AI-ML ENGINEER INTERVIEW

Complete Q&A Guide

PART 2: AI / ML / DL / LLMs

AI, ML & DL FUNDAMENTALS

Q1 What is Artificial Intelligence (AI)?

ANS

AI is the broad field of computer science focused on creating systems that can perform tasks that normally require human intelligence — such as reasoning, problem-solving, understanding language, recognizing patterns, and making decisions. AI encompasses both narrow AI (specialized systems like ChatGPT or image classifiers) and theoretical general AI (AGI) that can perform any intellectual task a human can.

Q2 What is Machine Learning (ML)?

ANS

ML is a subset of AI where systems learn from data to improve their performance on tasks without being explicitly programmed with rules. Instead of writing if-then rules, you feed data + expected outputs, and algorithms find patterns automatically. ML categories: Supervised Learning (labeled data), Unsupervised Learning (find patterns in unlabeled data), Reinforcement Learning (learn via rewards/penalties). ML powers recommendation systems, fraud detection, medical diagnosis, and more.

Q3 What is Deep Learning (DL)?

ANS

DL is a subset of ML that uses artificial neural networks with many layers (hence 'deep') to learn representations from raw data. Inspired by the human brain, DL excels when data is large and unstructured (images, audio, text). Key architectures: CNNs for vision, RNNs/LSTMs for sequences, and Transformers for NLP. DL powers modern LLMs (GPT, Claude), image generation (Stable Diffusion), and speech recognition.

Q4 What is the difference between AI, ML, and DL?

ANS

Think of them as nested circles:

AI (broadest): Any technique making machines act intelligently — includes rule-based systems, expert systems, ML, and DL.

ML (subset of AI): Systems that learn from data using algorithms (linear regression, random forests, SVMs).

DL (subset of ML): ML using deep neural networks. Requires large data and compute but achieves state-of-the-art on complex tasks.

Example: A spam filter is AI. A spam filter using logistic regression trained on emails is ML. A spam filter using a fine-tuned BERT model is DL.

Q5 **What is the difference between supervised, unsupervised, and reinforcement learning?**

ANS

Supervised Learning: Trains on labeled data (input → expected output). Examples: image classification (cat/dog), sentiment analysis, price prediction. Algorithms: linear regression, decision trees, neural networks.

Unsupervised Learning: Finds hidden patterns in unlabeled data. Examples: customer segmentation, anomaly detection, topic modeling. Algorithms: K-means, DBSCAN, autoencoders, PCA.

Reinforcement Learning: An agent learns by taking actions in an environment and receiving rewards or penalties. Examples: game-playing AI (AlphaGo), robot control, trading bots.

Q6 **How can we integrate AI/ML/DL into an application?**

ANS

There are several integration paths:

1. API Integration: Call a third-party model via API (OpenAI, Anthropic, Google). Fastest route — no training needed.
2. Pre-trained Models: Load models from HuggingFace, TensorFlow Hub, or PyTorch Hub and run inference locally.
3. Custom Training: Collect data → preprocess → train model → evaluate → deploy. Used when domain-specific accuracy is needed.
4. Fine-tuning: Take a pre-trained model and fine-tune it on your domain data — best of both worlds.
5. Deployment: Serve models via FastAPI, TorchServe, TensorFlow Serving, or cloud platforms (AWS SageMaker, Vertex AI).

Q7 **How do you choose the right AI/ML/DL model for an application?**

ANS

Key decision factors:

1. Data size: Small data → classical ML (Random Forest, SVM). Large data → DL.
2. Data type: Tabular → XGBoost, Random Forest. Images → CNNs, ViT. Text → Transformers. Time series → LSTM, Temporal Fusion Transformer.
3. Task type: Classification, regression, generation, clustering, object detection?
4. Latency requirements: Edge deployment needs small models (MobileNet, TinyBERT).
5. Explainability needed: Decision Trees/Linear models are interpretable; DL is a black box.
6. Budget: Training large models is expensive; consider fine-tuning or APIs.
7. Baseline first: Always start with the simplest model and increase complexity only if needed.

