AI / ML INTERVIEW PREPARATION HANDBOOK

Questions & Answers based on the 49 skills in your job profile

Section	Skills	Focus
Must-have	20	Python, FastAPI, APIs, async, Redis, LLMs, OpenAI, voice, RAG
Good-to-have	29	JS/TS, prompting, tools, vector DBs, Hugging Face, agents, guardrails
Practical	10	System design, production architecture, troubleshooting, security

How to use this handbook

1. First pass: learn the short answers. 2. Second pass: explain each answer in your own words. 3. Third pass: practice the scenario questions aloud. 4. For tools/providers, verify current API details against the official documentation before an interview because model and SDK behavior can change.

Interview strategy: For architecture questions, answer in this order: requirements → architecture → data flow → scaling → security → failure handling → observability → testing.

Contents

Part 1 — Must-have skills

1. Python
2. FastAPI
3. Django
4. REST APIs
5. WebSockets
6. Async Programming
7. Microservices
8. Redis
9. LLMs
10. OpenAI APIs
11. GPT-4o
12. GPT-5
13. Claude
14. Speech-to-Text
15. Text-to-Speech
16. OpenAI Realtime API
17. Embeddings
18. Vector Databases
19. OpenAI SDK
20. LangChain

Part 2 — Good-to-have skills

1. JavaScript
2. TypeScript
3. Gemini
4. Prompt Engineering
5. Function Calling
6. Structured Outputs
7. JSON Schema
8. Context Window Management
9. Token Optimization
10. AI Safety
11. Model Evaluation

12. Pinecone
13. Weaviate
14. Qdrant
15. Milvus
16. LlamaIndex
17. Hugging Face Transformers
18. Sentence Transformers
19. Pydantic AI
20. Conversational AI
21. Voice AI
22. AI Assistants
23. Workflow Automation
24. MCP
25. AI Guardrails
26. AI Evaluation Frameworks
27. streaming audio
28. voice activity detection
29. embeddings

Part 3 — Practical AI/ML & system-design questions

Part 4 — Rapid revision checklist

Part 1 — Must-have skills

Python

Q1. What are Python decorators?

Answer: Decorators wrap a function/class to add behavior without changing its core code. They are commonly used for logging, authentication, caching, timing, and FastAPI dependencies.

Q2. What is the difference between a list, tuple, set, and dictionary?

Answer: A list is ordered and mutable; a tuple is ordered and immutable; a set stores unique values; a dictionary stores key-value pairs. Choose based on mutability, uniqueness, and lookup needs.

Q3. How does Python manage memory?

Answer: Python uses private heaps, reference counting, and a cyclic garbage collector. Objects are allocated on the heap and released when no longer reachable.

Q4. What is a generator?

Answer: A generator produces values lazily using yield, so it can process large data without keeping the whole result in memory.

FastAPI

Q1. Why use FastAPI for AI services?

Answer: It provides type-driven request validation with Pydantic, async support, automatic OpenAPI documentation, dependency injection, and good performance for API-heavy workloads.

Q2. How would you structure a production FastAPI service?

Answer: Separate routers, schemas, services, repositories, configuration, authentication, background jobs, and tests. Keep business logic out of route handlers and use environment-based configuration.

Q3. How do you handle long-running AI work?

Answer: Do not block the request worker. Use a job queue/background worker for heavy inference or document processing, return a job ID, and expose status/result endpoints.

Q4. How do you secure a FastAPI API?

Answer: Use authentication/authorization, input validation, rate limits, secure headers, secret management, strict CORS, request-size limits, safe file handling, logging, and dependency updates.

Django

Q1. What is Django's MVT pattern?

Answer: Model represents data, View contains request/business handling, and Template renders presentation. Django also provides ORM, authentication, admin, middleware, and routing.

Q2. When would you choose Django over FastAPI?

Answer: Django is strong when you need a batteries-included web platform, ORM, admin, authentication, and conventional server-side application features. FastAPI is often attractive for API-first and async services.

REST APIs

Q1. What makes an API RESTful?

Answer: It uses resource-oriented URLs, standard HTTP methods/status codes, stateless requests, representations such as JSON, and predictable semantics.

Q2. PUT vs PATCH?

Answer: PUT generally replaces the resource representation; PATCH applies a partial update. The exact API contract should document validation and idempotency behavior.

Q3. Which status codes should an API commonly use?

Answer: 200/201 for successful operations, 204 for success without a body, 400 for invalid requests, 401 unauthenticated, 403 unauthorized, 404 not found, 409 conflict, 422 validation errors, and 429 rate limiting.

Q4. How do you version APIs?

Answer: Use a deliberate compatibility strategy such as /v1 paths, headers, or media types. Keep old versions stable while introducing documented breaking changes.

WebSockets

Q1. When should you use WebSockets instead of REST?

Answer: Use WebSockets for persistent bidirectional, low-latency communication such as live dashboards, agent state, chat, or streaming events. REST is better for ordinary request/response operations.

Q2. How do you scale WebSockets across multiple instances?

Answer: Use a shared broker/pub-sub layer such as Redis or Kafka for cross-instance events, maintain connection state carefully, and use heartbeats/reconnect logic.

Async Programming

Q1. What does async/await solve?

Answer: It allows cooperative concurrency for I/O-bound work. While one operation waits on network or disk I/O, the event loop can run other tasks.

Q2. What is a common async mistake?

Answer: Calling blocking CPU-heavy or synchronous I/O code directly inside the event loop. Move blocking work to workers/executors or use async-compatible libraries.

