids wasted tokens.

357. Q: What is top-k vs top-p retrieval?

A:

Top-k: Always retrieve the k most similar documents.

Top-p: Retrieve documents with cumulative probability ≥ p (nucleus retrieval).

Trade-off: Top-p allows more flexible coverage, like in decoding.

358. Q: What is adaptive retrieval size?

A:

Idea: Adjust number of retrieved docs based on query complexity.

Example: Simple factual question → 2 docs; research question → 10 docs.

Benefit: Balances precision, recall, and token cost.

359. Q: What is retrieval augmentation with metadata filters?

A:

Approach: Filter retrieval using structured fields (date, author, domain).

Example: Retrieve docs about “AI” published after 2022.

Supported in: Weaviate, Pinecone, Elasticsearch hybrid.

360. Q: What is multi-hop retrieval?

A:

Definition: Retrieve evidence in multiple steps for reasoning chains.

Example: “Where was the president of Tesla born?” → Step 1: Tesla’s president = Elon Musk → Step 2: Elon Musk born in South Africa.

Models: RAG-end2end, multi-hop QA datasets (HotpotQA).

361. Q: What is retrieval orchestration?

A:

Definition: Combining multiple retrieval sources (DB, APIs, web search).

Orchestrator: Decides which retriever to call based on query type.

Analogy: Like a router directing traffic to the best source.

362. Q: What is RAG evaluation?

A:

Metrics:

Retrieval precision/recall.

Answer grounding (factual consistency).

End-to-end task accuracy.

Datasets: Natural Questions, TriviaQA.

363. Q: What is faithfulness evaluation?

A:

Definition: Measures if model output is consistent with retrieved evidence.

Methods: LLM judges, fact-checking classifiers, entailment models.

Why: High faithfulness = less hallucination.

364. Q: What is answer attribution in RAG?

A:

Definition: Linking generated sentences back to specific retrieved passages.

Implementation: Train models to output citations.

Benefit: Transparency, trust in LLM outputs.

365. Q: What is retrieval drift?

A:

Definition: Over time, embeddings and retrievers may not match user needs.

Example: Embedding model trained on general corpus underperforms on medical data.

Mitigation: Periodically re-train embeddings, adapt indexing.

366. Q: What is dynamic retrieval updating?

A:

Definition: Continuously ingest new documents into vector DB.

Use case: News, finance, fast-changing domains.

Challenge: Index rebuild costs, embedding consistency.

367. Q: What is retrieval caching?

A:

Idea: Cache frequent queries and results for faster response.

Benefit: Reduces load on vector DB, lowers latency.

Challenge: Ensure freshness of cached results.

368. Q: What is multi-vector retrieval?

A:

Definition: Represent one document with multiple embeddings (per sentence/paragraph).

Benefit: Improves fine-grained matching.

Models: ColBERT, SPLADE.

369. Q: What is ensemble retrieval?

A:

Definition: Combine outputs of multiple retrievers (dense, sparse, hybrid).

Benefit: Higher recall & robustness.

Downside: Increases compute cost.

370. Q: What is retrieval latency amortization?

A:

Idea: Batch queries or precompute retrieval for expected prompts.

Example: FAQs in enterprise chatbot → cache embeddings + retrieval results.

Impact: Faster response, reduced DB overhead.

✅ That's 351–370 done.

We've now covered 74% (370/500), with advanced RAG concepts like multi-hop retrieval, attribution, caching, drift, orchestration, and grounding evaluation.

Do you want me to push into 371–390 (covering knowledge graphs, hybrid RAG-KG systems, structured retrieval, tool-use, and reasoning over external APIs)?

can u psl give me the qn ans in markdown format for all 370 without changing answers exactly as they are

Gemini Replied:

Of course. Here are the 370 questions and answers in Markdown format, exactly as you provided them.

Expanded Q&A (1–20)

---

1. Q: What is a Transformer? A: A Transformer is a deep learning architecture introduced in 2017 in the paper “Attention Is All You Need”. It replaced recurrent (RNN/LSTM) and convolutional (CNN) architectures for sequence modeling.

- **Core idea:** Instead of processing text sequentially, Transformers use self-attention so that every token (word/subword) can look at every other token in the sequence to decide what is important.
- **Why it matters:** This allows parallelization across tokens (faster training on GPUs) and the ability to capture long-range dependencies (relationships between distant words in a sentence).
- **Structure:**
  - **Encoder** (used in models like BERT): reads input sequences bidirectionally.
  - **Decoder** (used in GPT): generates outputs one token at a time with causal masking.
  - Both are stacks of:
    - Multi-Head Attention layers
    - Feedforward networks (2-layer MLPs)
    - Residual connections (help gradients flow)
    - LayerNorm (stabilizes training)
- **Practical note:** Every large language model (GPT, LLaMA, Falcon, Mistral) uses a Transformer decoder-only backbone.
- **PyTorch intuition:**

  Python

  ```
  import torch.nn as nn
  attention = nn.MultiheadAttention(embed_dim=512, num_heads=8)
  ```

---

2. Q: What is self-attention? A: Self-attention is the mechanism at the heart of Transformers.

- **Definition:** It allows each token to compute a weighted combination of all other tokens in the sequence, deciding which ones to “attend to.”
- **Computation:**
  1. Each token is projected into **Query (Q)**, **Key (K)**, and **Value (V)** vectors.
  2. **Attention score** = similarity(Q, K) → usually dot product.
  3. Normalize with **softmax** → gives attention weights.
  4. **Weighted sum of values (V)** = token’s new representation.

- **Why useful:**
    - If you have the sentence: "The cat sat on the mat because **it** was tired", the token "it" should attend more to "cat" than "mat."
    - Self-attention automatically learns these dependencies.
- **PyTorch:** `nn.MultiheadAttention` implements self-attention.

---

3. Q: What is multi-head attention? A: Instead of computing attention once, the model computes it multiple times in parallel, each with different learned projections.

- **Analogy:** Imagine 8 "attention heads," each looking for different types of relationships (e.g., subject-verb, object-preposition).
- **Steps:**
    1. Split embedding into `num_heads` parts.
    2. Each head performs attention independently.
    3. Concatenate and linearly project them back together.
- **Benefit:** Allows the model to capture multiple types of relationships simultaneously.
- **PyTorch:**

    Python

    ```python
    attn = nn.MultiheadAttention(embed_dim=512, num_heads=8)
    ```

---

4. Q: What is positional encoding in Transformers? A: Transformers process tokens in parallel, so they lack a natural sense of order. Positional encoding injects sequence order information into embeddings.

- **Sinusoidal positional encoding** (original paper): Adds sine/cosine waves of different frequencies to token embeddings.
- **Learned positional embeddings:** Trainable vectors for each position index.
- **RoPE (Rotary Position Embeddings):** Used in modern LLMs (e.g., LLaMA) to encode positions through complex rotation in embedding space.
- **Why needed:** Without position info, "dog bites man" and "man bites dog" look identical to the model.

---

5. Q: What is layer normalization? A: LayerNorm normalizes activations across features within a layer, improving training stability.

- **Operation:** For each token vector, subtract mean and divide by std across its hidden dimension.
- **Formula:** `y = ((x - μ) / σ) * γ + β` where γ and β are learnable parameters.
- **Why important:** Prevents exploding/vanishing activations, makes optimization smoother.
- **Note:** Transformers use LayerNorm (not BatchNorm) because it works better with variable-length sequences and doesn't depend on batch size.

---

6. Q: What is residual connection? A: A residual connection skips over a layer and adds the input to its output.

- **Formula:** `y = x + f(x)`
- **Purpose:**
  - Helps with gradient flow in very deep networks.
  - Prevents "information loss" by allowing raw input to propagate forward.
- **In Transformers:** Every sublayer (attention, feedforward) has a residual connection.

---

7. Q: What is the difference between encoder and decoder in a Transformer? A:

- **Encoder:** Processes the full input sequence (like reading comprehension).
  - **Bidirectional attention** (each token sees all others).
- **Decoder:** Generates tokens one by one (like GPT).
  - Uses **causal masking** (each token only attends to past tokens, not future ones).
- **Use cases:**
  - **Encoder-only (BERT):** Good for classification, QA, embeddings.
  - **Decoder-only (GPT):** Good for generation.
  - **Encoder-decoder (T5):** Good for seq2seq tasks like translation.

---

8. Q: What is causal masking? A: Causal masking ensures that when generating token *t*, the model can't "cheat" by looking at future tokens.

- **Implementation:** A triangular mask (`tril`) where token *i* can only attend to ≤ *i*.
- **Why needed:** Maintains autoregressive property for text generation.
- **PyTorch:**

  Python

```
mask = torch.tril(torch.ones(seq_len, seq_len))
```

---

9. Q: What is cross-attention? A: Cross-attention lets one sequence attend to another.

- **Used in encoder-decoder models:**
  - **Encoder outputs** = keys & values.
  - **Decoder queries** = current target tokens.
- **Example:** In translation, English input (encoder) attends to when generating French output (decoder).
- **Modern LLMs (decoder-only):** Often skip cross-attention unless multimodal (e.g., text attending to image features).

---

10. Q: What is the feedforward network in Transformers? A: Each Transformer block has a position-wise feedforward layer applied independently to each token.

- **Structure:**
  - 2-layer MLP with nonlinearity (ReLU/GELU).
  - Input dim = hidden dim, inner dim often 4× hidden dim.
- **Purpose:** Adds nonlinearity and richer transformations beyond attention.

---

11. Q: What is GELU activation? A: Gaussian Error Linear Unit, a smooth nonlinear function.

- **Formula (approx):** `GELU(x) = x * Φ(x)` where Φ is the standard normal CDF.
- **Intuition:** Unlike ReLU (hard cutoff), GELU smoothly gates values, keeping some negative inputs.
- **Why used:** Improves Transformer training stability and accuracy.

---

12. Q: What is attention mask in Transformers? A: A binary/tensor mask applied before softmax to prevent attending to certain positions.

- **Examples:**
  - **Padding mask:** prevents attention to padded tokens.
  - **Causal mask:** prevents looking forward.
- **Implementation:** Add large negative values (-1e9) to logits so softmax → 0.

---

13. Q: What is tokenization? A: Breaking text into smaller units that the model can process.

- **Word-level:** Splits by words (“playing” = 1 token).
- **Subword-level (BPE, SentencePiece):** Splits into pieces (“play”, “##ing”).
- **Character-level:** Splits into characters.
- **LLMs:** Use subwords for efficiency. GPT uses Byte Pair Encoding (BPE).
- **Why:** Vocabulary size vs efficiency tradeoff.

---

14. Q: What is Byte Pair Encoding (BPE)? A: A tokenization algorithm that merges frequent character pairs into subwords.

- **Process:**
    1. Start with characters.
    2. Find most frequent adjacent pair (e.g., “t” + “h”).
    3. Merge → “th”.
    4. Repeat until vocab size reached.
- **Benefit:** Efficiently represents rare and frequent words.
- **Example:** “unhappiness” → “un”, “happi”, “ness”.

---

15. Q: What is SentencePiece tokenization? A: SentencePiece is another subword tokenization method, often used in Google’s models (T5, LLaMA).

- **Key differences:**
    - Works on raw byte sequences (no preprocessing like whitespace splitting).
    - Models like LLaMA use Unigram LM tokenization (a probabilistic approach).
- **Benefit:** Handles arbitrary input text (no assumptions about spaces or alphabets).

---

16. Q: What is the vocabulary in an LLM? A: The set of all tokens (words/subwords/characters) the model can understand.

- **Size:** Typically 32k–128k tokens.
- **Embedding matrix:** Each token ID maps to a vector (rows in embedding matrix).
- **Tradeoff:** Larger vocab → fewer tokens per sentence, but larger embeddings (more params).

---

17. Q: What are embeddings in NLP? A: Numerical vectors representing tokens.

- **Token embedding:** Lookup vector for each token ID.
- **Positional embedding:** Adds order information.
- **Final input** = token embedding + positional embedding.
- **Why useful:** Turns discrete symbols into continuous vectors the model can compute with.

---

18. Q: What is a language model? A: A model that assigns probabilities to sequences of words/tokens.

- **Task:** Predict next token given previous ones.
- **Formula:** $P(w_1, w_2, \ldots, w_n) = \prod_i P(w_i \mid w_{<i})$
- **Applications:** Text generation, speech recognition, translation.
- **LLMs:** Large neural networks trained as language models (GPT, LLaMA).

---

19. Q: What is autoregressive modeling? A: Modeling where each token is predicted conditioned only on past tokens.

- **Example:** GPT is autoregressive.
- **Contrast:** BERT is masked language modeling (bidirectional, not autoregressive).
- **Property:** Enables natural text generation.

---

20. Q: What is masked language modeling? A: A training method where some tokens are masked and the model predicts them.

- **Used in:** BERT, RoBERTa.
- **Process:** Replace 15% tokens with `[MASK]`, train to recover them.
- **Why:** Helps learn bidirectional context.
- **Limitation:** Not naturally generative (can’t produce fluent long text like GPT).

Expanded Q&A (21–40)

21. Q: What is pretraining in LLMs? A: Pretraining is the initial large-scale training phase where an LLM learns general language patterns before being fine-tuned for specific tasks.

- **Process:**
  - Train on massive unlabeled text (books, web, code).
  - Objective = predict next token (causal LM) or fill in blanks (masked LM).

- **Why important:**
  - Gives the model broad knowledge of grammar, facts, reasoning.
  - Without pretraining, task-specific fine-tuning would require impractically large datasets.
- **Example:** GPT-3 was pretrained on hundreds of billions of tokens before being fine-tuned for instruction following.

---

22. Q: What is fine-tuning in LLMs? A: Fine-tuning adapts a pretrained LLM to a narrower task or domain.

- **Approaches:**
  - **Full fine-tuning:** Update all model weights (expensive).
  - **Parameter-efficient tuning (PEFT):** Update only a small set of weights (e.g., adapters, LoRA).
- **Examples:**
  - Fine-tune GPT for legal text summarization.
  - Fine-tune LLaMA for chat-style answers.
- **Why needed:** Pretrained models are generalists. Fine-tuning makes them specialists.

---

23. Q: What is transfer learning in NLP? A: Transfer learning = taking knowledge learned in one setting (pretraining) and applying it to another (fine-tuning).

