

Comprehensive Technical Preparation Guide for the Modern AI Engineer: Machine Learning, Large Language Models, and Agentic Architectures

Executive Overview

The role of the Artificial Intelligence Engineer has evolved precipitously as the industry transitions from static predictive modeling to dynamic, agentic orchestration. Entering a technical interview in the 2025–2026 cycle requires more than a functional understanding of model training; it demands a mastery of the entire lifecycle of intelligent systems—from the probabilistic foundations of machine learning to the architectural nuances of autonomous multi-agent frameworks like CrewAI. This report serves as an exhaustive reference document designed to bridge the gap between academic theory and production-grade engineering, structured to provide definitions, architectural dependencies, strategic alternatives, and deep technical "extra knowledge" that distinguishes senior candidates.

Part I: Foundational Machine Learning (ML)

1.1 Meaning and Definition

Machine Learning (ML) fundamentally represents the mathematical formalization of generalization. It is the subfield of computer science wherein algorithms parse data, learn from it, and then make a determination or prediction about something in the world. Unlike traditional software engineering, where logic is explicitly programmed via rules (if-then statements), ML algorithms derive their logic inductively from datasets. In an interview context, you must articulate ML not just as a set of tools, but as a probabilistic framework for

function approximation. The goal is to learn a mapping function $f : X \rightarrow Y$ that minimizes the expected error on unseen data drawn from the same distribution as the training set.¹

The modern definition extends beyond simple function approximation to include representation learning, where the system discovers the representations needed for feature detection or classification from raw data. This shift is what separates classical ML from Deep Learning, where layers of features are not designed by human engineers but learned from data using a general-purpose learning procedure.²

1.2 Parts and Dependencies

The architecture of a machine learning system is composed of several interdependent components, each serving as a potential point of failure or optimization in a production environment.

1.2.1 Data Processing and Feature Engineering

Data is the substrate of any ML system. The dependencies here are critical: raw data must be transformed into feature vectors that the algorithm can process. This involves **normalization** (scaling features to a standard range, typically 0–1 or -1–1) to ensure gradients do not explode or vanish, and **handling missing values** through imputation techniques. In classical ML, feature engineering—the manual creation of input variables based on domain knowledge—is the primary driver of model performance.³ For example, in a housing price model, a raw feature might be "date sold," but a carefully engineered feature would be "years since last renovation."

1.2.2 The Model and Hypothesis Space

The model is the chosen mathematical structure (e.g., a linear equation, a decision tree, or a neural network) that defines the hypothesis space—the set of all possible functions the algorithm can learn. A "Linear Regression" model assumes the relationship between inputs and outputs is linear. Selecting a model introduces a bias dependency; a linear model cannot learn a quadratic relationship, regardless of how much data is provided.⁴

1.2.3 The Objective Function (Loss Function)

This is the compass that guides learning. It quantifies "how bad" the model's predictions are compared to the actual labels.

- **Regression:** typically uses **Mean Squared Error (MSE)**, which penalizes large errors quadratically, making it sensitive to outliers.
- **Classification:** typically uses **Cross-Entropy Loss** (Log Loss), which measures the divergence between the predicted probability distribution and the true distribution. Understanding the derivative of these functions is crucial, as it drives the optimization process.⁵

1.2.4 The Optimizer

The mechanism that updates the model's parameters to minimize the loss function. While **Stochastic Gradient Descent (SGD)** is the foundational concept, modern deep learning relies on adaptive optimizers like **Adam (Adaptive Moment Estimation)** or **AdamW** (Adam with Weight Decay). These algorithms adjust the learning rate for each parameter individually, allowing for faster convergence on complex, non-convex loss surfaces.⁶

1.3 Advantages and Alternatives

1.3.1 Supervised vs. Unsupervised Learning

- **Supervised Learning:** The dominant paradigm in industry. It requires labeled data

(x, y) . Its primary advantage is clear metrics for success (accuracy, F1 score). However, it is fundamentally limited by the cost and availability of annotated data.