Q8 **What is overfitting and underfitting?**

ANS

Overfitting: Model memorizes training data including noise — performs great on training data but poorly on new data. High variance. Signs: training accuracy >> test accuracy.

Underfitting: Model is too simple to capture patterns — performs poorly on both training and test data. High bias.
Fixes for overfitting: More training data, dropout, regularization (L1/L2), cross-validation, early stopping, simpler model, data augmentation.
Fixes for underfitting: More complex model, more features, train longer, reduce regularization.

Q9 What is the bias-variance tradeoff?

ANS

Bias: Error from wrong assumptions — underfitting. Model is too simple.
Variance: Error from sensitivity to small fluctuations in training data — overfitting. Model is too complex.
The tradeoff: Decreasing bias typically increases variance and vice versa. The goal is to find the sweet spot with low bias AND low variance — achieved by choosing the right model complexity and amount of data. Regularization helps shift this balance. Cross-validation helps measure the true bias-variance profile of your model.

Q10 What is a loss function?

ANS

A loss function measures how wrong the model's predictions are compared to actual values. It quantifies the 'error' that the optimizer minimizes during training. Common loss functions:
- Regression: MSE (Mean Squared Error), MAE (Mean Absolute Error), Huber Loss
- Binary Classification: Binary Cross-Entropy (Log Loss)
- Multi-class Classification: Categorical Cross-Entropy, Sparse Categorical Cross-Entropy
- NLP/LLMs: Cross-Entropy over token predictions
- GANs: Adversarial Loss

The lower the loss, the better the model's predictions.

Q11 What is gradient descent and how does it work?

ANS

Gradient descent is the optimization algorithm used to minimize the loss function by iteratively adjusting model weights in the direction of the steepest decrease (negative gradient).

```
weight = weight - learning_rate * gradient_of_loss
```

Variants: Batch GD (all data at once — slow but stable), Stochastic GD (one sample at a time — noisy but fast), Mini-batch GD (small batches — best of both, industry standard). Advanced optimizers like Adam, AdaGrad, and RMSProp adapt the learning rate per parameter and converge faster.

Q12 What is backpropagation?

ANS

Backpropagation is the algorithm used to compute gradients of the loss function with respect to each weight in a neural network. It applies the chain rule of calculus, working backward from the output layer to the input layer. Steps: 1) Forward pass: compute predictions. 2) Compute loss. 3) Backward pass: compute gradient of loss w.r.t. each weight using chain rule. 4) Update weights using gradient descent. Modern deep learning frameworks (PyTorch, TensorFlow) implement autograd to compute backprop automatically.

Q13	What is a learning rate?
ANS	The learning rate (lr) is a hyperparameter controlling how large the weight update steps are during gradient descent. Too high: weights overshoot the minimum — training diverges. Too low: training is too slow and may get stuck in local minima. Typical values: 0.001 to 0.1. Techniques: Learning rate schedulers (reduce LR on plateau, cosine annealing), warmup (gradually increase LR at start), cyclical LR. In LLM fine-tuning, very small LRs (1e-5 to 5e-5) are used to avoid 'catastrophic forgetting'.

Q14	What is transfer learning?
ANS	Transfer learning is the practice of using a model pre-trained on a large dataset (source task) and adapting it for a different but related task (target task). Instead of training from scratch, you leverage the representations already learned. Two approaches: Feature extraction (freeze pre-trained layers, only train new classification head) and Fine-tuning (unfreeze some/all layers and retrain on new data with low LR). Examples: Using ResNet trained on ImageNet for medical imaging; fine-tuning BERT for legal document classification.