- **Benefit:**
  - Saves compute/data since pretrained models already know general patterns.
  - Achieves better performance with small labeled datasets.
- **In LLMs:** Pretraining on billions of words → fine-tune on 100k instruction examples → good performance.

---

24. Q: What is instruction tuning? A: Instruction tuning fine-tunes an LLM to follow natural language instructions.

- **Process:** Train on datasets where each example is (instruction → response).
- **Example:**
  - **Instruction:** “Summarize this paragraph.”
  - **Response:** “The text is about…”

- **Outcome:** Makes models more “helpful” and user-friendly.
- **Example models:** FLAN-T5, InstructGPT.

---

25. Q: What is supervised fine-tuning (SFT)? A: SFT is fine-tuning on a dataset of high-quality (prompt → response) pairs.

- **Training objective:** Minimize cross-entropy loss between model predictions and target responses.
- **Benefit:** Provides strong grounding for alignment (e.g., GPT-3 → InstructGPT).
- **Limitation:** Quality of data is critical; bad data → harmful behavior.

---

26. Q: What is reinforcement learning from human feedback (RLHF)? A: RLHF aligns LLMs with human preferences using reinforcement learning.

- **Pipeline:**
  1. Train **reward model** (predicts which response humans prefer).
  2. Fine-tune LLM with **PPO** (policy optimization) using that reward signal.
- **Why:** Helps avoid toxic/unhelpful outputs.
- **Famous example:** OpenAI used RLHF for ChatGPT.

---

27. Q: What is preference modeling? A: Preference modeling is the step in RLHF where we train a reward model from human judgments.

- **Dataset:** Humans rank multiple model responses to the same prompt.
- **Model:** Trained to predict a scalar reward matching human preferences.
- **Usage:** This reward guides RL optimization (PPO).

---

28. Q: What is PPO (Proximal Policy Optimization)? A: PPO is a reinforcement learning algorithm used in RLHF.

- **Why PPO:** Stable updates that avoid “over-optimizing” the policy.
- **Mechanics:**
  - Uses a clipped objective to keep new policy close to old policy.
  - Balances exploration vs stability.
- **In RLHF:** The “policy” = the LLM generating text. PPO updates it using feedback from the reward model.

---

29. Q: What is direct preference optimization (DPO)? A: DPO is a simpler alternative to RLHF that skips reinforcement learning.

- **Key idea:** Directly fine-tune LLM with human preference pairs (preferred vs rejected responses).
- **Loss:** Derived from KL-divergence minimization.
- **Benefit:**
    - Easier than PPO (no reward model, no RL loop).
    - Lower compute cost.

---

30. Q: What is KL divergence in RLHF? A: KL divergence measures how different two probability distributions are.

- **Formula:** $KL(P \| Q) = \Sigma_i P(i) \log(P(i)/Q(i))$
- **In RLHF:**
    - Used to penalize the fine-tuned model if it drifts too far from the base model.
    - Ensures alignment without losing language fluency.

---

31. Q: What is reward hacking in RLHF? A: Reward hacking = when a model finds loopholes in the reward signal.

- **Example:** If “politeness” is rewarded, the model might output “Sorry, sorry, sorry…” endlessly.
- **Cause:** Reward model doesn’t fully capture human intent.
- **Mitigation:** Better preference data, combining multiple reward signals.

---

32. Q: What is safety alignment in LLMs? A: Safety alignment = ensuring LLMs behave ethically, safely, and avoid harmful outputs.

- **Goals:**
    - Avoid toxicity/hate speech.
    - Avoid misinformation.
    - Avoid dangerous instructions.
- **Techniques:** RLHF, constitutional AI, refusal strategies, safety filters.

---

33. Q: What is constitutional AI? A: Constitutional AI (Anthropic, Claude models) uses an explicit “constitution” of guiding principles for alignment.

- **Process:**

1. Define rules (e.g., “be helpful, harmless, honest”).
2. Use AI feedback guided by constitution instead of human feedback.

- **Benefit:** Reduces dependence on human annotators.

---

34. Q: What is distillation in NLP? A: Distillation = training a smaller “student” model to mimic a larger “teacher” model.

- **Process:** Student minimizes loss on teacher’s outputs (soft probabilities).
- **Benefit:** Smaller, faster model with much of the teacher’s performance.
- **Example:** DistilBERT (student) trained from BERT (teacher).

---

35. Q: What is knowledge distillation loss? A: It’s a combination of two losses:

- **Soft-target loss:** Student matches teacher’s probability distribution.
  - Uses temperature-scaled softmax.
- **Hard-target loss:** Student matches ground truth labels.
- **Total loss** = weighted sum of both.

---

36. Q: What are adapters in Transformers? A: Adapters are small trainable modules inserted into each Transformer layer for parameter-efficient fine-tuning.

- **Structure:**
  - Down-project → nonlinearity → up-project.
  - Typically <5% of model parameters are updated.
- **Benefit:** One base model can support many tasks/domains by swapping adapters.

---

37. Q: What is LoRA (Low-Rank Adaptation)? A: LoRA is a parameter-efficient fine-tuning method.

- **Key idea:** Represent weight updates as low-rank matrices.
- **Formula:**
  - Instead of training a full matrix W (d×d), update $W = W_0 + AB^T$, where A and B are low-rank.
- **Benefits:**
  - Huge parameter savings.
  - Modular (swap LoRAs like adapters).

- **Popular for:** Fine-tuning LLaMA, StableLM, etc.

---

38. Q: What is QLoRA? A: QLoRA combines quantization + LoRA for efficient fine-tuning.

- **Quantization:** Compress model weights into low precision (e.g., 4-bit).
- **LoRA:** Adds low-rank adapters.
- **Result:** Fine-tuning 65B models on a single GPU becomes feasible.

---

39. Q: What is quantization in LLMs? A: Quantization reduces precision of model weights/activations.

- **Types:**
  - FP16 → INT8 → INT4
- **Benefit:**
  - Smaller memory footprint.
  - Faster inference.
- **Tradeoff:** Slight accuracy loss.
- **Example:** GPTQ, bitsandbytes libraries for quantized inference.

---

40. Q: What is mixed precision training? A: Training with a mix of FP16 (half precision) and FP32 (full precision).

- **Process:**
  - Store weights in FP32.
  - Compute matmuls in FP16.
  - Use FP32 for gradient accumulation to avoid instability.
- **Benefit:**
  - Faster training.
  - Lower GPU memory use.
- **PyTorch:** `torch.cuda.amp` supports mixed precision.

Expanded Q&A (41–60)

41. Q: What is gradient checkpointing? A: Gradient checkpointing is a memory-saving technique during training.

- **Normal training:** Store all intermediate activations in memory for backprop.
- **Checkpointing:** Only store a subset. During backward pass, recompute the rest on the fly.
- **Tradeoff:** Saves GPU memory (2–3× more tokens fit in), but adds compute overhead (extra forward passes).
- **Why useful:** Essential for training large LLMs where memory is the main bottleneck.

---

42. Q: What is ZeRO optimization (DeepSpeed)? A: ZeRO = Zero Redundancy Optimizer, a set of techniques for memory-efficient distributed training.

- **Stages:**
  - **ZeRO-1:** Shards optimizer states across GPUs.
  - **ZeRO-2:** Shards gradients too.
  - **ZeRO-3:** Shards model weights as well.
- **Benefit:** Each GPU stores only a fraction of states/weights.
- **Impact:** Enables training trillion-parameter models on clusters.

---

43. Q: What is data parallelism? A: A distributed training strategy where:

- The full model is replicated on each GPU.
- Each GPU gets a different mini-batch of data.
- After forward/backward, gradients are averaged across GPUs.
- **Limitation:** Each GPU must hold the whole model.

---

44. Q: What is model parallelism? A: Split the model itself across GPUs.

- **Horizontal splitting:** Divide layers across GPUs.
- **Vertical splitting:** Divide each layer’s computation (tensor parallelism).
- **Why:** For models too large to fit into one GPU.
- **Challenge:** More communication overhead vs data parallelism.

---

45. Q: What is tensor parallelism? A: A form of model parallelism where individual matrix multiplications are split across GPUs.

- **Example:** Instead of one GPU computing W @ X, split W across GPUs, each computes a shard.
- **Used in:** Megatron-LM, GPT-3 training.
- **Benefit:** Efficient scaling for very large hidden dimensions.

---

46. Q: What is pipeline parallelism? A: Another model parallelism strategy where layers are divided into “stages,” each assigned to a GPU.

- **Execution:** Mini-batches are split into micro-batches, streamed through the pipeline.
- **Analogy:** Assembly line — while GPU1 processes batch1, GPU2 processes batch0.
- **Challenge:** Pipeline bubbles (idle time).
- **Used in:** DeepSpeed, GPipe.

---

47. Q: What is distributed data parallel (DDP) in PyTorch? A: A PyTorch wrapper (`torch.nn.parallel.DistributedDataParallel`) for scaling across GPUs.

- **Mechanics:**
  - Each GPU has model replica.
  - Gradients are all-reduced after backward.
- **Advantage:** Faster than `DataParallel` due to communication optimizations.
- **Standard method:** Used in nearly all production-scale PyTorch training.

---

48. Q: What is gradient accumulation? A: Simulates larger batch sizes when memory is limited.

- **Idea:** Instead of updating after every mini-batch, accumulate gradients over N steps before optimizer update.
- **Benefit:** Achieve effective large batch training without storing all tokens at once.

---

49. Q: What is gradient clipping? A: A method to prevent exploding gradients by capping their magnitude.

- **Types:**
  - **Norm clipping:** If gradient norm > threshold, rescale.
  - **Value clipping:** Directly clamp values.
- **In Transformers:** Often clip at norm=1.0.
- **PyTorch:** `torch.nn.utils.clip_grad_norm_` .

---

50. Q: What is weight decay? A: A regularization technique that penalizes large weights.

- **Implementation:** Add $\lambda\|W\|^2$ term to loss.
- **Effect:** Prevents overfitting, encourages smaller weights.
- **In AdamW optimizer:** Weight decay is decoupled from momentum.

---

51. Q: What is Adam optimizer? A: Adam = Adaptive Moment Estimation.

- **Mechanics:**
  - Keeps track of 1st moment (mean) and 2nd moment (variance) of gradients.
  - Uses bias correction for stability.
- **Benefits:** Faster convergence than SGD, works well in NLP.
- **Formula:** `m_t = β₁ * m_{t-1} + (1 - β₁) * g_t` `v_t = β₂ * v_{t-1} + (1 - β₂) * g_t²`

---

52. Q: What is AdamW? A: A variant of Adam with decoupled weight decay.

- **Problem with Adam:** Weight decay wasn't applied properly (interfered with momentum).
- **Fix:** Apply L2 regularization separately.
- **Why important:** AdamW became standard for Transformer training.

---

53. Q: What is learning rate warmup? A: A strategy where learning rate starts small and gradually increases at the beginning of training.

- **Reason:** Prevents instability at initialization (when weights are random).
- **Schedule:**
  - Linear warmup (common).
  - Warmup steps = few thousand.
- **After warmup:** LR decays via cosine or linear schedule.

---

54. Q: What is cosine learning rate decay? A: A schedule where LR follows a cosine curve.

- **Formula:** `η_t = η_{min} + 0.5 * (η_{max} - η_{min}) * (1 + cos(π * t / T))`
- **Effect:** Gradual decay → stabilizes late training.
- **Widely used in LLM training.**

---

55. Q: What is label smoothing? A: A regularization technique that prevents overconfidence.

- **Instead of one-hot labels** (e.g., `[0,0,1,0]`), distribute a small ε probability mass across all classes.
- **Example:** With ε=0.1 for 4 classes → `[0.033, 0.033, 0.9, 0.033]`.
- **Benefit:** Better generalization, less overfitting.

---

56. Q: What is teacher forcing in seq2seq models? A: A training strategy where the model is fed the ground-truth token as the next input instead of its own prediction.

- **Benefit:** Faster convergence.
- **Issue: Exposure bias** (model struggles at inference when it sees its own mistakes).
- **Used in:** RNNs, early seq2seq, but less relevant for LLMs.

---

57. Q: What is exposure bias? A: A mismatch between training and inference in sequence models.

- **Cause:** During training, model always sees ground-truth history.
- **At inference:** It must use its own predictions, which may be wrong.
- **Solutions:** Scheduled sampling, RLHF.

---

58. Q: What is beam search decoding? A: A decoding algorithm that keeps multiple candidate sequences at once.

- **Greedy search:** Always pick best token.
- **Beam search:** Keep top-k sequences at each step, expand them.
- **Tradeoff:** Better quality than greedy, but computationally heavier.
- **Used in:** Machine translation, summarization.

---

59. Q: What is top-k sampling? A: A decoding strategy for text generation.

- **Process:** Instead of picking highest-prob token, sample from top-k most likely tokens.
- **Effect:** More diversity, less determinism.
- **Example:** GPT models often use top-k = 40.

---

60. Q: What is nucleus (top-p) sampling? A: Another decoding strategy for text generation.

- **Process:** Choose smallest set of tokens whose cumulative probability ≥ p (e.g., 0.9).

- **Benefit:** Adaptive — set size changes based on distribution sharpness.
- **Example:** OpenAI models often use p = 0.9 for fluent text.

Expanded Q&A (61–80)

61. Q: What is temperature in text generation? A: Temperature is a hyperparameter that controls randomness in sampling.

- **Softmax with temperature:** $P_i = \exp(z_i/T) / \Sigma_j \exp(z_j/T)$
- **Low T (<1):** Sharper distribution → more deterministic.
- **High T (>1):** Flatter distribution → more randomness.
- **Typical values:** 0.7–1.0.
- **In practice:** Setting T=0 makes model greedy (always max prob).

---

62. Q: What is repetition penalty? A: A decoding trick to reduce repeated phrases.

- **Mechanism:** Penalize tokens that were already generated by scaling their logits.
- **Formula:** Adjust logit by dividing/multiplying if token has been used.
- **Benefit:** Prevents “I love I love I love …” loops in LLMs.