- **Alternative: Unsupervised Learning** removes the dependency on labels, finding hidden structures in data x (e.g., clustering customers). While less prescriptive, it is essential for exploratory analysis and dimensionality reduction.
- **Alternative: Semi-Supervised Learning** bridges the gap, using a small set of labeled data to guide the learning on a larger pool of unlabeled data, often via pseudo-labeling techniques.¹

1.3.2 Generative vs. Discriminative Models

A frequent interview distinction.

- **Discriminative Models:** Learn the boundary between classes, modeling the conditional probability $P(Y|X)$. They answer "Given this image, is it a cat or a dog?" Examples include Logistic Regression and SVMs. They are generally more accurate for classification tasks but cannot generate new data.
- **Generative Models:** Learn the distribution of the data itself, modeling the joint probability $P(X, Y)$ or just $P(X)$. They answer "What does a cat look like?" and can generate new samples. Examples include Naive Bayes (simple) and Diffusion Models or GPT (complex). Generative models are more robust to outliers but computationally more expensive to train.⁴

1.4 Extra Knowledge: Deep Technical Insights

1.4.1 The Bias-Variance Tradeoff and "Double Descent"

Classically, the Bias-Variance tradeoff asserts that increasing model complexity decreases bias (underfitting) but increases variance (overfitting), leading to a U-shaped test error curve. However, in the era of deep learning, we observe the **Double Descent** phenomenon. As models become vastly over-parameterized (parameters $\gg$ data points), the test error initially rises (classical overfitting) but then descends again as the model interpolates the training data smoothly. This defies classical statistical wisdom and is a key concept for advanced roles.⁶

1.4.2 Evaluation Beyond Accuracy

For imbalanced datasets (e.g., fraud detection where 99.9% of transactions are valid), accuracy is a misleading metric. You must demonstrate knowledge of:

- **Precision-Recall Tradeoff:** High precision minimizes false positives (good for spam filters); high recall minimizes false negatives (good for cancer screening).
- **F1 Score:** The harmonic mean of precision and recall, providing a single metric that balances both.
- **ROC-AUC:** Measures the model's discriminatory power across all possible classification

thresholds. A model with high AUC is robust to threshold tuning.⁵

Part II: Large Language Models (LLMs)

2.1 Meaning and Definition

Large Language Models (LLMs) represent a specific class of deep learning models designed to understand, generate, and manipulate human language. Unlike traditional NLP models that were task-specific (e.g., a dedicated model for translation), LLMs are **foundation models**: they are trained on a vast corpus of data to learn a general statistical representation of language, which allows them to perform a wide variety of downstream tasks (summarization, translation, coding, reasoning) via zero-shot or few-shot inference.

At their core, LLMs are probabilistic engines trained to predict the next token in a sequence ($P(w_t | w_1, \dots, w_{t-1})$). However, due to their scale (billions of parameters) and the richness of their training data, they exhibit emergent behaviors—capabilities like logical reasoning or code synthesis that were not explicitly encoded in the training objective.¹⁰

2.2 Parts and Dependencies: The Transformer Architecture

The architectural backbone of the LLM revolution is the **Transformer**, introduced by Vaswani et al. in 2017. Understanding its components is non-negotiable for an AI engineer.¹¹

2.2.1 The Self-Attention Mechanism

The defining feature of the Transformer is its ability to process input data in parallel (unlike RNNs) and model long-range dependencies effectively.

- **Query (Q), Key (K), Value (V):** Each input token is projected into these three vectors via learned weight matrices.
 - *Query*: What the token is looking for.
 - *Key*: What the token describes.
 - *Value*: The content of the token.