Q15	What is fine-tuning a model?
ANS	Fine-tuning is training a pre-trained model on domain-specific data to specialize its capabilities. For LLMs, fine-tuning adapts a general model (e.g., LLaMA, Mistral) to specific tasks, tones, or knowledge domains. Types: Full fine-tuning (all weights updated, expensive), LoRA/QLoRA (Low-Rank Adaptation — update small matrices, very efficient, popular for LLM fine-tuning), Instruction tuning (teach model to follow instructions), RLHF (Reinforcement Learning from Human Feedback — used by OpenAI and Anthropic). Fine-tuning beats prompting for specialized, high-volume tasks.

Q16	What is a confusion matrix?
ANS	A confusion matrix is a table summarizing the performance of a classification model: - True Positives (TP): Correctly predicted positive - True Negatives (TN): Correctly predicted negative - False Positives (FP): Predicted positive, actually negative (Type I error) - False Negatives (FN): Predicted negative, actually positive (Type II error) Derived metrics: Accuracy = (TP+TN)/Total, Precision = TP/(TP+FP), Recall = TP/(TP+FN), F1 = 2*(Precision*Recall)/(Precision+Recall). In medical AI, high recall is prioritized (don't miss diseases); in spam filters, high precision is prioritized (don't flag valid emails).

Q17	What is cross-validation?
ANS	Cross-validation is a technique to assess model performance reliably by training and evaluating on multiple data splits. K-Fold CV: Split data into k folds; train on k-1 folds, validate on remaining fold; repeat k times; average the scores. Stratified K-Fold: maintains class distribution in each fold (important for imbalanced datasets). Leave-One-Out: Extreme case where k=n (one sample per fold). Prevents over-optimistic evaluation from a single train/test split. Standard practice: 5-fold or 10-fold CV.

Q18	What is an embedding?
ANS	An embedding is a dense vector representation of data (text, images, items) in a continuous high-dimensional space, where semantically similar items are close together. Text embeddings capture meaning: 'king - man + woman ≈ queen'. Generated by neural networks (embedding layers, Transformer encoders). Applications: Semantic search, recommendation systems, RAG (retrieval-augmented generation), duplicate detection, clustering. Popular embedding models: OpenAI text-embedding-3, BGE, E5, Sentence-Transformers. Stored in vector databases for efficient similarity search.

Q19	What is a vector database and why is it used in AI?
ANS	A vector database is a specialized database designed to store, index, and efficiently search high-dimensional vector embeddings. Traditional databases search by exact matches; vector DBs search by similarity (nearest neighbor search). Critical for RAG pipelines: 1) Chunk documents into text. 2) Embed chunks with an embedding model. 3) Store vectors in vector DB. 4) At query time, embed the query and find similar chunks. 5) Feed chunks as context to LLM. Popular vector databases: Pinecone, Weaviate, Chroma, Qdrant, Milvus, pgvector (PostgreSQL extension).

Q20	What is PCA (Principal Component Analysis)?
ANS	PCA is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while retaining maximum variance. It finds 'principal components' — orthogonal directions of maximum variance. Uses: Reduce input features before ML (reduces curse of dimensionality), visualize high-dimensional data in 2D/3D, compress data, remove noise. Limitation: Components are linear combinations of features — not interpretable. For non-linear dimensionality reduction, use t-SNE or UMAP (commonly used to visualize embeddings in LLM research).

LLMs, TRANSFORMERS & NLP

Q21 What is a Transformer?

ANS

The Transformer is a neural network architecture introduced in the 2017 paper 'Attention Is All You Need' by Google. It replaced RNNs for sequence tasks by using self-attention mechanisms, enabling parallel processing and capturing long-range dependencies. Components: Multi-head self-attention, feed-forward layers, positional encoding, layer normalization, residual connections. The Transformer is the backbone of all modern LLMs including GPT, BERT, Claude, LLaMA, and Mistral. Its scalability with data and compute drove the current AI revolution.

Q22 What is attention mechanism in Transformers?