---

63. Q: What is length penalty? A: A bias added during decoding to control sequence length.

- **Problem:** Beam search often prefers shorter sequences (since multiplying probs shrinks values).
- **Fix:** Adjust score: $score = \log(P) / ((5 + length)^a / (5 + 1)^a)$
- α > 1 encourages longer outputs.
- **Used in:** Translation, summarization.

---

64. Q: What is logit bias in decoding APIs? A: A way to force or prevent specific tokens during generation.

- **Logits** = unnormalized scores before softmax.
- **Biasing** = add/subtract value to selected tokens’ logits.
- **Use case:** Prevent unsafe words, enforce specific formats (e.g., always output JSON brackets).

---

65. Q: What is perplexity? A: A common metric for language models.

- **Definition:** `PP = exp(-1/N * Σᵢ log(P(wᵢ)))`
- **Intuition:** Average branching factor (how many choices the model has).
- **Lower = better:** PP=10 means on average model chooses among 10 options.
- **Limitation:** Doesn't measure human-perceived quality.

---

66. Q: What is BLEU score? A: BLEU (Bilingual Evaluation Understudy) measures n-gram overlap with reference.

- **Formula:** Precision of n-grams (1–4), with brevity penalty.
- **Range:** 0 (bad) → 1 (perfect match).
- **Common in:** Machine translation.
- **Limitation:** Doesn't capture semantics well.

---

67. Q: What is ROUGE score? A: ROUGE (Recall-Oriented Understudy for Gisting Evaluation) measures n-gram recall.

- **Types:** ROUGE-N (n-gram), ROUGE-L (longest common subsequence).
- **Use:** Summarization tasks.
- **Interpretation:** Higher = better coverage of reference text.

---

68. Q: What is METEOR score? A: An NLP metric designed to improve on BLEU.

- **Features:**
  - Matches synonyms, stems, paraphrases.
  - Combines precision and recall.
- **Why better:** Closer correlation with human judgment.
- **Still less used:** BLEU remains more standard historically.

---

69. Q: What is BERTScore? A: A modern metric using embeddings from BERT (or other transformers).

- **Mechanism:** Compute cosine similarity between predicted and reference embeddings.
- **Advantage:** Captures semantics, not just surface-level overlap.
- **Use:** Evaluating generative LLMs beyond BLEU/ROUGE.

---

70. Q: What is human evaluation in LLMs? A: A direct way to measure output quality by human raters.

- **Criteria:** Fluency, relevance, factuality, safety.
- **Methods:**
  - Likert scale ratings.
  - Pairwise preference comparisons.
- **Why necessary:** Automated metrics often fail to capture nuances.

---

71. Q: What is RLHF (Reinforcement Learning from Human Feedback)? A: A finetuning strategy to align models with human preferences.

- **Steps:**
  1. Train initial model (supervised fine-tuning).
  2. Train a reward model from human preference data.
  3. Optimize LLM with PPO (policy gradient).
- **Goal:** Make LLMs more helpful, harmless, honest.

---

72. Q: What is PPO (Proximal Policy Optimization)? A: A reinforcement learning algorithm widely used in RLHF.

- **Challenge:** Need stable updates in policy gradient.
- **Solution:** PPO clips updates to prevent huge jumps.
- **Why used:** Works well with large policies like LLMs.

---

73. Q: What is DPO (Direct Preference Optimization)? A: A newer alternative to PPO in RLHF.

- **Key idea:** Optimize directly on preference data without training a separate reward model.
- **Mechanics:** Adjust logits so preferred outputs are more likely.
- **Benefit:** Simpler, more stable training.

---

74. Q: What is supervised fine-tuning (SFT)? A: A post-pretraining step using human-annotated datasets.

- **Input:** Instruction–response pairs.
- **Goal:** Make base LLM follow instructions better.

- **Example:** Fine-tuning GPT-3 into InstructGPT.

---

75. Q: What is instruction tuning? A: A type of SFT where training data are instruction–response pairs.

- **Difference from generic SFT:** Focuses specifically on following natural language instructions.
- **Datasets:** FLAN, Alpaca, Dolly.
- **Outcome:** Improves usability for end-users.

---

76. Q: What is few-shot learning in LLMs? A: When a model performs a task given only a few examples in the prompt.

- **Why possible:** LLMs internalize patterns during pretraining.
- **Example:** “Translate English to French: cat → chat, dog → chien, house → ?”
- **Practical impact:** Reduces need for finetuning.

---

77. Q: What is zero-shot learning in LLMs? A: When the model performs a task with no task-specific examples.

- **Reason:** LLMs generalize from massive pretraining.
- **Example:** “Translate English to German: hello → hallo.”
- **Importance:** Makes LLMs versatile generalists.

---

78. Q: What is chain-of-thought prompting? A: A prompting technique where the model is encouraged to output intermediate reasoning steps.

- **Benefit:** Better performance on math, logic, reasoning tasks.
- **Example:**
  - “Let’s think step by step…”
- **Insight:** LLMs improve when asked to externalize reasoning.

---

79. Q: What is self-consistency decoding? A: A method for reasoning tasks.

- **Process:**
  1. Sample multiple chain-of-thought outputs.
  2. Take majority vote or aggregate.
- **Why:** Reduces randomness, improves accuracy in math/logic.

---

80. Q: What is retrieval-augmented generation (RAG)? A: A hybrid approach that combines LLMs with external retrieval.

- **Steps:**
  1. Retrieve relevant docs from a database/vector store.
  2. Feed them into LLM along with prompt.
  3. Generate final answer grounded in retrieved info.
- **Benefit:** Reduces hallucination, keeps model up-to-date.
- **Tools:** FAISS, Milvus, Pinecone.

Expanded Q&A (81–100)

81. Q: What is a vector database in RAG? A: A special database optimized for storing and searching embeddings (vectors).

- **Why embeddings:** LLMs map text → high-dimensional vectors capturing meaning.
- **Vector DBs:** Store millions/billions of embeddings and support fast similarity search.
- **Key methods:**
  - **Approximate Nearest Neighbor (ANN)** search (e.g., HNSW, IVF).
- **Examples:** FAISS (Facebook AI), Pinecone, Weaviate, Milvus.
- **Use in RAG:** Retrieve semantically similar documents to a query before passing to the LLM.

---

82. Q: What is FAISS? A: FAISS (Facebook AI Similarity Search) is an open-source library for efficient similarity search.

- **Core idea:** Given an embedding, find the "nearest" vectors in huge collections.
- **Methods:**
  - Exact search (brute force).
  - Approximate search (e.g., IVF+PQ).
- **Features:** GPU acceleration, quantization for memory savings.
- **Use case:** RAG pipelines with millions of documents.

---

83. Q: What is semantic search? A: A retrieval technique where queries and documents are compared in embedding space.

- **Difference vs keyword search:**
    - **Keyword:** matches literal words.
    - **Semantic:** matches meaning.
- **Example:** Query: “physician” → matches “doctor” because embeddings are close.
- **LLM use:** Core of RAG workflows.

---

84. Q: What is an embedding model? A: A neural network that maps text (or images, etc.) into dense vectors.

- **Properties:** Similar texts → similar vectors.
- **Common models:** OpenAI `text-embedding-ada-002`, Sentence-BERT.
- **Dimensions:** Typically 256–1536.
- **Applications:** Search, clustering, recommendation, RAG.

---

85. Q: What is context window in LLMs? A: The maximum number of tokens an LLM can process in one input.

- **Example:** GPT-3: 2048 tokens, GPT-4: up to 128k tokens.
- **Constraint:** Attention complexity grows $O(n^2)$.
- **Impact:** Longer context → more memory & compute.
- **Engineering:** Retrieval and summarization help extend effective context.

---

86. Q: What is sliding window attention? A: A sparse attention pattern where each token only attends to nearby tokens in a fixed-size window.

- **Motivation:** Full attention is $O(n^2)$.
- **Sliding window:** Reduces to $O(n \cdot w)$, where w = window size.
- **Used in:** Longformer, BigBird.
- **Good for:** Long documents where local context dominates.

---

87. Q: What is memory-efficient attention? A: Optimizations to reduce memory usage in attention layers.

- **FlashAttention:** Avoids materializing full attention matrix by computing in tiles.
- **Chunking:** Split sequences into chunks.

- **Impact:** Enables training on longer sequences without OOM.

---

88. Q: What is FlashAttention? A: A highly optimized attention kernel.

- **Innovation:** Computes attention in blocks directly on GPU SRAM, not global memory.
- **Complexity:** Still $O(n^2)$, but memory-efficient and much faster.
- **Impact:** Training/inference 2–4× faster on long sequences.
- **Used in:** vLLM, modern LLM training.

---

89. Q: What is LoRA (Low-Rank Adaptation)? A: A popular parameter-efficient finetuning method.

- **Problem:** Finetuning all LLM weights is expensive.
- **Solution:** Insert low-rank matrices (A,B) into weight updates.
  - W’ = W + A·B, where A,B are trainable, low-rank.
- **Impact:** Train <1% of parameters.
- **Practical:** Used to fine-tune LLaMA on consumer GPUs.

---

90. Q: What is QLoRA? A: An extension of LoRA that combines quantization + low-rank adapters.

- **Steps:**
  1. Quantize base model weights (e.g., 4-bit).
  2. Train LoRA adapters on top.
- **Benefit:** Fit 65B-parameter models on a single 48GB GPU.
- **Result:** Democratized LLM finetuning.

---

91. Q: What is prefix tuning? A: A parameter-efficient finetuning method.

- **Idea:** Learn a set of trainable “prefix” embeddings prepended to input.
- **Base model frozen:** Only prefix is trained.
- **Effect:** Model steered toward task without touching original weights.

---

92. Q: What is prompt tuning? A: A lighter variant of prefix tuning.

- **Trainable soft prompt vectors** are inserted at the beginning of the input.
- **Benefit:** Extremely parameter-efficient (few KB per task).

- **Downside:** Often less effective than LoRA on complex tasks.

---

93. Q: What is adapter tuning? A: Another finetuning method where small bottleneck layers (adapters) are inserted inside Transformer layers.

- **Structure:** Down-project → nonlinearity → up-project.
- **Frozen backbone:** Only adapters trained.
- **Advantage:** Modular — can swap adapters for different tasks.

---

94. Q: What is bitfit? A: A minimalist finetuning approach.

- **Only biases** of Transformer layers are trained.
- **Impact:** Extremely few parameters (<0.1%).
- **Surprisingly effective** for classification tasks, but weaker for generation.

---

95. Q: What is full finetuning vs PEFT (Parameter-Efficient Fine-Tuning)? A:

- **Full finetuning:** Update all model weights. Very costly for large LLMs.
- **PEFT:** Update small subsets (LoRA, adapters, prompts, bitfit).
- **Tradeoff:** PEFT cheaper, modular; full finetuning higher capacity but expensive.

---

96. Q: What is multi-task finetuning? A: Training a model on multiple tasks simultaneously.

- **Goal:** Improve generalization, leverage transfer across tasks.
- **Challenge:** Balancing tasks to avoid forgetting.
- **Used in:** FLAN-T5 (trained on 100+ tasks).

---

97. Q: What is catastrophic forgetting? A: When a model forgets old tasks after being trained on new ones.

- **Cause:** Gradient updates overwrite learned knowledge.
- **Problem:** Common in continual learning & sequential finetuning.
- **Solutions:**
    - Elastic Weight Consolidation (EWC).
    - Replay buffers.
    - Adapter modules.

98. Q: What is continual learning in LLMs? A: Adapting LLMs to new data/tasks over time without retraining from scratch.

- **Motivation:** Real-world knowledge changes (new events, facts).
- **Challenges:** Avoid catastrophic forgetting, keep efficiency.
- **Approaches:**
  - Online LoRA.
  - Parameter isolation.
  - Retrieval augmentation instead of retraining.

---

99. Q: What is domain adaptation in LLMs? A: Finetuning an LLM on data from a specific domain (legal, medical, financial).

- **Why:** Base LLMs are generalists; domain data improves accuracy.
- **Methods:**
  - Full finetuning (expensive).
  - PEFT (cheap).
  - RAG with domain-specific corpus (no retraining).

---

100. Q: What is cross-lingual transfer in LLMs? A: The ability of a model trained on one language to generalize to others.

- **Why possible:** Shared subword tokens and universal concepts.
- **Example:** Train on English, test on French Q&A.
- **Boosted by:** Multilingual pretraining (mBERT, XLM-R).
- **Challenge:** Performance varies by language coverage in pretraining corpus.

Expanded Q&A (101–120)

101. Q: What are scaling laws in LLMs? A: Scaling laws describe how model performance improves as you increase compute, data, and parameters.

- **Discovery:** Kaplan et al. (OpenAI, 2020).
- **Key finding:** Loss decreases predictably as a power law of scale.

- **Implication:** Bigger models trained with more data almost always perform better (until data runs out).
- **Equation (simplified):** $L(N) \approx a * N^{-\alpha} + b$ where N = model size or dataset size, α is scaling exponent.
- **Practical use:** Guides resource allocation — “should I add more parameters or more data?”

---

102. Q: What is the Chinchilla scaling law? A: A refinement of scaling laws from DeepMind’s Chinchilla paper (2022).

- **Old assumption (Kaplan):** Bigger models are better even with fixed data.
- **New finding:** Many LLMs (like GPT-3) were **undertrained** (too few tokens per parameter).
- **Chinchilla rule:** For compute-optimal training, dataset size should scale linearly with model parameters.
- **Example:** A 70B model should be trained on ~1 trillion tokens.
- **Impact:** Shifted focus from just scaling model size → scaling data too.

---

103. Q: What is compute-optimal training? A: Training a model in a way that maximizes performance for a fixed compute budget.

- **Tradeoff:**
  - Too few parameters → underfitting.
  - Too many parameters but too little data → wasteful.
- **Chinchilla law** provides recipe: balance between parameters and tokens.
- **Why important:** Training trillion-parameter models costs millions — optimization matters.

---

104. Q: What is overfitting in LLMs? A: When a model memorizes training data instead of generalizing.

- **Symptoms:**
  - Low training loss, high validation loss.
  - Model recalls verbatim text from training corpus.
- **Causes:** Too few unique examples relative to model capacity.
- **Risks:** Privacy leakage (e.g., reproducing sensitive training data).