- **Scaled Dot-Product Attention:** The attention score is calculated as

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- The dot product QK^T determines the similarity between a query and a key.
 - The scaling factor $\sqrt{d_k}$ prevents the dot product from growing too large, which would push the softmax function into regions with extremely small gradients, effectively halting learning (the "vanishing gradient" problem).¹¹
- **Multi-Head Attention:** The model runs this attention mechanism multiple times in parallel ("heads"). This allows the model to attend to different types of relationships simultaneously—one head might focus on syntactic agreement, while another focuses on semantic causal relationships.¹¹

2.2.2 Positional Encodings

Since the Transformer processes all tokens simultaneously, it has no inherent sense of order (unlike an RNN which processes sequentially). **Positional Encodings** are added to the token embeddings to inject information about the relative or absolute position of words.

- **Sinusoidal Encodings:** The original method using sine and cosine functions.
- **RoPE (Rotary Positional Embeddings):** The modern standard (used in Llama, Mistral). It encodes position by rotating the embedding vectors in the conceptual space, allowing for better generalization to sequence lengths longer than those seen during training.¹²

2.2.3 The Training Pipeline

The lifecycle of an LLM involves three distinct stages:

1. **Pre-training:** The computational heavy lifting. The model trains on trillions of tokens of raw text (The Pile, CommonCrawl) using a self-supervised objective (Next Token Prediction). The result is a "Base Model" that is knowledgeable but unruly.¹³
2. **Supervised Fine-Tuning (SFT):** The base model is trained on a smaller, high-quality dataset of (Instruction, Response) pairs. This aligns the model to follow user instructions rather than just continuing text.¹¹
3. **RLHF (Reinforcement Learning from Human Feedback) / DPO (Direct Preference Optimization):** The final polish. The model generates multiple responses, humans rank them, and the model is updated to favor higher-ranked responses. This is crucial for safety, reducing toxicity, and improving helpfulness. DPO is a newer, more stable alternative to PPO-based RLHF that optimizes the policy directly from preference data without a separate reward model.¹⁴

2.3 Advantages and Alternatives

2.3.1 Context Window vs. RAG

One of the primary limitations of LLMs is the **Context Window**—the maximum amount of text the model can process at once. While modern models (Gemini 1.5) support millions of tokens, filling the context window is slow and expensive.

- **Retrieval-Augmented Generation (RAG):** The standard alternative to massive context. Instead of feeding the entire library to the model, an external retrieval system (Vector Database) finds the most relevant "chunks" of text and injects them into the prompt.
 - *Advantage:* Access to potentially infinite knowledge; lower latency; easier to update knowledge (just add to the database).
 - *Dependency:* Requires a robust embedding model and vector index (e.g., HNSW).¹⁵

2.3.2 Fine-Tuning vs. Prompt Engineering

- **Prompt Engineering:** The cheapest way to steer a model. Techniques like **Chain-of-Thought (CoT)** prompting (asking the model to "think step-by-step")

significantly improve reasoning performance by allowing the model to generate intermediate computation steps.¹⁷

- **Parameter-Efficient Fine-Tuning (PEFT):** When prompting fails, we fine-tune. To avoid the cost of retraining all parameters, methods like **LoRA (Low-Rank Adaptation)** freeze the main weights and train small adapter matrices.
 - *Advantage:* Reduces trainable parameters by ~99%, enabling fine-tuning on consumer hardware.
 - *Advantage:* Adapters can be swapped hot-swapped at inference time, allowing one base model to serve many different specialized tasks.¹⁸

2.4 Extra Knowledge: Production Considerations

2.4.1 Quantization (GGUF, GPTQ, AWQ)

In production, running models at full 16-bit precision (FP16) is often wasteful. **Quantization** compresses weights to 4-bit or 8-bit integers.

- **GGUF:** A file format designed for CPU inference (especially Apple Silicon). It allows splitting layers between CPU and GPU.
- **GPTQ (Generative Pre-trained Transformer Quantization):** A post-training quantization method that focuses on minimizing the reconstruction error of the weights layer-by-layer. Ideal for GPU serving.
- **AWQ (Activation-Aware Weight Quantization):** A newer method that recognizes not all weights are equal. It preserves the precision of the most "important" weights (those with large activation magnitudes), resulting in better performance than GPTQ at the same compression level.²⁰

2.4.2 Hallucinations and Grounding

Hallucinations are not bugs; they are a feature of the probabilistic nature of LLMs. They occur when the model creates a plausible-sounding completion that is factually incorrect.