ANS

The attention mechanism allows a model to focus on relevant parts of the input when processing each element. For each token, it computes a weighted sum of all other tokens' representations, where weights (attention scores) reflect relevance.

```
Attention(Q, K, V) = softmax(QK^T / sqrt(d_k)) * V
```

Q = Query, K = Key, V = Value. Multi-head attention runs this in parallel multiple times with different weight matrices, allowing the model to attend to different aspects (syntax, semantics, coreference) simultaneously.

Q23 What is tokenization in NLP?

ANS

Tokenization is the process of splitting text into smaller units called tokens that a model can process. Types: Word-level (split by spaces), Character-level (each character is a token), Subword (most modern LLMs). Subword tokenizers like BPE (Byte Pair Encoding, used by GPT), WordPiece (BERT), and SentencePiece handle rare words and morphology well. 'unbelievable' might tokenize as ['un', 'believ', 'able']. Token count directly affects LLM inference cost and context window usage.

Q24 How are tokens counted?

ANS

Tokens are not exactly words or characters — they are subword units determined by the model's tokenizer. General estimates: 1 token ≈ 4 characters or 0.75 words in English. Code and non-English text often use more tokens. You can count tokens using the tiktoken library (OpenAI) or the tokenizer from HuggingFace.

```
import tiktoken
```

enc = tiktoken.get_encoding('cl100k_base')
tokens = enc.encode('Hello, how are you?')
print(len(tokens)) # prints token count
API costs are based on input + output tokens.

Q25 What is a context window?

ANS

The context window (or context length) is the maximum number of tokens a model can process in a single inference call — including both the input (prompt) and output (response).

	If you exceed the context window, earlier content is cut off. Context windows have grown dramatically: GPT-3 had 4K tokens, GPT-4 supports 128K, Claude supports up to 200K tokens. Larger context windows allow processing entire codebases, books, or long conversations. However, longer contexts increase computational cost (attention is $O(n^2)$ in tokens).

Q26	**What is NLP (Natural Language Processing)?**
ANS	NLP is the branch of AI focused on enabling computers to understand, interpret, and generate human language. Core NLP tasks: text classification, sentiment analysis, named entity recognition (NER), machine translation, question answering, text summarization, language generation, information extraction, POS tagging, and dependency parsing. NLP has been transformed by the Transformer architecture — modern LLMs (GPT, Claude, Gemini) are trained on massive text corpora and achieve human-level performance on many NLP benchmarks.

Q27	**What is BERT and how does it work?**
ANS	BERT (Bidirectional Encoder Representations from Transformers, Google 2018) is an encoder-only Transformer model trained on two tasks: Masked Language Modeling (predict randomly masked tokens using both left and right context — bidirectional) and Next Sentence Prediction. BERT produces rich contextual embeddings excellent for understanding tasks: sentiment analysis, NER, QA. Unlike GPT (left-to-right only), BERT uses full context in both directions. Common fine-tuned variants: RoBERTa, DistilBERT, DeBERTa.

Q28	**What is GPT and how does it work?**
ANS	GPT (Generative Pre-trained Transformer, OpenAI) is a decoder-only Transformer model trained via causal language modeling — predict the next token given all previous tokens (left-to-right autoregressive). Pre-training on massive text corpora gives it broad world knowledge. GPT-3 (175B params) demonstrated emergent few-shot capabilities. GPT-4 is multimodal (text + images). The key insight: scale (more data + parameters + compute) leads to qualitatively new capabilities. RLHF (Reinforcement Learning from Human Feedback) aligns GPT to be helpful and safe.

Q29	**What is the difference between encoder-only, decoder-only, and encoder-decoder models?**
ANS	Encoder-only (BERT family): Reads entire input bidirectionally. Best for understanding tasks: classification, NER, sentence embeddings. No text generation. Decoder-only (GPT, LLaMA, Mistral, Claude): Autoregressive text generation (next token prediction). Best for: text generation, chat, completion, code generation. Encoder-Decoder (T5, BART, mT5): Encoder reads input, decoder generates output. Best for: translation, summarization, question answering where input and output are separate sequences. Modern trend: Decoder-only models (LLMs) have become dominant for most NLP tasks due to scale.