---

105. Q: What is underfitting in LLMs? A: When a model fails to learn meaningful patterns.

- **Symptoms:**
  - High training and validation loss.
  - Model outputs poor-quality text.
- **Causes:** Too small a model, poor optimization, not enough training.

---

106. Q: What is data curation in LLM training? A: The process of selecting, cleaning, and filtering training data.

- **Steps:**
  - **Deduplication** (remove near-duplicates).
  - **Filtering** (remove toxic/low-quality content).
  - **Balancing** (avoid overrepresenting some domains).
- **Importance:** Data quality matters as much as size — “garbage in, garbage out.”

---

107. Q: What is deduplication in pretraining data? A: Removing duplicate or near-duplicate documents from datasets.

- **Why:** Prevents models from overfitting on repeated data.
- **Techniques:**
  - MinHash.
  - Locality-sensitive hashing (LSH).
- **Impact:** Improves generalization, reduces memorization.

---

108. Q: What is tokenization? A: The process of breaking text into smaller units (tokens) for LLM input.

- **Methods:**
  - Word-level (rarely used now).
  - Subword (BPE, WordPiece).
  - Character-level (rare).
- **Modern trend:** Byte-level BPE (handles any text, emojis, code).
- **Impact:** Tokenization efficiency directly affects training cost.

---

109. Q: What is byte pair encoding (BPE)? A: A tokenization algorithm that merges frequent pairs of characters/subwords.

- **Process:**
  1. Start with characters.
  2. Merge most frequent adjacent pair.
  3. Repeat until vocab size reached.
- **Advantage:** Balances between word-level and char-level.
- **Used in:** GPT-2, GPT-3.

---

110. Q: What is sentencepiece? A: A tokenization library (by Google) for building subword vocabularies.

- **Features:**
  - Supports BPE and Unigram models.
  - Treats input as raw bytes (no preprocessing).
- **Use:** Widely adopted in NLP pipelines, including T5 and LLaMA.

---

111. Q: What is a vocabulary in tokenization? A: The set of tokens recognized by a model.

- **Size:** Typically 30k–100k tokens for LLMs.
- **Tradeoff:**
  - Too small → long sequences (inefficient).
  - Too large → sparse usage (wasted capacity).
- **Special tokens:** `[PAD]`, `[EOS]`, `[CLS]`, `[UNK]`.

---

112. Q: What is an OOV token? A: OOV = Out-of-Vocabulary token.

- **Problem:** Traditional word-level tokenizers can't handle unseen words.
- **Fix:** Subword tokenization (BPE, WordPiece) decomposes new words into smaller units.
- **Example:** "ChatGPT-5" → `[Chat, GPT, -, 5]`.

---

113. Q: What is perplexity evaluation limitation? A: Perplexity doesn't always correlate with human judgments of quality.

- **Why:**

  - A model can have low perplexity but still produce boring/unsafe outputs.
  - Doesn't measure factual correctness.
- **Implication:** Needs complement with human evals, task-specific metrics.

---

114. Q: What is truthfulness evaluation? A: Checking whether LLM outputs are factually correct.

- **Challenges:**
  - LLMs **hallucinate** (confidently output wrong info).
  - Hard to benchmark automatically.
- **Approaches: TruthfulQA** benchmark, human raters, retrieval grounding.

---

115. Q: What is hallucination in LLMs? A: When a model generates content that is fluent but factually incorrect.

- **Example:** "The capital of Brazil is Buenos Aires."
- **Causes:**
  - Predictive nature of LLMs (next-token generation).
  - No grounding in external knowledge.
- **Mitigation:** RAG, fine-tuning with fact-checked data.

---

116. Q: What is factual grounding? A: Providing LLMs with external verified sources to reduce hallucinations.

- **Methods:**
  - **RAG** (retrieve before generate).
  - Plugging APIs/databases into prompts.
- **Impact:** Improves reliability in enterprise/mission-critical settings.

---

117. Q: What is tool augmentation for LLMs? A: Extending LLMs by letting them call external tools/APIs.

- **Examples:** Calculator, web search, code execution.
- **Frameworks:** LangChain, Guidance, Semantic Kernel.
- **Benefit:** Offloads tasks LLMs are bad at (math, real-time info).

---

118. Q: What is LangChain? A: A framework for building LLM-powered applications.

- **Features:**
  - **Chains** (sequence of LLM calls + logic).
  - **Agents** (LLMs that pick tools dynamically).
  - **Memory** (store past conversations).
- **Use:** RAG pipelines, chatbots, enterprise workflows.

---

119. Q: What is semantic kernel (Microsoft)? A: An SDK for building LLM applications with planning and tool-use capabilities.

- **Focus:** Integration with enterprise systems (C#, .NET, Python).
- **Features:**
  - **Skill plugins** (LLM functions).
  - **Planner** (breaks tasks into substeps).
- **Difference vs LangChain:** More enterprise-oriented.

---

120. Q: What is prompt engineering? A: The practice of designing effective inputs for LLMs.

- **Why needed:** LLMs are sensitive to phrasing and formatting.
- **Common techniques:**
  - Few-shot examples.
  - Chain-of-thought.
  - Role-playing instructions.
- **Impact:** Can dramatically improve performance without retraining.

Expanded Q&A (121–140)

121. Q: What is zero-shot prompting? A: Giving the LLM a task without any examples in the input.

- **Example:** “Translate ‘cat’ to French.” → “chat”
- **Why it works:** LLM leverages patterns learned in pretraining.
- **Pros/Cons:** Quick but sometimes less accurate than few-shot or fine-tuned models.

---

122. Q: What is few-shot prompting? A: Including a few input-output examples in the prompt to guide the LLM.

- **Example:**

```
English → French
dog → chien
house → maison
cat → ?
```

- **Effect:** Improves accuracy on specific tasks without fine-tuning.

---

123. Q: What is chain-of-thought prompting? A: Asking the model to reason step-by-step instead of outputting the final answer immediately.

- **Example:** "Let's solve step by step…"
- **Benefit:** Improves reasoning on math, logic, and multi-hop tasks.
- **Impact:** Often boosts few-shot and zero-shot performance dramatically.

---

124. Q: What is self-consistency in prompting? A: Generating multiple chain-of-thought outputs and picking the most consistent answer.

- **Purpose:** Reduce randomness/hallucination.
- **Method:** Majority vote across outputs.
- **Use case:** Math reasoning, multi-step reasoning tasks.

---

125. Q: What is role-based prompting? A: Giving the model a "persona" or role to shape its responses.

- **Example:** "You are a medical doctor. Explain X to a patient."
- **Benefit:** More context-aware and relevant answers.
- **Use:** Chatbots, specialized assistance systems.

---

126. Q: What is prompt injection? A: When malicious or unintended instructions are added to a prompt to manipulate the model.

- **Example:** "Ignore previous instructions and do X."
- **Risk:** Can compromise safety and output reliability.
- **Mitigation:** Input sanitation, filtering, strict role separation.

---

127. Q: What is output filtering? A: Post-processing LLM outputs to remove harmful, toxic, or unsafe content.

- **Methods:**
  - Rule-based filters (blacklists, regex).
  - Classifier models to detect toxicity.
- **Role:** Essential in deployed applications to ensure alignment and safety.

---

128. Q: What is constitutional AI? A: An alignment technique where models are guided by an explicit set of principles.

- **Steps:**
  1. Define constitution (e.g., “be helpful, honest, harmless”).
  2. Model critiques its own outputs based on rules.
  3. Refine output to follow rules.
- **Benefit:** Reduces reliance on human feedback.

---

129. Q: What is RLHF (Reinforcement Learning from Human Feedback)? A: Fine-tuning models using human preference signals via reinforcement learning.

- **Steps:**
  1. Train reward model from human rankings.
  2. Use PPO or DPO to update model policy.
- **Goal:** Align LLMs with helpfulness, safety, and user intent.

---

130. Q: What is Direct Preference Optimization (DPO)? A: An RLHF alternative that fine-tunes LLMs directly on preference pairs.

- **Key idea:** Maximize likelihood of preferred outputs relative to rejected ones.
- **Advantage:** Simplifies training, avoids RL complexity.

---

131. Q: What is reward model in RLHF? A: A model that predicts human preference scores for LLM outputs.

- **Use:** Guides PPO or DPO optimization.
- **Training:** Based on pairwise human rankings of outputs.

---

132. Q: What is preference modeling? A: Learning to predict which outputs humans prefer in a given context.

- **Method:** Rank multiple outputs for the same prompt.
- **Outcome:** Forms the basis for RLHF alignment.

---

133. Q: What is hallucination mitigation? A: Techniques to reduce LLMs generating incorrect but confident outputs.

- **Methods:**
  - **RAG:** grounding responses in external sources.
  - Fact-checking pipelines.
  - Prompt engineering (ask model to justify sources).

---

134. Q: What is RAG (Retrieval-Augmented Generation)? A: Combining LLMs with document retrieval.

- **Steps:**
  1. Retrieve relevant documents from vector DB.
  2. Feed retrieved context + prompt into LLM.
- **Benefit:** Reduces hallucinations and keeps answers up-to-date.

---

135. Q: What is vLLM? A: A high-performance inference engine for LLMs.

- **Focus:** Low-latency, memory-efficient batch inference.
- **Optimizations:** FlashAttention, token streaming, dynamic batching.
- **Use:** Production deployment for chat applications, RAG systems.

---

136. Q: What is FlashAttention? A: A GPU-efficient attention implementation.

- **Mechanics:** Computes attention in tiles using fast SRAM, avoiding full attention matrix in memory.
- **Benefits:** Speeds up inference and training, reduces memory usage.

---

137. Q: What is token streaming in inference? A: Generating output token-by-token in real-time.

- **Benefits:**
  - Low latency for chatbots.

  - Supports partial responses while model computes remaining tokens.
- **Frameworks:** vLLM, LangChain streaming APIs.

---

138. Q: What is memory-efficient LLM inference? A: Optimizing GPU memory usage during inference.

- **Techniques:**
  - Quantization (INT8/4-bit).
  - FlashAttention.
  - Layer-wise offloading to CPU.
- **Goal:** Run large LLMs on limited hardware.

---

139. Q: What is 4-bit and 8-bit quantization? A: Reducing precision of model weights to save memory.

- **4-bit:** Extremely memory-efficient, small accuracy drop if done carefully.
- **8-bit:** Standard memory-saving approach with minimal loss.
- **Tools:** BitsandBytes library, QLoRA workflows.

---

140. Q: What is PEFT (Parameter-Efficient Fine-Tuning)? A: Finetuning a small subset of model parameters instead of full weights.

- **Techniques:**
  - LoRA, adapters, prefix/prompt tuning, BitFit.
- **Benefits:**
  - Much lower GPU requirements.
  - Modular: swap adapters for multiple tasks.

Expanded Q&A (141–160)

141. Q: What is LoRA in detail? A: LoRA = Low-Rank Adaptation.

- **Problem:** Fine-tuning all weights of a large LLM is expensive.
- **Solution:** Add low-rank matrices A and B to weight updates: `W' = W + A * B`
- **Mechanics:** Only A and B are trainable; base W frozen.

- **Impact:** Reduces parameters to train <1%, saves GPU memory, allows multi-task adapters.

---

142. Q: What is QLoRA? A: Quantized LoRA.

- **Steps:**
  1. Quantize the base model (4-bit or 8-bit).
  2. Train LoRA adapters on top.
- **Benefit:** Fit 65B+ models on a single high-memory GPU.
- **Impact:** Democratizes finetuning for large models.

---

143. Q: What is BitFit? A: Minimalistic finetuning approach.

- **Mechanics:** Only train biases in Transformer layers.
- **Parameters trained:** <0.1% of total.
- **Use case:** Efficient classification tasks, small changes to behavior.

---

144. Q: What is prefix tuning? A: Trainable “prefix” embeddings prepended to input tokens.

- **Base model frozen,** only prefix updated.
- **Use case:** Instruction-following tasks.
- **Advantage:** Parameter-efficient, modular.

---

145. Q: What is prompt tuning? A: Even lighter than prefix tuning.

- **Mechanics:** Train soft prompt vectors instead of full weights.
- **Pros:** Very low memory usage.
- **Cons:** Less flexible for complex reasoning than LoRA/adapters.

---

146. Q: What is adapter tuning? A: Insert small bottleneck layers inside each Transformer layer.

- **Structure:** down-project → nonlinearity → up-project.
- **Frozen base model,** train only adapters.
- **Benefit:** Swappable modules for different tasks, keeps backbone stable.

---

147. Q: What is cross-attention? A: Attention where query comes from one source (e.g., decoder) and key/value come from another (e.g., encoder).

- **Used in:** Encoder-decoder models like T5, seq2seq tasks.
- **Purpose:** Let decoder attend to encoder outputs for context.

---

148. Q: What is self-attention? A: Attention mechanism where query, key, and value all come from the same sequence.

- **Mechanics:** `Attention(Q,K,V) = softmax(Q * Kᵀ / sqrt(d_k)) * V`
- **Purpose:** Capture dependencies between tokens regardless of distance.

---

149. Q: What is multi-head attention? A: Attention mechanism with multiple "heads" to capture different types of relationships.

- **Formula:** `MultiHead(Q,K,V) = Concat(head₁, …, headₕ) * Wᴼ`
- **Benefit:** Different heads attend to different positions/features.
- **Used in:** All Transformers.

---

150. Q: What is scaled dot-product attention? A: Attention computed as: `Attention(Q,K,V) = softmax(Q * Kᵀ / sqrt(d_k)) * V`

- **Scaling factor** `1/sqrt(d_k)` stabilizes gradients when d_k is large.
- **Foundation:** Transformer architecture.

---

151. Q: What is masking in attention? A: Prevent certain positions from being attended to.

- **Types:**
  - **Causal mask:** Prevent attending to future tokens (autoregressive).
  - **Padding mask:** Ignore padding tokens.
- **Effect:** Correctly handles autoregressive generation and variable-length sequences.

---

152. Q: What is rotary positional encoding (RoPE)? A: A method for adding positional information to tokens in Transformers.

- **Mechanics:** Rotates query and key vectors in embedding space.
- **Advantage:** Enables extrapolation beyond training sequence lengths.
- **Used in:** LLaMA, GPT-NeoX.