- **Mitigation:** Beyond RAG, techniques like **Self-Consistency** (sampling multiple answers and taking the majority vote) and **Citation-Based Decoding** (forcing the model to reference source chunks) are critical for trusted systems.¹⁴

Part III: Agentic AI

3.1 Meaning and Definition

Agentic AI represents the transition from "tools" to "teammates." While a standard LLM is a passive engine (Input $\rightarrow$ Output), an **AI Agent** is an active system capable of pursuing complex goals. It is defined by its ability to:

1. **Perceive** its environment.
2. **Reason** about how to bridge the gap between the current state and the goal state.

3. **Act** upon the environment using tools (APIs, code execution).
4. **Observe** the result of the action and adapt its plan.

This loop—Perception, Reasoning, Action, Observation—is the heartbeat of an agentic system. It shifts the burden of orchestration from the human user to the AI system itself.²³

3.2 Parts and Dependencies

3.2.1 The Cognitive Architecture (The "Brain")

The core of an agent is the LLM, but it is wrapped in a cognitive architecture that structures its thinking. The most famous pattern is **ReAct (Reason + Act)**.

- *Thought*: "The user wants to know the weather in Tokyo. I should use the weather tool."
- *Action*: `weather_tool("Tokyo")`
- *Observation*: "22°C, Cloudy."
- *Thought*: "I have the answer. I will reply to the user." Without this structure, the LLM is just a text generator; with it, it becomes a problem solver.²⁴

3.2.2 Tools (The "Hands")

Agents interact with the world via Tools. A Tool is simply a function that the agent can invoke.

- **Function Calling**: The mechanism by which the LLM outputs structured JSON (e.g., `{"function": "search", "args": {"query": "CrewAI docs"}}`) instead of unstructured text. This deterministic output is parsed by the runtime to execute the actual code.¹⁷

3.2.3 Memory (The Context)

For an agent to operate over time, it needs memory systems deeper than the context window.

- **Short-Term Memory**: The immediate context of the current reasoning chain.
- **Long-Term Memory**: Usually implemented via a Vector Database (RAG), storing the history of past interactions or a knowledge base of facts.
- **Entity Memory**: A structured record of specific entities (users, companies) and their attributes, often maintained in a key-value store or knowledge graph.²⁵

3.3 Advantages and Alternatives

3.3.1 Single vs. Multi-Agent Systems

- **Single Agent**: Simple, easy to debug. Good for linear tasks (e.g., "Summarize this URL").
- **Multi-Agent Systems**: Specialized agents collaborate. One agent might be the "Researcher," another the "Writer," and a third the "Critic."
 - *Advantage: Specialization*. A "Coder" agent can be prompted with specific coding best practices, while the "Reviewer" agent is prompted with security guidelines. This separation of concerns leads to higher quality outcomes than trying to stuff all instructions into one prompt.²⁷

3.3.2 Frameworks: LangGraph vs. AutoGen vs. CrewAI

- **LangGraph:** Provides low-level control. You define the graph of nodes (functions) and edges (control flow). It is ideal for complex, stateful applications where you need to handle cycles and precise error recovery. *Analogy: Writing in C++.*²⁸
- **AutoGen:** Focuses on conversation. Agents are conversationalists that talk to each other to solve tasks. It is excellent for code generation and open-ended exploration but can be hard to steer in rigid business processes. *Analogy: A chatroom of experts.*²⁸
- **CrewAI:** Focuses on process and roles. It models a "Crew" of employees with specific roles and tasks. It is highly opinionated and structured, making it the fastest path to building production business automation. *Analogy: Managing a Jira board.*²⁸