Q30	What is RAG (Retrieval-Augmented Generation)?
ANS	RAG is a technique that combines a retrieval system with an LLM to ground responses in real, up-to-date information rather than relying solely on the model's training knowledge. Pipeline: 1. Index: Chunk documents → embed → store in vector DB 2. Retrieve: Embed user query → find top-k similar chunks from vector DB 3. Augment: Add retrieved chunks to the prompt as context 4. Generate: LLM generates answer grounded in retrieved context RAG reduces hallucinations, allows using proprietary data, and keeps information current without retraining. It's the foundation of most enterprise AI applications.

Q31	What is hallucination in LLMs and how can it be reduced?
ANS	Hallucination is when an LLM generates confident-sounding but factually incorrect or invented information. Types: Factual hallucinations (wrong facts), source hallucinations (citing non-existent papers), reasoning hallucinations (flawed logic). Reduction strategies: 1. RAG: Ground answers in retrieved real documents 2. Temperature = 0: More deterministic outputs 3. Fine-tuning on accurate domain data 4. Chain-of-thought prompting: Force step-by-step reasoning 5. Verification agents: Use another LLM to fact-check 6. Structured output: JSON schema constrains outputs 7. System prompt instructions: Tell the model to say 'I don't know'

Q32	What is temperature in LLM inference?
ANS	Temperature controls the randomness/creativity of LLM outputs by scaling the probability distribution over tokens before sampling. Temperature = 0: Greedy decoding — always pick the highest probability token. Deterministic and consistent. Temperature = 1: Standard sampling — use the model's raw probability distribution. Temperature > 1: More random, creative, unpredictable. Temperature < 1: More focused and conservative. For production AI APIs: use 0-0.2 for factual tasks, 0.7-1.0 for creative writing, 0 for structured JSON output.

Q33	What is quantization in the context of LLMs?
ANS	Quantization reduces the precision of model weights from 32-bit floating point (FP32) to lower precision (FP16, INT8, INT4) to reduce memory usage and increase inference speed with minimal accuracy loss. A 70B LLM in FP32 requires ~280GB VRAM — impossible on consumer GPUs. In INT4 (4-bit quantization), it needs only ~35GB. Tools: GGUF/llama.cpp (CPU+GPU inference), GPTQ (GPU quantization), bitsandbytes (4-bit QLoRA fine-tuning), AWQ (Activation-aware Weight Quantization). Quantization enables running large models on consumer hardware.

Q34	What is word embedding? (Word2Vec, GloVe, FastText)
ANS	Word embeddings map words to dense vectors in a continuous space where semantically related words cluster together. Word2Vec (Google, 2013): Neural network trained to predict context words. Two modes: CBOW (predict word from context) and Skip-gram (predict context from word). Enables famous analogy: king - man + woman = queen. GloVe (Stanford): Matrix factorization on co-occurrence statistics. Global context. FastText (Facebook): Extends Word2Vec to use character n-grams — handles out-of-vocabulary words and morphologically rich languages. Modern LLMs produce contextual embeddings (same word, different vectors based on context) which are far superior.

Q35	What is semantic search?
ANS	Semantic search finds results based on meaning and intent rather than exact keyword matching. Traditional search: 'fast car' doesn't find 'quick automobile'. Semantic search: both queries produce the same results because their embeddings are close in vector space. Implementation: 1) Embed all documents with a sentence transformer. 2) Store vectors in a vector DB. 3) Embed the search query. 4) Find nearest neighbors using cosine similarity or dot product. 5) Return top-k results. Semantic search is the core retrieval mechanism in RAG pipelines.

ML LIBRARIES & FRAMEWORKS

Q36 What is Pandas and NumPy?

ANS

NumPy: The foundation of Python scientific computing. Provides N-dimensional array (ndarray) — fast, memory-efficient, supports vectorized operations and broadcasting. Core of most ML libraries.