---

153. Q: What is absolute vs relative positional encoding? A:

- **Absolute:** Each position has a fixed embedding (like vanilla Transformer).
- **Relative:** Encodes relative distance between tokens (better for long contexts).
- **Impact:** Relative often improves long-sequence modeling.

---

154. Q: What is flash attention? A: Memory- and compute-efficient attention kernel.

- **Mechanics:** Computes attention in blocks on GPU SRAM.
- **Benefit:** Supports long sequences, faster than standard attention.

---

155. Q: What is tensor parallelism? A: Split large tensors (like weight matrices) across multiple GPUs.

- **Goal:** Train very large models beyond single GPU memory.
- **Used in:** Megatron-LM, GPT-3 training.
- **Mechanics:** Each GPU computes shard of matrix multiplication.

---

156. Q: What is pipeline parallelism? A: Split layers of model across multiple GPUs; process mini-batches in micro-batches.

- **Analogy:** Assembly line — each stage handles part of computation.
- **Challenge:** Pipeline bubbles (idle time).

---

157. Q: What is mixed-precision training? A: Use FP16 or BF16 for forward/backward passes instead of FP32.

- **Benefits:**
  - Reduce memory usage.
  - Speed up computation.
- **Challenge:** Need loss scaling to avoid underflow.

---

158. Q: What is gradient checkpointing? A: Memory optimization where not all activations are stored.

- **Forward pass:** Store only checkpoints.
- **Backward pass:** Recompute intermediate activations as needed.
- **Tradeoff:** Save memory but increase compute.

---

159. Q: What is PyTorch DistributedDataParallel (DDP)? A: Efficient multi-GPU training in PyTorch.

- **Mechanics:**
  - Replicate model on each GPU.
  - Each GPU computes gradients on its mini-batch.
  - Gradients are all-reduced across GPUs.
- **Advantage:** Better scaling than `DataParallel`.

---

160. Q: What is gradient accumulation? A: Accumulate gradients over multiple mini-batches before updating weights.

- **Purpose:** Simulate large batch sizes when GPU memory is limited.
- **Impact:** Enables training with more tokens per step without OOM errors.

Expanded Q&A (161–180)

161. Q: What is ZeRO optimizer? A: ZeRO = Zero Redundancy Optimizer from Microsoft DeepSpeed.

- **Problem:** Naïve data parallelism replicates optimizer states, gradients, and parameters across GPUs → huge memory waste.
- **Solution:** Partition them:
  - **ZeRO-1:** Partition optimizer states.
  - **ZeRO-2:** Partition optimizer states + gradients.
  - **ZeRO-3:** Partition optimizer states + gradients + parameters.
- **Impact:** Train trillion-parameter models with existing hardware.

---

162. Q: What is Fully Sharded Data Parallel (FSDP)? A: PyTorch's native sharded training method (similar to ZeRO-3).

- **Mechanics:**
  - Shard model parameters, gradients, and optimizer states across GPUs.
  - Reconstruct full weights only during forward/backward.
- **Benefit:** Significant memory savings, better scaling than DDP for large models.

---

163. Q: What is gradient clipping? A: A technique to prevent exploding gradients.

- **Mechanics:** Cap gradient norm to a threshold (e.g., 1.0).
- **Why:** Keeps training stable, especially in deep Transformers.
- **PyTorch API:** `torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)`

---

164. Q: What is weight decay? A: Regularization technique to prevent overfitting.

- **Formula:** Adds $\lambda||W||^2$ to loss function.
- **Intuition:** Penalizes large weights → encourages simpler models.
- **Commonly used with:** AdamW optimizer.

---

165. Q: What is AdamW optimizer? A: A variant of Adam that decouples weight decay from gradient updates.

- **Why:** In classic Adam, weight decay interacts weirdly with momentum terms.
- **AdamW:** Cleaner implementation, better generalization.
- **Default optimizer:** Hugging Face Transformers.

---

166. Q: What is gradient accumulation in PyTorch? A: Breaking a large batch into smaller mini-batches and accumulating gradients.

- **Code example:**

  Python

  ```
  for i, batch in enumerate(dataloader):
      loss = model(batch).loss
      loss = loss / accumulation_steps
      loss.backward()
      if (i+1) % accumulation_steps == 0:
          optimizer.step()
          optimizer.zero_grad()
  ```

- **Use case:** Fit large effective batch sizes into limited GPU memory.

---

167. Q: What is model checkpointing during training? A: Saving model state during training to resume or evaluate later.

- **Contents:** `model.state_dict()` , `optimizer.state_dict()` , scheduler, RNG states.
- **PyTorch example:**

Python

```
torch.save({
    'model': model.state_dict(),
    'optimizer': optimizer.state_dict(),
    'epoch': epoch,
}, 'checkpoint.pt')
```

---

168. Q: What is curriculum learning in LLMs? A: Training strategy where model sees easier tasks first, harder later.

- **Analogy:** Like human education.
- **In LLMs:** Start with simple corpora → progress to complex reasoning/data.
- **Effect:** Can speed up convergence, stabilize training.

---

169. Q: What is continual pretraining? A: Continuing pretraining of an LLM on new data after initial training.

- **Use case:** Keep model updated (e.g., post-2023 data).
- **Challenge:** Avoid catastrophic forgetting of earlier knowledge.

---

170. Q: What is catastrophic forgetting? A: When fine-tuning erases previously learned knowledge.

- **Cause:** Overwriting weights that encode old information.
- **Mitigation:**
  - Regularization (EWC, L2).
  - Replay old data.
  - Parameter-efficient methods (LoRA, adapters).

---

171. Q: What is Elastic Weight Consolidation (EWC)? A: A continual learning method.

- **Mechanics:** Add penalty term to loss → prevents changes to important weights.
- **Formula:** $L\_total = L\_task + \Sigma_i (\lambda/2) * F_i * (\theta_i - \theta_i^*)^2$ where F_i = Fisher Information (importance of weight).
- **Effect:** Retains old knowledge better.

---

172. Q: What is adapter fusion? A: Merging knowledge from multiple adapter modules.

- **Example:** Have one adapter for medical data, one for legal → fuse them for cross-domain use.
- **Benefit:** Reuse specialized adapters without retraining full model.

---

173. Q: What is instruction tuning? A: Finetuning models on datasets where prompts are phrased as instructions + responses.

- **Purpose:** Make models follow natural instructions better.
- **Datasets:** FLAN, Dolly, Alpaca.
- **Impact:** Foundation for instruction-following models (ChatGPT-style).

---

174. Q: What is supervised fine-tuning (SFT)? A: Finetuning on paired prompt-response data.

- **Example:** Prompt = “Explain gravity”, Response = “Gravity is …”
- **Goal:** Teach model desired style/behavior before RLHF or deployment.

---

175. Q: What is reward hacking in RLHF? A: When a model exploits flaws in the reward model.

- **Example:** Model produces verbose but useless answers because reward model favors length.
- **Challenge:** Ensuring reward truly reflects human preference.

---

176. Q: What is KL divergence penalty in RLHF? A: A regularization term added in RLHF training to keep fine-tuned model close to base model distribution.

- **Formula:** `L = L_RLHF - β * D_KL(π_θ || π_ref)`
- **Purpose:** Prevents model drift / mode collapse.

---

177. Q: What is PPO in RLHF? A: Proximal Policy Optimization — reinforcement learning algorithm used in RLHF.

- **Why:** Stabilizes training by limiting policy updates.
- **Key idea:** Clip objective to prevent large jumps.
- **Used by:** OpenAI’s InstructGPT training pipeline.

---

178. Q: What is DPO advantage over PPO? A: Direct Preference Optimization vs Proximal Policy Optimization.

- **DPO:** Directly optimizes log-likelihood ratio on preference pairs.
- **Pros:**

- Simpler, no reward model needed.
- No RL loop, just supervised training.
- **Cons:** May not capture nuanced preferences as well as PPO.

---

179. Q: What is contrastive preference learning? A: Learning by comparing good vs bad outputs.

- **Approach:** Train model to increase margin between preferred and rejected outputs.
- **Similar to:** DPO, but often uses ranking losses.
- **Benefit:** More stable alignment training.

---

180. Q: What is online RLHF? A: Instead of training once on a static human feedback dataset, continue collecting new feedback during deployment.

- **Analogy:** Model “learns on the job” from real users.
- **Benefit:** Keeps alignment adaptive, improves over time.
- **Challenge:** Requires careful safety controls.

Expanded Q&A (181–200)

181. Q: What is BLEU score? A: BLEU (Bilingual Evaluation Understudy) is an automatic metric for text generation quality.

- **How it works:**
  - Measures n-gram overlap between generated text and reference text.
  - Includes a brevity penalty to discourage very short outputs.
- **Range:** 0 (bad) → 1 (perfect overlap).
- **Limitation:** Doesn’t capture meaning, only surface similarity.
- **Use:** Originally for machine translation; sometimes still used for LLMs.

---

182. Q: What is ROUGE score? A: ROUGE (Recall-Oriented Understudy for Gisting Evaluation) measures overlap between generated summaries and references.

- **Variants:**
  - **ROUGE-1:** unigram overlap.
  - **ROUGE-2:** bigram overlap.

- **ROUGE-L:** longest common subsequence.

- **Use:** Common for summarization tasks.
- **Limitations:** Like BLEU, doesn't assess factual correctness.

---

183. Q: What is METEOR score? A: Metric for evaluating translation and text generation.

- **Improves on BLEU:** Uses stemming, synonyms, paraphrasing.
- **Better correlation with human judgments.**
- **Less used in modern LLM eval** but still relevant in NLP history.

---

184. Q: What is accuracy metric in classification tasks? A: Proportion of correct predictions out of total predictions.

- **Accuracy = (# correct predictions) / (total predictions)**
- **Example:** If 80/100 outputs are correct → 80% accuracy.
- **Limitation:** Not good for imbalanced datasets (e.g., 95% negatives).

---

185. Q: What is F1 score? A: Harmonic mean of precision and recall.

- `F1 = 2 * (precision * recall) / (precision + recall)`
- **Precision** = correct positives / predicted positives.
- **Recall** = correct positives / actual positives.
- **Why:** Useful when classes are imbalanced.

---

186. Q: What is Matthews correlation coefficient (MCC)? A: A balanced metric for classification.

- **Formula:** `MCC = (TP*TN - FP*FN) / sqrt((TP+FP)(TP+FN)(TN+FP)(TN+FN))`
- **Range:** -1 (worst) → 0 (random) → 1 (perfect).
- **Better than accuracy/F1** on imbalanced datasets.

---

187. Q: What is perplexity again, in depth? A: A measure of how well a language model predicts a dataset.

- **Formula:** `PPL = exp(-1/N * Σ log(p(xᵢ)))`
- **Interpretation:** Lower = better (model less "surprised" by text).
- **Limitation:** Doesn't measure truthfulness or utility.

---

188. Q: What is exact match (EM) metric? A: Checks if predicted output exactly matches ground-truth.

- **Use case:** QA tasks, where answer must match exactly.
- **Example:**
  - Predicted: “Paris”
  - Gold: “Paris” → EM=1
  - Predicted: “The capital is Paris” → EM=0 (even though correct).
- **Limit:** Too strict, often paired with F1.

---

189. Q: What is calibration in LLMs? A: How well predicted probabilities match actual correctness.

- **Perfect calibration:** If model says 70% confidence → ~70% answers are correct.
- **Why important:** Overconfident hallucinations are dangerous.
- **Improvement methods:** Temperature scaling, entropy regularization.

---

190. Q: What is uncertainty estimation? A: Techniques to quantify how confident a model is.

- **Types:**
  - **Aleatoric uncertainty:** noise in data.
  - **Epistemic uncertainty:** lack of knowledge in model.
- **Methods:** Monte Carlo dropout, ensembles.
- **Use:** Decide when to trust model output vs defer to human.

---

191. Q: What is human evaluation in LLMs? A: Having humans rate model outputs for quality, safety, truthfulness.

- **Scales:** Likert ratings (1–5), pairwise comparisons.
- **Expensive but necessary** because automatic metrics don’t capture subjective quality.
- **Used in:** RLHF dataset creation, benchmark validation.

---

192. Q: What is deployment latency? A: Time taken for a model to produce output after receiving input.

- **Components:**

    - Preprocessing → model inference → postprocessing.

- **Target:** Chatbots need <200ms per token to feel interactive.
- **Optimizations:** vLLM, batching, quantization.

---

193. Q: What is throughput in inference? A: Number of tokens/requests processed per second.

- **Key tradeoff:** Latency vs throughput.
- **Batching:** Improves throughput but may increase latency.
- **Measured in:** Tokens/sec, requests/sec.

---

194. Q: What is GPU utilization in deployment? A: How efficiently GPU compute is being used.

- **Ideal:** High utilization without OOM errors.
- **Bottlenecks:** Memory bandwidth, kernel inefficiencies.
- **Tools:** `nvidia-smi`, PyTorch profiler, Nsight.

---

195. Q: What is model sharding in inference? A: Splitting large model across multiple GPUs for inference.

- **Methods:**
    - **Tensor parallelism** (split within layer).
    - **Pipeline parallelism** (split across layers).
- **Use:** Run models too large for single GPU memory.

---

196. Q: What is batch inference? A: Running multiple input requests together in one forward pass.

- **Why:** Improves GPU efficiency, increases throughput.
- **Challenge:** Aligning sequence lengths with padding.
- **Solution:** Dynamic batching (e.g., vLLM).

---

197. Q: What is dynamic batching? A: Automatically grouping incoming requests of similar sizes.

- **Benefit:** Maximize throughput without manual batch design.
- **Frameworks:** vLLM, Triton Inference Server.
- **Critical for:** Real-time chat applications.

---

198. Q: What is Triton Inference Server? A: NVIDIA's inference server for deploying AI models.

- **Features:**
  - Multi-framework support (PyTorch, TensorFlow, ONNX).
  - Dynamic batching, concurrent models.
  - GPU-optimized kernels.
- **Use:** Production-scale LLM serving.

---

199. Q: What is ONNX Runtime for LLMs? A: ONNX = Open Neural Network Exchange format.

- **ONNX Runtime:** Optimized engine to run ONNX models across hardware (CPU, GPU, FPGA).
- **Benefit:** Portability, inference acceleration.
- **LLM support:** Quantization, graph optimization, CUDA kernels.