3.4 Extra Knowledge: Advanced Patterns

3.4.1 Reflection and Self-Correction

Advanced agents don't just act; they reflect. The **Reflection Pattern** involves the agent reviewing its own output before finalizing it. For example, a "Coder" agent generates code, and a "Reviewer" agent (or the same agent in a different mode) analyzes the code for errors. If errors are found, the "Coder" is tasked with fixing them. This loop mimics the human drafting process.²⁴

3.4.2 Planning Architectures

- **Chain of Thought (CoT):** Linear step-by-step reasoning.
- **Tree of Thoughts (ToT):** The agent explores multiple possible reasoning paths simultaneously, backtracking if a path leads to a dead end. This is computationally expensive but necessary for complex problem-solving.¹⁷

Part IV: Deep Dive into CrewAI

4.1 Meaning and Definition

CrewAI is an open-source framework designed to orchestrate role-playing, autonomous AI agents. Unlike other frameworks that might focus on open-ended conversation or rigid graphs, CrewAI mimics a corporate structure: it defines **Agents** (employees), **Tasks** (assignments), and **Process** (workflow strategy). It is built on top of LangChain, leveraging its tool ecosystem, but provides a higher-level abstraction focused on **collaborative intelligence**.³¹

4.2 Parts and Dependencies

A CrewAI application is constructed from four primary primitives. A solid interview answer must detail how these interact.

4.2.1 Agents: The Specialized Workers

An Agent is an autonomous unit with a specific personality and capability set.

- **Role:** The job title (e.g., "Senior Financial Analyst").
- **Goal:** The specific objective (e.g., "Analyze stock trends and predict movement").
- **Backstory:** This is the "secret sauce" of CrewAI. By providing a rich textual backstory ("You are a veteran Wall Street analyst with a cynical view of tech stocks..."), you steer the LLM's tone and focus, reducing variance and hallucinations.³²
- **Configuration:**
 - `allow_delegation`: Determines if the agent can offload work to others (default is False).
 - `max_iter`: A safety valve to prevent infinite loops (e.g., stops after 15 attempts).
 - `memory`: Enables the agent to use Short-Term, Long-Term, and Entity memory.³³

4.2.2 Tasks: The Units of Execution

Tasks connect Agents to business value. A task is not just a prompt; it is a structured definition of work.

- **Description:** A clear, detailed explanation of what needs to be done.
- **Expected Output:** A definition of what "done" looks like (e.g., "A 3-paragraph executive summary"). This is crucial for the agent to know when to stop.
- **Context:** A powerful dependency injection mechanism. You can pass the output of Task A and Task B as context to Task C. This allows for sophisticated data pipelines where the "Writer" automatically sees the research from the "Researcher" without manual string concatenation.³¹

4.2.3 Crews: The Team Structure

The Crew object bundles Agents and Tasks and defines the execution strategy.

- **Process.sequential:** The default mode. Tasks are executed in the order they are listed. Ideal for linear pipelines.
- **Process.hierarchical:** Mimics a management structure. A "Manager Agent" (often utilizing a stronger LLM like GPT-4) is automatically created or assigned. This manager receives the high-level goal, breaks it down, delegates tasks to the worker agents, and reviews their output. This requires a `manager_llm` to be specified.³²

4.2.4 Flows: Event-Driven State Management

Introduced to handle complex, non-linear workflows, **Flows** allow you to chain multiple Crews together or create event-driven systems.

- **Decorators:**
 - `@start`: Marks the entry point of the flow.
 - `@listen`: Triggers a function when another function completes (e.g., `@listen(research_crew_finished)`).
 - `@router`: Allows for conditional branching based on the output of a step (e.g., if the research is inconclusive, route to a "Deep Search" crew; otherwise, route to "Writer" crew).