Pandas: Built on NumPy. Provides DataFrame (2D tabular data) and Series (1D). Essential for data wrangling: loading CSV/Excel/JSON, filtering, grouping, merging, handling missing values, feature engineering. The 'Excel of Python'. In ML projects: NumPy for numerical operations/model inputs, Pandas for data exploration and preprocessing.

Q37 What is SciPy and when do you use it?

ANS

SciPy is a scientific computing library built on NumPy, providing algorithms for: optimization (scipy.optimize), signal processing (scipy.signal), statistics (scipy.stats), linear algebra (scipy.linalg), integration, interpolation, sparse matrices, and spatial algorithms (KD-trees for nearest neighbor search). In ML projects, scipy.stats is used for statistical tests, scipy.spatial for distance computations, and scipy.sparse for handling large sparse feature matrices (e.g., TF-IDF). Less commonly used directly in deep learning (PyTorch/TF handle those needs).

Q38 What is Scikit-learn?

ANS

Scikit-learn (sklearn) is Python's standard classical ML library. Provides: Classification (SVM, Random Forest, Logistic Regression, KNN, Naive Bayes), Regression (Linear/Ridge/Lasso, Gradient Boosting), Clustering (K-Means, DBSCAN, Hierarchical), Dimensionality Reduction (PCA, t-SNE), Feature Engineering (scaling, encoding), Model Evaluation (cross-validation, metrics), Pipelines (chain preprocessing + model). Key strengths: consistent API (fit/transform/predict), excellent documentation, integrates with Pandas/NumPy. Not for deep learning — use PyTorch or TensorFlow for that.

Q39 What is PyTorch?

ANS

PyTorch (Facebook/Meta, 2016) is the leading deep learning framework. Key features: Dynamic computation graphs (define-by-run) — easier debugging than TensorFlow's static graphs. Autograd for automatic differentiation. GPU acceleration via CUDA. Rich ecosystem: torchvision, torchaudio, torchtext, HuggingFace Transformers. Native Python feel — code reads like numpy. Industry standard for research and increasingly for production. Used for training and fine-tuning LLMs, computer vision models, and reinforcement learning agents.

Q40 What is the difference between PyTorch and TensorFlow?

ANS

PyTorch: Dynamic computation graph, Pythonic, preferred for research, easier debugging, dominant in academia and LLM research (HuggingFace ecosystem). TensorFlow/Keras: Static computation graph (though TF 2.x added eager mode), better production tooling (TFLite, TensorFlow Serving, TensorFlow.js), Google ecosystem. In 2025, PyTorch has largely won the research battle — most state-of-the-art models (LLaMA, Mistral, Stable

	Diffusion) use PyTorch. TensorFlow is still dominant in mobile (TFLite) and Google Cloud production deployments.

Q41	**How many model formats are there and when do you use each?**
ANS	Key model formats: 1. PyTorch (.pt / .pth): Native PyTorch. Training and research. Use torch.save() / torch.load(). 2. ONNX (.onnx): Open standard for cross-framework interoperability. Export PyTorch/TF model, deploy anywhere. 3. TensorFlow SavedModel: TF production deployment. TensorFlow Serving, Vertex AI. 4. TFLite (.tflite): Mobile and edge devices (Android, Raspberry Pi). Quantized, optimized. 5. CoreML (.mlpackage): Apple devices (iPhone, Mac). Converted from PyTorch/ONNX. 6. GGUF (.gguf): llama.cpp format for running quantized LLMs on CPU/GPU. Used by Ollama. 7. Keras H5 (.h5 / .keras): Legacy Keras model save format. 8. Safetensors (.safetensors): Hugging Face safe format — faster loading, no arbitrary code execution risk. 9. GPTQ: GPU-quantized LLMs (4-bit). Used with AutoGPTQ. 10. TorchScript: Serialized PyTorch for production (avoids Python GIL).