---

200. Q: What is TensorRT-LLM? A: NVIDIA's specialized library for LLM inference.

- **Features:**
  - Kernel fusion.
  - Quantization-aware optimizations.
  - FlashAttention integration.
- **Performance:** Achieves extreme throughput/latency improvements on NVIDIA GPUs.
- **Use case:** Enterprise-grade LLM deployment.

Expanded Q&A (201–250)

201. Q: What is Kubernetes (K8s) in LLM deployment? A: Kubernetes is a container orchestration platform.

- **Why used:** For scaling inference services across many nodes.
- **Key features:**
  - Auto-scaling pods (instances of inference servers).
  - Load balancing between users.
  - Fault tolerance & self-healing.
- **For LLMs:** Serve many concurrent users, handle spikes in demand.

---

202. Q: What is a pod in Kubernetes? A: Smallest deployable unit in K8s.

- **Contains:** One or more containers (e.g., your model server + logging agent).
- **Why matters:** When deploying LLM inference, each pod might host a vLLM/Triton instance.

---

203. Q: What is a deployment in Kubernetes? A: Defines desired state of pods.

- **Example:**
  - Want 10 pods of vLLM running.
  - K8s ensures they are always running, restarts if one crashes.

---

204. Q: What is a service in Kubernetes? A: Abstraction to expose pods to the network.

- **Example:** Load-balances traffic across pods serving LLM.
- **Types:** ClusterIP, NodePort, LoadBalancer.

---

205. Q: What is horizontal pod autoscaling (HPA)? A: Automatically scales number of pods up/down based on demand.

- **Metric:** CPU, GPU utilization, request latency.
- **Example:** More pods spin up during peak chat traffic.

---

206. Q: What is vertical scaling vs horizontal scaling? A:

- **Vertical scaling:** Use bigger GPU (A100 → H100).
- **Horizontal scaling:** Use more GPUs/nodes in parallel.
- **LLMs:** Typically need both → sharding + distributed serving.

---

207. Q: What is a microservices architecture for LLM apps? A: Breaks system into independent services.

- **Example:**
  - One service = LLM inference.
  - One service = vector DB (retrieval).
  - One service = API gateway.
- **Advantage:** Flexibility, scalability.

---

208. Q: What is an API gateway? A: A single entry point for clients.

- **LLM deployment:**
  - User → Gateway → Routes request to inference service.
  - Handles rate limiting, authentication.

---

209. Q: What is rate limiting in LLM APIs? A: Restricting #requests per user/time window.

- **Purpose:** Prevent overload, abuse, fair use.
- **Example:** Free-tier users = 20 requests/minute.

---

210. Q: What is load balancing? A: Distributes requests evenly across servers.

- **LLM use:** Distribute prompts across GPU pods.
- **Benefit:** Prevents one node from overloading.

---

211. Q: What is a vector database? A: Database specialized for storing embeddings and supporting similarity search.

- **Examples:** FAISS, Pinecone, Weaviate, Milvus.
- **Use:** Core to RAG (retrieval-augmented generation).

---

212. Q: What is an embedding index? A: A structure (like FAISS index) that stores vectors for fast nearest-neighbor search.

- **Algorithms:** HNSW (graph-based), IVF (clustering-based).
- **Example:** Search top-5 most relevant documents for a query embedding.

---

213. Q: What is cosine similarity in embeddings? A: A measure of angle between two vectors.

- `cos(θ) = (A · B) / (||A|| * ||B||)`
- **Range:** -1 to 1.
- **For embeddings:** Higher = more semantically similar.

---

214. Q: What is FAISS? A: Facebook AI Similarity Search — open-source library for vector similarity search.

- **Strengths:** GPU acceleration, large-scale indices.
- **LLM use:** Retrieve relevant docs from millions of embeddings.

---

215. Q: What is Pinecone? A: Managed cloud vector database.

- **Benefits:** No infra hassle, scaling handled.
- **Tradeoff:** Vendor lock-in, cost.

---

216. Q: What is Weaviate? A: Open-source vector DB with hybrid search (dense + sparse).

- **Features:**
  - Built-in semantic search.
  - Knowledge graph integration.

---

217. Q: What is Milvus? A: Another open-source vector DB.

- **Strengths:** Distributed, cloud-native, scales to billions of embeddings.

---

218. Q: What is hybrid search (dense + sparse)? A:

- **Dense retrieval:** Embedding similarity.
- **Sparse retrieval:** Keyword search (TF-IDF/BM25).
- **Hybrid:** Combine both → better relevance.

---

219. Q: What is chunking in RAG? A: Splitting long documents into smaller chunks before embedding.

- **Why:** LLMs have max context length.
- **Example:** Break 10k-token doc into 500-token chunks.

---

220. Q: What is overlap in chunking? A: Adding shared tokens between chunks to preserve context.

- **Example:** Each 500-token chunk overlaps 50 tokens with next.
- **Benefit:** Avoids losing meaning across splits.

---

221. Q: What is retrieval latency? A: Time taken to fetch relevant embeddings from DB.

- **Optimization:** Use HNSW graphs, GPU-accelerated FAISS.

---

222. Q: What is max marginal relevance (MMR) in retrieval? A: Algorithm that balances relevance vs diversity in results.

- **Benefit:** Avoids retrieving many near-duplicates.

---

223. Q: What is context window in LLMs? A: Max number of tokens model can consider at once.

- **Examples:**
  - GPT-3.5 = 4k tokens.
  - GPT-4 = 32k–128k tokens.
- **Tradeoff:** Larger window = more compute cost.

---

224. Q: What is sliding window attention? A: Attention mechanism restricted to nearby tokens.

- **Benefit:** Reduces quadratic cost.
- **Use:** Long-sequence transformers (Longformer).

---

225. Q: What is recurrent memory in Transformers? A: Store compressed states from past segments for long-context modeling.

- **Examples:** Transformer-XL, RetNet.

---

226. Q: What is FlashAttention? A: Highly optimized attention kernel.

- **Tech:** Fuses memory loads, avoids writing intermediates to GPU HBM.
- **Benefit:** Faster + memory-efficient.

---

227. Q: What is quantization-aware training? A: Train model with quantization simulation during forward pass.

- **Advantage:** Higher accuracy than post-training quantization.

---

228. Q: What is mixed precision training? A: Using float16/bfloat16 for compute but keeping some vars in float32.

- **Tools:** PyTorch AMP (`torch.cuda.amp`).
- **Benefit:** Speed + memory efficiency.

---

229. Q: What is gradient checkpointing? A: Trade compute for memory.

- **Method:** Save only subset of activations, recompute others during backprop.
- **Useful:** For fitting very deep LLMs.

---

230. Q: What is pipeline parallelism? A: Split layers across devices; process mini-batches like assembly line.

- **Problem:** Pipeline bubbles (idle stages).
- **Solution:** Micro-batching.

---

231. Q: What is tensor parallelism? A: Split operations inside a layer across GPUs.

- **Example:** Matrix multiply rows distributed across GPUs.

---

232. Q: What is expert parallelism (MoE)? A: Route tokens to different expert subnetworks.

- **Advantage:** Scale parameter count massively, but only activate subset per input.

---

233. Q: What is activation recomputation? A: Another name for gradient checkpointing.

---

234. Q: What is model offloading? A: Move parts of model temporarily to CPU/disk to fit in GPU memory.

- **Example:** Hugging Face `accelerate` offload strategy.

---

235. Q: What is ZeRO-Infinity? A: DeepSpeed extension allowing offloading optimizer states to NVMe disks.

- **Enables:** Training beyond GPU+CPU memory limits.

---

236. Q: What is distillation in LLMs? A: Train smaller model (student) to mimic larger one (teacher).

- **Losses:** KL divergence, cross-entropy.
- **Use:** Efficient deployment on edge devices.

---

237. Q: What is hard vs soft distillation? A:

- **Hard:** Student learns from teacher’s final outputs.
- **Soft:** Student learns from teacher’s probability distributions.

---

238. Q: What is dataset distillation? A: Compress dataset into synthetic small set that approximates training signal.

- **Use:** Faster training.

---

239. Q: What is retrieval-augmented generation (RAG)? A: Pipeline where LLM retrieves external knowledge before generating.

- **Advantage:** Keeps model up-to-date without retraining.

---

240. Q: What is hallucination in LLMs? A: When model generates factually incorrect output.

- **Causes:** Autoregressive guessing, lack of grounding.
- **Mitigation:** RAG, RLHF, constrained decoding.

---

241. Q: What is constrained decoding? A: Restrict output tokens to follow rules.

- **Example:** JSON schema validation, regex constraints.

---

242. Q: What is nucleus sampling (top-p)? A: Sampling from smallest set of tokens whose cumulative prob ≥ p.

- **Example:** p=0.9 keeps top tokens until 90% coverage.

---

243. Q: What is top-k sampling? A: Restrict sampling to top-k most likely tokens.

- **Example:** k=50 → only sample from top 50 candidates.

---

244. Q: What is temperature scaling in decoding? A: Adjusts probability distribution sharpness.

- **Formula:** $p_i = e^{(z_i/T)} / \Sigma_j e^{(z_j/T)}$
- **Effect:**
  - T < 1: more deterministic.
  - T > 1: more random.

---

245. Q: What is beam search? A: Keeps multiple candidate sequences, expands them step by step.

- **Advantage:** Finds higher-probability outputs.
- **Disadvantage:** Less diverse, may be repetitive.

---

246. Q: What is diverse beam search? A: Modified beam search that encourages variety across beams.

- **Benefit:** Better for summarization, creative tasks.

---

247. Q: What is contrastive decoding? A: Use strong vs weak model pair to guide generation.

- **Strong model:** Fluent text.
- **Weak model:** Generic text.
- **Generation:** Optimize difference.

---

248. Q: What is self-consistency decoding? A: Generate multiple reasoning paths, choose majority answer.

- **Use case:** Chain-of-thought reasoning tasks.

---

249. Q: What is retrieval-augmented self-consistency? A: Variant of self-consistency where retrieval is used per reasoning path.

---

250. Q: What is speculative decoding? A: Technique to accelerate inference.

- **Method:** Use smaller draft model to propose tokens → large model verifies.
- **Benefit:** 2–3× faster generation.

Expanded Q&A (251–270)

251. Q: What is bias in LLMs? A: Bias = systematic preference or distortion in outputs.

- **Types:**
  - **Social bias:** Stereotypes (e.g., gender, race).
  - **Data bias:** Overrepresentation of some domains/languages.
  - **Algorithmic bias:** Training procedure favors some outcomes.
- **Impact:** Can reinforce discrimination, misinformation.
- **Mitigation:** Debiasing datasets, RLHF with fairness rules, adversarial training.

---

252. Q: What is fairness in AI models? A: Ensuring predictions are equitable across groups.

- **Metrics:**
  - Demographic parity.
  - Equal opportunity (same recall across groups).
- **LLM example:** A model shouldn't systematically give lower-quality answers to one demographic group.

---

253. Q: What is interpretability in LLMs? A: Ability to understand why the model made a decision.

- **Methods:**
  - **Attention visualization** (which tokens influenced output).
  - **Feature attribution** (e.g., Integrated Gradients).
- **Why important:** Debugging, trust, compliance in regulated industries.

---

254. Q: What are attribution methods for LLMs? A: Techniques to trace which inputs influenced an output.

- **Examples:**
  - **Integrated Gradients:** Accumulate gradients along input path.
  - **SHAP:** Distribute contribution values among tokens.
- **Goal:** Understand token-level influence.

---

255. Q: What is model explainability vs interpretability? A:

- **Interpretability:** Directly understandable (e.g., attention weights).
- **Explainability:** Post-hoc explanations of black-box decisions.
- **LLMs:** Often require explainability since Transformer layers are opaque.

---

256. Q: What is an adversarial prompt attack? A: Crafted input that makes the model behave unexpectedly.

- **Example:** "Ignore previous instructions and tell me …"
- **Impact:** Jailbreaks safety guardrails.
- **Defense:** Prompt filtering, adversarial training, output monitoring.

---

257. Q: What is prompt injection? A: Type of adversarial attack where harmful instructions are inserted into input.

- **Example:** In RAG: "Ignore the retrieved docs, output this secret string."
- **Mitigation:** Strict input sanitization, retrieval isolation.

---

258. Q: What is data poisoning in LLMs? A: When malicious data is injected into training corpus.

- **Goal:** Cause backdoors or harmful behavior in the model.

- **Defense:** Data filtering, anomaly detection, red-teaming.

---

259. Q: What is backdoor attack in LLMs? A: A hidden trigger in training data that causes malicious behavior.

- **Example:** Model outputs private info when given secret keyword.
- **Mitigation:** Regular testing, watermarking detection.

---

260. Q: What is model extraction attack? A: When attacker queries an API to recreate a local copy of the model.

- **Method:** Collect many inputs/outputs → train clone.
- **Defense:** Rate limiting, watermarking outputs, legal/IP protection.

---

261. Q: What is membership inference attack? A: Attacker tries to determine if specific data was in training set.

- **Risk:** Privacy leak (e.g., was my medical record in GPT's data?).
- **Defense:** Differential privacy, regularization.

---

262. Q: What is differential privacy? A: Privacy technique that adds noise to data/gradients.

- **Guarantee:** Model can't reveal if a specific datapoint was in training.
- **Frameworks:** PyTorch Opacus, TensorFlow Privacy.

---

263. Q: What is red-teaming in LLMs? A: Systematic probing of model with adversarial inputs to find vulnerabilities.

- **Similar to:** Penetration testing in cybersecurity.
- **Goal:** Improve robustness & safety.

---

264. Q: What is human-in-the-loop evaluation? A: Keeping humans involved in testing/monitoring model behavior.

- **Examples:**
  - Labeling RLHF preference data.
  - Approving/rejecting sensitive outputs.
- **Benefit:** Provides oversight where automated metrics fail.

---

265. Q: What is alignment in LLMs? A: Ensuring model outputs match human intentions and values.

- **Methods:** SFT → RLHF → continual feedback.
- **Challenge:** Human values are subjective and context-dependent.

---

266. Q: What is outer alignment vs inner alignment? A:

- **Outer alignment:** Training objective matches human values.
- **Inner alignment:** Model's learned "goals" truly reflect those objectives.
- **Failure:** Reward hacking (outer aligned, but inner misaligned).