- **State:** Flows use **Pydantic** models to manage state, ensuring that data passed between steps is structured and type-safe. This is a massive advantage for production systems over unstructured dictionaries.³²

4.3 Advantages and Alternatives

4.3.1 Why Choose CrewAI?

- **Structure:** It forces a clean separation of concerns. You don't write monolithic prompts; you write modular agents and tasks.
- **Production Readiness:** Features like max_rpm (rate limiting to avoid OpenAI 429 errors), output validation (Pydantic), and built-in observability make it safer for enterprise deployment than raw LangChain scripts.³³
- **Memory Architecture:** CrewAI handles the complexity of RAG and vector stores internally, giving agents "out of the box" long-term memory.²⁵

4.3.2 Comparative Table

Feature	CrewAI	AutoGen	LangGraph
Core Abstraction	Role-based (Agents/Tasks)	Conversational (Agents chatting)	Graph (Nodes/Edges)
Control Flow	Sequential or Hierarchical	Dynamic/Conversation-driven	Explicit State Machine
Best For	Business Process Automation	Coding/Exploratory Tasks	Complex, Cyclic Logic
Learning Curve	Low/Medium	Medium	High

4.4 Extra Knowledge: Production Engineering with CrewAI

4.4.1 Rate Limiting and Cost Control

In production, "lazy" agents can burn through API credits.

- **Max RPM:** Set the max_rpm (Requests Per Minute) on your agents to ensure you stay within your API tier limits.
- **Model Selection:** Use a "smart" model (GPT-4o, Claude 3.5 Sonnet) for the **Manager** or **Planner** agent, but cheaper/faster models (GPT-4o-mini, Haiku) for execution agents (e.g., Summarizer). This optimizes the cost/performance ratio.³⁶

4.4.2 Handling Context Window Limits

A common failure mode is the "Context Length Exceeded" error.

- **Mechanism:** CrewAI agents accumulate history. If a task takes too many steps, the history blows up the context window.
- **Solution:** Enable respect_context_window=True. This feature automatically summarizes the agent's past thoughts and actions when the limit approaches, preserving the

essential context while discarding the verbose logs.³³

4.4.3 Custom Tools and Error Handling

Writing custom tools is a primary task for an AI engineer.

- **Pydantic Arguments:** Always define your tool's input arguments using Pydantic models. This ensures the LLM knows exactly what fields to provide (and their types), reducing `ValidationError` exceptions.
- **Error Bubbling:** When a tool fails (e.g., API 500 error), the error message should be returned *to the agent* (not just logged to the console). This allows the agent to self-correct (e.g., "The API failed, I will try again" or "I will use a different search query").³⁷

4.4.4 Observability

Deploying agents blind is dangerous.

- **Tracing:** Integrate with **LangSmith** or **Arize Phoenix**. These tools visualize the "trace" of the agent—showing the exact prompt sent to the LLM, the raw response, the tool output, and the latency of each step. This is essential for debugging why an agent got stuck in a loop or gave a bad answer.²³

Conclusion and Interview Strategy

The modern AI interview is a test of breadth and depth. You must be able to pivot from the mathematical definition of **Cross-Entropy Loss** to the architectural implementation of a **Hierarchical Crew** in Python.

Key Takeaways to articulate:

1. **ML is the Foundation:** Understand that LLMs are just probability distributions learned via self-supervision.
2. **Architecture Matters:** Know *why* you would choose RAG over Fine-tuning (Dynamic data vs. Style/Behavior) and *why* you would use a Multi-Agent system over a single prompt (Specialization/Reliability).
3. **CrewAI is for Process:** Position CrewAI as the tool for "operationalizing" LLMs—taking the raw intelligence and constraining it into reliable business workflows via Agents, Tasks, and strong Process management.
4. **Production is Hard:** Be ready to discuss the "unhappy paths"—hallucinations, loops, rate limits, and context exhaustion—and how you engineer around them.

By mastering the definitions, dependencies, and architectural patterns detailed in this report, you position yourself not just as a user of AI tools, but as an architect of intelligent systems.