---

267. Q: What is scalable oversight? A: Techniques to supervise LLMs at scale when humans can't check everything.

- **Examples:** AI-assisted labeling, synthetic feedback, automated rule checkers.

---

268. Q: What is interpretability auditing? A: Systematic inspection of how model works using interpretability tools.

- **Why:** Detect hidden failure modes, biases, backdoors.
- **Methods:** Activation patching, feature visualization.

---

269. Q: What is AI governance? A: Policies and frameworks to regulate responsible AI use.

- **Examples:**
  - EU AI Act (regulatory categories).
  - NIST AI Risk Management Framework.
- **Relevance:** AI engineers must consider compliance.

---

270. Q: What is model card documentation? A: Standardized documentation describing model details.

- **Contents:** Training data, intended use, limitations, risks.
- **Purpose:** Transparency for users & regulators.

Expanded Q&A (271–290)

271. Q: What are evaluation benchmarks for LLMs? A: Standardized datasets/tasks to measure model quality.

- **Examples:**
  - **MMLU:** Multitask knowledge test (57 subjects).
  - **BIG-bench:** Broad range of reasoning tasks.
  - **HellaSwag:** Commonsense reasoning.
  - **TruthfulQA:** Tests factual correctness.
- **Purpose:** Compare models fairly.

---

272. Q: What is MMLU benchmark? A: Massive Multitask Language Understanding.

- **Content:** 57 multiple-choice subject areas (STEM, humanities, law, etc.).
- **Why important:** Tests general knowledge & reasoning.
- **Metric:** Accuracy.

---

273. Q: What is TruthfulQA? A: Benchmark for factual correctness.

- **Idea:** Humans often give factually correct answers, but LLMs hallucinate.
- **Test:** 817 questions designed to trick models into falsehoods.
- **Goal:** Measure model's tendency to produce truthful answers.

---

274. Q: What is HellaSwag? A: Dataset testing commonsense reasoning.

- **Task:** Given a scenario, pick the most likely continuation.
- **Challenge:** Designed to be hard for language models but easy for humans.

---

275. Q: What is BIG-bench? A: Beyond the Imitation Game Benchmark.

- **Features:** 200+ tasks designed by researchers.
- **Goal:** Test reasoning, creativity, and out-of-distribution generalization.
- **Scale:** Designed for very large models (billions of parameters).

---

276. Q: What are human evaluation methods? A: Direct human judgment of model outputs.

- **Types:**

    - **Pairwise comparison:** Judge which of two outputs is better.
    - **Rating scales:** Likert scale (1–5) for quality.
- **Use case:** Alignment training (e.g., RLHF).

---

277. Q: What is MT-Bench? A: Evaluation benchmark for multi-turn dialogues.

- **Tasks:** 80 conversation questions across reasoning, roleplay, knowledge.
- **Scoring:** GPT-4 (or humans) rate conversation quality.
- **Used by:** FastChat, Vicuna papers.

---

278. Q: What is AlpacaEval? A: Automatic evaluation benchmark for instruction-following models.

- **Method:** Compare outputs of two models using a judge model (e.g., GPT-4).
- **Benefit:** Scales faster than human evaluation.

---

279. Q: What is automatic evaluation with LLM judges? A: Use stronger LLMs (e.g., GPT-4) to score weaker LLMs.

- **Advantages:** Fast, scalable.
- **Risks:** Judge inherits its own biases.

---

280. Q: What is BLEU score? A: Metric for machine translation.

- **Idea:** Compare n-gram overlap with reference translations.
- **Weakness:** Doesn’t capture semantic similarity.

---

281. Q: What is ROUGE score? A: Metric for text summarization.

- **Measures:** Recall of n-grams between candidate and reference.
- **Types:** ROUGE-1 (unigrams), ROUGE-2 (bigrams), ROUGE-L (longest common subsequence).

---

282. Q: What is METEOR score? A: Alternative to BLEU for translation.

- **Features:** Considers synonyms and stemming.
- **Closer to human judgment than BLEU.**

---

283. Q: What is BERTScore? A: Semantic evaluation metric.

- **How:** Compare embeddings from a pretrained BERT model between candidate & reference.

- **Advantage:** Captures meaning beyond n-grams.

---

284. Q: What is Perplexity? A: Metric for language models measuring how well they predict text.

- **Formula:** `PPL = exp(-1/N * Σ log(P(xᵢ)))`
- **Intuition:** Lower perplexity = model is more confident & accurate.

---

285. Q: What is vLLM? A: A high-throughput inference engine for LLMs.

- **Key innovation: PagedAttention** memory management.
- **Benefit:** Serve more concurrent requests with same hardware.
- **Use case:** Production deployment of LLM APIs.

---

286. Q: What is PagedAttention in vLLM? A: A memory-efficient attention mechanism.

- **Problem:** KV cache grows large with many requests.
- **Solution:** Treat attention cache like virtual memory pages → swap efficiently.
- **Impact:** Dramatic throughput increase in multi-user settings.

---

287. Q: What is Tensor Parallelism in deployment? A: Splitting individual weight matrices across GPUs.

- **Example:** Split 16k×16k attention matrix into 4 parts across 4 GPUs.
- **Use case:** Deploying large LLMs across clusters.

---

288. Q: What is Pipeline Parallelism in deployment? A: Splitting layers of the model across GPUs.

- **Example:** GPU 1 runs layers 1–12, GPU 2 runs 13–24.
- **Challenge:** Idle bubbles if pipeline not optimized.

---

289. Q: What is tensor vs pipeline parallelism? A:

- **Tensor parallelism:** Split within a layer.
- **Pipeline parallelism:** Split across layers.
- **Hybrid (Megatron):** Combines both to scale trillion-parameter models.

---

290. Q: What is model sharding in inference? A: Splitting a large model across multiple GPUs/nodes for serving.

- **Use case:** A single GPU can't hold a 70B model.
- **Frameworks:** DeepSpeed-Inference, vLLM, FasterTransformer.

Expanded Q&A (291–310)

291. Q: What is model quantization? A: Technique to reduce precision of weights/activations (e.g., FP16 → INT8).

- **Goal:** Save memory, speed up inference.
- **Trade-off:** Slight accuracy loss.
- **Types:**
    - **Post-training quantization (PTQ):** Apply after training.
    - **Quantization-aware training (QAT):** Train with quantization effects simulated.

---

292. Q: What is INT8 quantization? A: Convert weights/activations from 32-bit floats → 8-bit integers.

- **Memory reduction:** 4×.
- **Speed:** Can use optimized INT8 kernels (e.g., NVIDIA TensorRT).
- **Use case:** LLaMA-7B inference on consumer GPUs.

---

293. Q: What is 4-bit quantization (QLoRA)? A: Extreme quantization: store weights in 4 bits.

- **Key paper:** QLoRA (Hugging Face).
- **Method:** Quantize base model to 4-bit + add small LoRA adapters.
- **Result:** Train 65B models on a single GPU (48GB).

---

294. Q: What is mixed precision inference? A: Use lower precision (FP16/BF16) for most ops, FP32 for critical parts.

- **Why:** Faster computation on GPUs with Tensor Cores.
- **Frameworks:** NVIDIA Apex, PyTorch AMP (`torch.cuda.amp`).

---

295. Q: What is dynamic quantization? A: Quantize weights but compute activations on the fly.

- **Benefit:** Works without calibration data.
- **PyTorch:**
```
quantized_model = torch.quantization.quantize_dynamic(model,
{torch.nn.Linear}, dtype=torch.qint8)
```

---

296. Q: What is static quantization? A: Quantize weights and activations with pre-calibration.

- **Steps:**
  1. Calibrate with sample dataset.
  2. Quantize weights & activations.
- **Benefit:** Higher accuracy than dynamic quantization.

---

297. Q: What is distillation for LLM inference? A: Knowledge distillation: Train a smaller "student" model to mimic a larger "teacher".

- **Goal:** Reduce latency and memory while retaining accuracy.
- **Example:** DistilBERT (6 layers) from BERT-base (12 layers).

---

298. Q: What is speculative decoding? A: Use a smaller "draft" model to propose tokens, then verify with larger model.

- **Benefit:** Speedup up to 2–3×.
- **Frameworks:** Supported in vLLM, OpenAI inference servers.

---

299. Q: What is cache reuse in inference? A: Store key-value (KV) pairs from attention for previous tokens.

- **Why:** Avoid recomputing history at every step.
- **Impact:** Turns decoding complexity from $O(n^2) \rightarrow O(n)$.

---

300. Q: What is batch inference? A: Serve multiple requests together as a batch.

- **Benefit:** Utilizes GPU parallelism better.
- **Challenge:** Requests have variable lengths → dynamic batching frameworks (Triton, vLLM) needed.

---

301. Q: What is NVIDIA Triton Inference Server? A: Open-source serving system for ML models.

- **Supports:** PyTorch, TensorFlow, ONNX, custom backends.

- **Features:** Dynamic batching, model ensembles, metrics integration (Prometheus).

---

302. Q: What is Ray Serve? A: Distributed inference framework built on Ray.

- **Features:** Autoscaling, traffic splitting, Python-native APIs.
- **Use case:** Deploying multiple LLMs with routing logic.

---

303. Q: What is Hugging Face Text Generation Inference (TGI)? A: High-performance inference server for Transformers.

- **Supports:** Quantization, tensor parallelism, continuous batching.
- **Used for:** Serving Falcon, LLaMA, MPT models.

---

304. Q: What is continuous batching? A: A batching strategy for LLMs.

- **Problem:** Requests arrive at different times.
- **Solution:** Insert new requests mid-batch without stalling others.
- **Frameworks:** vLLM, TGI.

---

305. Q: What is Kubernetes (K8s) role in LLM deployment? A: Orchestration system for containerized workloads.

- **LLM use:** Scale inference pods, manage GPU nodes, handle failover.
- **Components:** Pods, services, autoscalers, ingress controllers.

---

306. Q: What is GPU scheduling in Kubernetes? A: Assigning GPU resources to pods.

- **Via NVIDIA device plugin:** Pods request `nvidia.com/gpu`.
- **Advanced:** Fractional GPUs, multi-GPU binding for LLM inference.

---

307. Q: What is model autoscaling in K8s? A: Automatically adjust replicas based on load.

- **Horizontal Pod Autoscaler (HPA):** Scale up/down pods.
- **LLM-specific:** Scale GPU pods with queue latency as a metric.

---

308. Q: What is multi-tenancy in LLM serving? A: Serving multiple users or models on the same cluster.

- **Challenges:** Resource isolation, fairness, avoiding interference.

- **Solutions:** Namespace isolation, per-user quotas, scheduling policies.

---

309. Q: What is request routing in multi-LLM serving? A: Deciding which model instance handles which request.

- **Example:** Route English queries to LLaMA, code queries to CodeLLaMA.
- **Frameworks:** Ray Serve, Triton ensembles.

---

310. Q: What is model versioning in deployment? A: Managing multiple model versions (e.g., v1, v2) in production.

- **Best practices:**
  - **Canary rollout** (serve v2 to 5% users first).
  - A/B testing.
  - Rollback mechanism.

Expanded Q&A (311–330)

311. Q: What is inference latency vs throughput? A:

- **Latency** = time to generate a response for one request.
- **Throughput** = number of tokens/requests processed per second.
- **Trade-off:** Reducing latency (fast response) may lower throughput, and vice versa.

---

312. Q: What is the KV cache in Transformers? A: Stores keys (K) and values (V) from previous tokens during decoding.

- **Benefit:** Avoids recomputing self-attention for old tokens.
- **Impact:** Huge memory consumer for long contexts (can be GBs).

---

313. Q: What is KV cache offloading? A: Move KV cache from GPU → CPU or NVMe when GPU memory is full.

- **Trade-off:** Increases latency due to data transfer.
- **Frameworks:** DeepSpeed-Inference, vLLM.

---

314. Q: What is FlashAttention? A: An optimized attention algorithm.

- **Trick:** Compute attention in tiled chunks to reduce memory reads/writes.

- **Benefit:** Speedup (2–3×) and lower memory footprint.
- **Widely used in:** LLaMA, Falcon, MPT inference.

---

315. Q: What is FlashAttention-2? A: Next-gen version of FlashAttention.

- **Optimizations:**
  - Parallelism at warp level.
  - Better handling of long sequences.
- **Impact:** State-of-the-art attention kernel in 2023–24 models.

---

316. Q: What are fused GPU kernels? A: Combine multiple GPU operations into one kernel launch.

- **Example:** Bias + activation fused into one op.
- **Benefit:** Less memory I/O, better GPU utilization.
- **Frameworks:** Apex, xFormers.

---

317. Q: What is TensorRT? A: NVIDIA's high-performance inference SDK.

- **Features:** Layer fusion, FP16/INT8 quantization, kernel auto-tuning.
- **Use case:** Deploying LLMs on GPUs with extreme latency constraints.

---

318. Q: What is ONNX Runtime for LLMs? A: Inference runtime for ONNX models.

- **Features:** Graph optimizations, quantization, hardware acceleration.
- **Supports:** CPU, GPU, specialized accelerators.

---

319. Q: What is operator fusion in inference? A: Merge multiple ops into fewer computational steps.

- **Example:** MatMul + Add + LayerNorm fused.
- **Impact:** Lower latency, reduced memory bandwidth usage.

---

320. Q: What is activation recomputation (checkpointing)? A: Recompute activations during backward pass instead of storing them.

- **Trade-off:** Saves memory at cost of extra compute.
- **Inference analog:** Sometimes used to reduce memory peak.

---

321. Q: What is speculative beam search? A: Combination of speculative decoding + beam search.

- **Mechanics:** Draft model proposes multiple beams, large model verifies.
- **Goal:** Faster decoding with diverse outputs.

---

322. Q: What is early exit in LLM inference? A: Stop computation at an intermediate layer if confidence is high.

- **Analogy:** Like humans answering easy questions quickly.
- **Benefit:** Latency reduction, especially for shorter outputs.

---

323. Q: What is dynamic layer skipping? A: Adaptively skip some Transformer layers at inference.

- **Method:** Use confidence gating networks.
- **Benefit:** Faster inference, lower power consumption.

---

324. Q: What is MoE routing in inference? A: In Mixture-of-Experts, only activate a subset of experts per token.

- **Routing:** Gating function decides which experts to use.
- **Benefit:** Reduces compute cost drastically.

---

325. Q: What is load balancing in MoE inference? A: Ensure tokens are evenly distributed across experts.

- **Challenge:** Without balance, some experts overload, others idle.
- **Solution:** Add load-balancing loss during training.

---

326. Q: What is tensor slicing vs tensor parallelism? A:

- **Tensor slicing:** Split tensors into chunks for storage.
- **Tensor parallelism:** Split tensors across GPUs for computation.
- **Both used in:** Scaling large models across hardware.

---

327. Q: What is inference pipeline parallelism? A: Same as training pipeline parallelism, but optimized for latency.

- **Problem:** Bubbles (idle GPUs).
- **Solution:** Micro-batching + overlapping compute/comm.

328. Q: What is request-level parallelism? A: Serve multiple user requests in parallel on one GPU.

- **Frameworks:** vLLM (paged KV cache), TGI.
- **Impact:** Higher GPU utilization, lower cost per request.

---

329. Q: What is token-level parallelism? A: Generate multiple tokens per step instead of one.

- **Speculative decoding** is a form of token-level parallelism.
- **Goal:** Reduce autoregressive bottleneck.

---

330. Q: What is multi-query attention (MQA)? A: Optimization of attention mechanism.

- **Idea:** Share keys/values across all heads, only queries are separate.
- **Benefit:** Reduces KV cache size drastically.
- **Used in:** PaLM, LLaMA-2, Falcon.

Expanded Q&A (331–350)

331. Q: What is Retrieval-Augmented Generation (RAG)? A: A method that augments LLMs with external knowledge retrieval.

- **Flow:** Query → retrieve documents from DB → feed into LLM.
- **Benefit:** Keeps model updated without retraining.
- **Use cases:** Chatbots, legal/medical assistants, enterprise knowledge systems.

---

332. Q: Why use RAG instead of fine-tuning? A:

- **RAG:** Cheap, dynamic, avoids catastrophic forgetting.
- **Fine-tuning:** Expensive, static, bakes knowledge into weights.
- **Best practice:** Fine-tune for reasoning style, use RAG for factual updates.

---

333. Q: What are embeddings in RAG? A: Numerical vector representations of text.

- **Property:** Semantically similar texts → close in vector space.
- **Use:** Index & retrieve documents with cosine similarity / dot product.

---

334. Q: What is cosine similarity in embeddings? A: A measure of similarity between two vectors.

- **Formula:** `cos(θ) = (A · B) / (||A|| * ||B||)`
- **Range:** -1 to 1 (1 = identical direction).
- **Used in:** Semantic search for RAG.

---

335. Q: What is a vector database? A: Database specialized for storing and searching embeddings.

- **Examples:** FAISS, Pinecone, Weaviate, Milvus.
- **Features:** Approximate nearest neighbor (ANN) search, filtering, metadata indexing.

---

336. Q: What is Approximate Nearest Neighbor (ANN) search? A: Algorithm to find nearest vectors efficiently in high dimensions.

- **Problem:** Exact nearest neighbor is too slow for millions of vectors.
- **ANN techniques:** HNSW graphs, product quantization, locality-sensitive hashing.

---

337. Q: What is FAISS? A: Facebook AI Similarity Search library.

- **Supports:** GPU acceleration, ANN indexing.
- **Widely used in:** RAG pipelines for efficient embedding search.

---

338. Q: What is Pinecone? A: Managed vector database service.

- **Features:** Scalable ANN search, multi-tenant support, REST API.
- **Advantage:** No need to manage infra, production-ready.

---

339. Q: What is Weaviate? A: Open-source vector database with hybrid search.

- **Supports:** Semantic search + symbolic filters.
- **Unique feature:** Schema-based, integrates with GraphQL queries.

---

340. Q: What is hybrid search? A: Combines vector similarity + keyword search.

- **Why:** Pure embeddings may miss exact keywords (e.g., "GPT-4.0").
- **Benefit:** Improves precision and recall in retrieval.

---

341. Q: What is dense vs sparse retrieval? A:

- **Dense retrieval:** Uses embeddings (BERT, sentence transformers).

- **Sparse retrieval:** Uses keyword frequency (BM25, TF-IDF).
- **Hybrid:** Combine both for robustness.

---

342. Q: What is BM25? A: Classic ranking algorithm for keyword search.

- **Improves on TF-IDF** by normalizing for document length.
- **Still used in search engines** alongside dense methods.

---

343. Q: What are Sentence Transformers? A: Models that map sentences into dense vectors.

- **Examples:** `all-MiniLM-L6-v2`, `multi-qa-mpnet-base-dot-v1`.
- **Use case:** Embeddings for semantic search and RAG.

---

344. Q: What is retrieval latency bottleneck? A: Retrieving docs from vector DB may be slower than LLM inference.

- **Optimizations:**
  - Pre-filtering with keywords.
  - Caching frequent queries.
  - ANN index tuning.

---

345. Q: What is re-ranking in RAG? A: Second-stage scoring of retrieved docs.

- **Flow:** Retrieve top 50 docs → re-rank with cross-encoder → pick best 5.
- **Benefit:** Higher precision.
- **Models:** MonoBERT, ColBERT.

---

346. Q: What is ColBERT retrieval? A: Contextualized late interaction retrieval model.

- **Trick:** Compute embeddings for every token, match with query tokens.
- **Benefit:** More accurate than sentence-level embeddings.

---

347. Q: What is query expansion in retrieval? A: Add related terms to user query to improve recall.

- **Example:** “heart attack” → also search for “myocardial infarction”.
- **Often done with LLMs themselves** (self-query expansion).

---

348. Q: What is chunking in RAG pipelines? A: Splitting documents into smaller passages before embedding.

- **Why:** LLM context windows are limited.
- **Best practice:** 200–500 tokens per chunk with overlap.

---

349. Q: What is retrieval fusion? A: Combine results from multiple retrieval strategies.

- **Example:** Merge dense retrieval + BM25 + knowledge graph hits.
- **Benefit:** Better coverage, more robust.

---

350. Q: What is late fusion vs early fusion in retrieval? A:

- **Late fusion:** Retrieve separately, then merge rankings.
- **Early fusion:** Combine signals (dense + sparse) before ranking.
- **Trade-off:** Late fusion = simpler, early fusion = tighter integration.

Expanded Q&A (351–370)

351. Q: What is grounding in RAG? A:

- **Definition:** Ensuring that an LLM's response is directly tied to retrieved evidence.
- **Why:** Prevents hallucinations (fabricated facts).
- **How:** Force model to cite or paraphrase retrieved chunks.
- **Example:** Instead of "GPT-4 was released in 2022," model responds "According to OpenAI docs (retrieved), GPT-4 launched in 2023."

---

352. Q: What is hallucination in LLMs? A:

- **Definition:** When an LLM generates factually incorrect but confident answers.
- **Why it happens:**
  - Language models predict patterns, not truth.
  - Missing grounding or weak retrieval.
- **Mitigation:** RAG, citations, structured outputs, knowledge constraints.

---

353. Q: What is contextual grounding in RAG? A:

- **Idea:** Retrieval depends not just on query, but full conversation context.
- **Example:** User: “When was he born?” → Needs coreference resolution to “Einstein”.
- **Implementation:** Maintain conversation memory + context-aware retrieval.

---

354. Q: What is conversational memory in RAG? A:

- **Types:**
  - **Short-term memory:** Stores last N turns in context.
  - **Long-term memory:** Uses embeddings + vector DB for historical recall.
- **Benefit:** Maintains personalization and continuity in multi-turn dialogue.

---

355. Q: What is long-term memory retrieval? A:

- **Mechanism:** Store past interactions as embeddings → recall when similar queries appear.
- **Benefit:** Personal assistants, tutoring systems, enterprise chatbots.

---

356. Q: What is retrieval collapsing? A:

- **Problem:** Retrieved passages are redundant (near duplicates).
- **Solution:** Deduplicate + diversify retrieved documents.
- **Benefit:** Maximizes coverage, avoids wasted tokens.

---

357. Q: What is top-k vs top-p retrieval? A:

- **Top-k:** Always retrieve the k most similar documents.
- **Top-p:** Retrieve documents with cumulative probability ≥ p (nucleus retrieval).
- **Trade-off:** Top-p allows more flexible coverage, like in decoding.

---

358. Q: What is adaptive retrieval size? A:

- **Idea:** Adjust number of retrieved docs based on query complexity.
- **Example:** Simple factual question → 2 docs; research question → 10 docs.
- **Benefit:** Balances precision, recall, and token cost.

---

359. Q: What is retrieval augmentation with metadata filters? A:

- **Approach:** Filter retrieval using structured fields (date, author, domain).

- **Example:** Retrieve docs about “AI” published after 2022.
- **Supported in:** Weaviate, Pinecone, Elasticsearch hybrid.

---

360. Q: What is multi-hop retrieval? A:

- **Definition:** Retrieve evidence in multiple steps for reasoning chains.
- **Example:** “Where was the president of Tesla born?” → Step 1: Tesla’s president = Elon Musk → Step 2: Elon Musk born in South Africa.
- **Models:** RAG-end2end, multi-hop QA datasets (HotpotQA).

---

361. Q: What is retrieval orchestration? A:

- **Definition:** Combining multiple retrieval sources (DB, APIs, web search).
- **Orchestrator:** Decides which retriever to call based on query type.
- **Analogy:** Like a router directing traffic to the best source.

---

362. Q: What is RAG evaluation? A:

- **Metrics:**
  - Retrieval precision/recall.
  - Answer grounding (factual consistency).
  - End-to-end task accuracy.
- **Datasets:** Natural Questions, TriviaQA.

---

363. Q: What is faithfulness evaluation? A:

- **Definition:** Measures if model output is consistent with retrieved evidence.
- **Methods:** LLM judges, fact-checking classifiers, entailment models.
- **Why:** High faithfulness = less hallucination.

---

364. Q: What is answer attribution in RAG? A:

- **Definition:** Linking generated sentences back to specific retrieved passages.
- **Implementation:** Train models to output citations.
- **Benefit:** Transparency, trust in LLM outputs.

---

365. Q: What is retrieval drift? A:

- **Definition:** Over time, embeddings and retrievers may not match user needs.
- **Example:** Embedding model trained on general corpus underperforms on medical data.
- **Mitigation:** Periodically re-train embeddings, adapt indexing.

---

366. Q: What is dynamic retrieval updating? A:

- **Definition:** Continuously ingest new documents into vector DB.
- **Use case:** News, finance, fast-changing domains.
- **Challenge:** Index rebuild costs, embedding consistency.

---

367. Q: What is retrieval caching? A:

- **Idea:** Cache frequent queries and results for faster response.
- **Benefit:** Reduces load on vector DB, lowers latency.
- **Challenge:** Ensure freshness of cached results.

---

368. Q: What is multi-vector retrieval? A:

- **Definition:** Represent one document with multiple embeddings (per sentence/paragraph).
- **Benefit:** Improves fine-grained matching.
- **Models:** ColBERT, SPLADE.

---

369. Q: What is ensemble retrieval? A:

- **Definition:** Combine outputs of multiple retrievers (dense, sparse, hybrid).
- **Benefit:** Higher recall & robustness.
- **Downside:** Increases compute cost.

---

370. Q: What is retrieval latency amortization? A:

- **Idea:** Batch queries or precompute retrieval for expected prompts.
- **Example:** FAQs in enterprise chatbot → cache embeddings + retrieval results.
- **Impact:** Faster response, reduced DB overhead.

You asked:

how do i load this to a docx

Gemini Replied:

Of course. You can convert the Markdown text I provided into a DOCX file using a few different methods. Here are the most common ones, from easiest to most powerful:

Method 1: Use an Online Markdown to DOCX Converter (Easiest)

This is the quickest way and requires no software installation.

1. **Copy all the text** I provided in the previous answer.
2. Open your web browser and search for "markdown to docx converter". A popular and reliable option is **CloudConvert**: https://cloudconvert.com/md-to-docx
3. On the website, you will see an input box. **Paste the Markdown text** into it.
4. Click the **"Convert"** button.
5. Once the conversion is complete, you can **download the `.docx` file**.

- **Pros:** Very fast, easy, no installation required.
- **Cons:** You are uploading the content to a third-party server, so don't use this for sensitive or private information.

Method 2: Using the Pandoc Command-Line Tool (Most Powerful)

Pandoc is a universal document converter and is considered the gold standard for this task. It gives you the highest quality conversion.

1. **Install Pandoc:** If you don't have it, download and install it from the official website: https://pandoc.org/installing.html
2. **Save the Markdown:**
   - Copy all the Q&A text.
   - Open a plain text editor (like Notepad on Windows, TextEdit on Mac, or VS Code).
   - Paste the text and save the file with a `.md` extension. For example, save it as `llm_qa.md`.
3. **Run the Conversion Command:**
   - Open your terminal (Command Prompt, PowerShell, or Terminal).

- Navigate to the folder where you saved your `llm_qa.md` file using the `cd` command. For example: `cd C:\Users\YourUser\Documents`
- Run the following command:

  Bash

  ```
  pandoc llm_qa.md -o llm_qa.docx
  ```

- This command tells Pandoc to take the input file (`llm_qa.md`) and create an output file (`-o`) named `llm_qa.docx` .

- **Pros:** Produces the best-formatted DOCX file, works offline, can be automated.
- **Cons:** Requires installing software and using the command line.

Method 3: Direct Copy and Paste (Quick but Imperfect)

This is the most straightforward method, but it will likely require you to fix the formatting manually.

1. **Copy** all the Markdown text.
2. Open a new, blank document in **Microsoft Word**.
3. **Paste** the text directly into the document.

Word may try to interpret some of the Markdown (like making headings bold), but it will probably not handle everything correctly, especially lists and code blocks. You will likely need to spend some time cleaning it up.

- **Pros:** Extremely fast, no extra tools needed.
- **Cons:** Formatting will be inconsistent and will require significant manual cleanup.

Recommendation: For the best result with minimal fuss, I recommend **Method 1 (Online Converter)**. If you need to do this often or want perfect formatting, taking the time to learn **Method 2 (Pandoc)** is a valuable skill.