Works cited

1. Top 35 Machine Learning Interview Questions For 2026 - DataCamp, accessed February 15, 2026, <https://www.datacamp.com/blog/top-machine-learning-interview-questions>

2. Deep Learning vs Machine Learning: Key Differences - iSchool, accessed February 15, 2026,
<https://ischool.syracuse.edu/deep-learning-vs-machine-learning/>
3. Machine Learning vs Deep Learning: Methods and Insights | Databricks, accessed February 15, 2026,
<https://www.databricks.com/glossary/machine-learning-vs-deep-learning>
4. Top 50+ Machine Learning Interview Questions and Answers [2026] - NetCom Learning, accessed February 15, 2026,
<https://www.netcomlearning.com/blog/machine-learning-interview-questions>
5. Machine Learning Interview Questions and Answers - GeeksforGeeks, accessed February 15, 2026,
<https://www.geeksforgeeks.org/machine-learning/machine-learning-interview-questions/>
6. andrewekhale/MLQuestions: Machine Learning and Computer Vision Engineer - Technical Interview Questions - GitHub, accessed February 15, 2026,
<https://github.com/andrewekhale/MLQuestions>
7. The Ethical Interview: How to Discuss AI Bias Responsibly | by Intonix AI - Medium, accessed February 15, 2026,
<https://medium.com/@intonix.ai/the-ethical-interview-how-to-discuss-ai-bias-responsibly-e6d469984f4a>
8. Top ML Interview Questions (2026 Guide) - Exponent, accessed February 15, 2026,
<https://www.tryexponent.com/blog/top-machine-learning-interview-questions>
9. Machine Learning Interview Questions & Answers for Data Scientists.md - GitHub, accessed February 15, 2026,
<https://github.com/youssefHosni/Data-Science-Interview-Questions-Answers/blob/main/Machine%20Learning%20Interview%20Questions%20%26%20Answers%20for%20Data%20Scientists.md>
10. LLM Interview Questions & Answers (2026): A Complete Guide for Engineers - LockedIn AI, accessed February 15, 2026,
<https://www.lockedinai.com/blog/llm-interview-questions-answers-complete-guide-engineers>
11. Mastering Transformers: The 2025 AI Interview Prep Guide ..., accessed February 15, 2026,
<https://www.sundeepteki.org/advice/the-transformer-revolution-the-ultimate-guide-for-ai-interviews>
12. LLM & Transformer Interview Essentials A-Z, Part 1: Architecture Fundamentals - Medium, accessed February 15, 2026,
<https://medium.com/@profxfang/llm-transformer-interview-essentials-a-z-part-1-architecture-fundamentals-054c91114843>
13. LLM (ML) Job Interviews - Resources - Mimansa Jaiswal, accessed February 15, 2026, <https://mimansajaiswal.github.io/posts/llm-ml-job-interviews-resources/>
14. Top Generative AI and LLM Interview Question with Answer ..., accessed February 15, 2026,
<https://www.geeksforgeeks.org/artificial-intelligence/generative-ai-interview-que>

[stion-with-answer/](#)

15. Top 30 RAG Interview Questions and Answers for 2026 - DataCamp, accessed February 15, 2026, <https://www.datacamp.com/blog/rag-interview-questions>
16. Top 9 Fine-tuning Interview Questions and Answers - Analytics Vidhya, accessed February 15, 2026, <https://www.analyticsvidhya.com/blog/2024/04/fine-tuning-interview-questions-and-answers/>
17. Top 30 Agentic AI Interview Questions and Answers for 2026 | DataCamp, accessed February 15, 2026, <https://www.datacamp.com/blog/agentic-ai-interview-questions>
18. Gen AI Fine-Tuning Techniques: LoRA, QLoRA, And Adapters Compared - Digitaldividedata.com, accessed February 15, 2026, <https://www.digitaldividedata.com/blog/ai-fine-tuning-techniques-lora-qlora-and-adapters>
19. Fine-Tuning of Large Language Models with LoRA and QLoRA, accessed February 15, 2026, <https://www.analyticsvidhya.com/blog/2023/08/lora-and-qlora/>
20. Which Quantization Method Is Best for You?: GGUF, GPTQ, or AWQ... | E2E Networks, accessed February 15, 2026, <https://www.e2enetworks.com/blog/which-quantization-method-is-best-for-you-gguf-gptq-or-awq>
21. Benchmarking Quantized LLMs: What Works Best for Real Tasks? - Ionio, accessed February 15, 2026, <https://www.ionio.ai/blog/llm-quantize-analysis>
22. LLM Quantization Techniques. GGUF GPTQ AWQ BitNet | by joydeep bhattacharjee, accessed February 15, 2026, <https://joydeep31415.medium.com/llm-quantization-techniques-4229b7eac20c>
23. 30 Agentic AI Interview Questions and Answers: From Beginner to Advanced, accessed February 15, 2026, <https://www.analyticsvidhya.com/blog/2026/02/agentic-ai-interview-questions-and-answers/>
24. Agentic AI : Concepts, Architecture, Design Patterns and Interview ..., accessed February 15, 2026, <https://medium.com/@subhransun20/agentic-ai-concepts-architecture-design-patterns-and-interview-preparations-6fc784425494>
25. How to Build AI Agents with Redis Memory Management, accessed February 15, 2026, <https://redis.io/blog/build-smarter-ai-agents-manage-short-term-and-long-term-memory-with-redis/>
26. Short-Term vs Long-Term Agent Memory: A Deep Dive - Sparkco, accessed February 15, 2026, <https://sparkco.ai/blog/short-term-vs-long-term-agent-memory-a-deep-dive>
27. AI Agentic Design Patterns Interview Questions and Answers | by Dr. Sanjay Kumar, accessed February 15, 2026, <https://skphd.medium.com/agentic-design-patterns-interview-questions-and-answers-37f80cc59b3e>
28. LangGraph vs AutoGen vs CrewAI: Complete AI Agent Framework ..., accessed

February 15, 2026,

<https://latenode.com/blog/platform-comparisons-alternatives/automation-platform-comparisons/langgraph-vs-autogen-vs-crewai-complete-ai-agent-framework-comparison-architecture-analysis-2025>

29. Autogen vs CrewAI vs LangGraph 2026 Comparison Guide - Python in Plain English, accessed February 15, 2026,
<https://python.plainenglish.io/autogen-vs-crewai-vs-langgraph-2026-comparison-guide-fd8490397977>
30. Agentic AI #3 — Top AI Agent Frameworks in 2025: LangChain, AutoGen, CrewAI & Beyond | by Aman Raghuvanshi | Medium, accessed February 15, 2026,
<https://medium.com/@iamanraghuvanshi/agentic-ai-3-top-ai-agent-frameworks-in-2025-langchain-autogen-crewai-beyond-2fc3388e7dec>
31. Crews - CrewAI Documentation, accessed February 15, 2026,
<https://docs.crewai.com/en/concepts/crews>
32. Introduction - CrewAI, accessed February 15, 2026,
<https://docs.crewai.com/en/introduction>
33. Agents - CrewAI Documentation, accessed February 15, 2026,
<https://docs.crewai.com/en/concepts/agents>
34. Processes - CrewAI Documentation, accessed February 15, 2026,
<https://docs.crewai.com/en/concepts/processes>
35. CrewAI: Scaling Human-Centric AI Agents in Production | by Takafumi Endo | Medium, accessed February 15, 2026,
<https://medium.com/@takafumi.endo/crewai-scaling-human-centric-ai-agents-in-production-a023e0be7af9>
36. How Small Language Models Are Key to Scalable Agentic AI | NVIDIA Technical Blog, accessed February 15, 2026,
<https://developer.nvidia.com/blog/how-small-language-models-are-key-to-scalable-agentic-ai/>
37. CrewAI Troubleshooting: Common Issues and Solutions, accessed February 15, 2026,
<https://www.wednesday.is/writing-articles/crewai-troubleshooting-common-issues-and-solutions>