Microservices

Q1. What is a microservice?

Answer: A relatively independently deployable service organized around a bounded business capability, with explicit interfaces and ownership.

Q2. What are the main trade-offs?

Answer: Benefits include independent deployment and scaling; costs include distributed-system complexity, observability, networking failures, data consistency challenges, and operational overhead.

Redis

Q1. What is Redis commonly used for?

Answer: Caching, sessions, rate limiting, distributed locks, counters, queues, pub/sub, and short-lived state.

Q2. What is a Redis distributed lock?

Answer: A mechanism that lets one worker claim exclusive access to a resource for a bounded time. Production designs need unique tokens, expiration, safe release, and careful failure handling.

LLMs

Q1. What is an LLM?

Answer: A large neural language model trained to predict tokens from context. Modern LLM applications use prompting, tool use, retrieval, structured outputs, and application-side controls.

Q2. Temperature vs top-p?

Answer: Both influence sampling diversity. Temperature rescales token probabilities; top-p limits sampling to a probability mass. Avoid tuning both blindly; choose based on observed behavior.

Q3. What causes hallucinations?

Answer: The model optimizes for plausible token sequences rather than guaranteed factual truth. Missing context, ambiguous prompts, weak retrieval, and overconfident generation can increase the risk.

Q4. How do you evaluate an LLM application?

Answer: Define task-specific metrics such as correctness, groundedness, refusal quality, latency, cost, and tool-call success, then test against a representative evaluation set.

OpenAI APIs

Q1. What should you consider when integrating an LLM API?

Answer: Authentication, request/response schemas, retries, timeouts, rate limits, token/cost tracking, structured outputs, safety controls, observability, and model/version strategy.

Q2. How do you make an LLM API integration reliable?

Answer: Use bounded retries with exponential backoff for transient failures, timeouts, idempotency where applicable, fallback behavior, validation, logging, and graceful degradation.

GPT-4o

Q1. How would you discuss GPT-4o in an interview?

Answer: Describe it as a multimodal GPT model family and focus on application design: text/vision capabilities, API integration, latency/cost trade-offs, structured outputs, and evaluation rather than memorizing benchmark numbers.

Q2. How would you select a model for a production feature?

Answer: Compare required capability, latency, context needs, reliability, tool use, safety, and cost using a task-specific evaluation set.

GPT-5

Q1. How should you approach GPT-5 in an interview?

Answer: Focus on the model's role in an application: reasoning/task capability, context handling, tool integration, structured outputs, latency, cost, and evaluation. Avoid assuming that a newer model is automatically better for every task.

Q2. How do you make an application resilient to model changes?

Answer: Keep prompts/configuration versioned, validate outputs, maintain regression tests, evaluate candidate models on the same dataset, and separate model-specific logic from business logic.

Claude

Q1. What is important when integrating Claude or another LLM provider?

Answer: Use a provider abstraction where practical, validate outputs, track model/version, handle provider-specific limits, and keep prompts/evaluations under version control.

Q2. Why use multiple model providers?

Answer: Potential benefits include resilience, cost/capability optimization, and avoiding dependence on one provider. The downside is more integration and evaluation complexity.

Speech-to-Text

Q1. What is speech-to-text?

Answer: Automatic speech recognition converts audio into text. Production systems must handle noise, accents, overlapping speech, sampling formats, latency, and domain vocabulary.

Q2. How would you improve transcription quality for a call-center app?

Answer: Use appropriate audio preprocessing, correct sample rate/codec, domain vocabulary, speaker segmentation where available, confidence checks, and evaluation on real call recordings.

Text-to-Speech

Q1. What is text-to-speech?

Answer: TTS converts text into synthesized speech. Production considerations include voice quality, latency, pronunciation, streaming, language support, and interruption handling.

Q2. How do you design low-latency voice responses?

Answer: Stream generated text/audio where supported, keep prompts compact, precompute reusable messages, use fast models/services, and design barge-in and cancellation correctly.

OpenAI Realtime API

Q1. What is the purpose of a realtime voice API?

Answer: It supports low-latency interactive audio conversations by combining streaming input/output with conversational model processing.

Q2. What are key engineering concerns?

Answer: WebSocket/session lifecycle, audio formats, interruptions, buffering, latency, authentication, tool calls, reconnection, observability, and preventing duplicate events.

Embeddings

Q1. What is an embedding?

Answer: An embedding is a vector representation that places semantically related content near each other in vector space.

Q2. How do you choose an embedding strategy?

Answer: Match the embedding model to language/domain needs, chunk documents appropriately, normalize or use the metric expected by the vector store, and evaluate retrieval quality on real queries.

Vector Databases

Q1. What is a vector database used for?

Answer: It stores and searches vector representations for semantic retrieval, often combined with metadata filters and keyword search.

Q2. What is approximate nearest neighbor search?

Answer: ANN methods trade a small amount of exactness for much faster similarity search at scale. Index parameters control the speed/recall trade-off.

OpenAI SDK

Q1. Why use an SDK instead of raw HTTP calls?

Answer: An SDK provides typed/request helpers, authentication handling, response models, and a more convenient interface. You still need application-level validation, retries, logging, and security.

Q2. How should SDK code be organized?

Answer: Wrap provider calls in a service/client layer so routes and business logic do not depend directly on provider-specific details.

LangChain

Q1. What problem does LangChain solve?

Answer: It provides abstractions for composing LLM prompts, models, tools, retrieval, agents, and application workflows.

Q2. When should you not use a framework?

Answer: If a workflow is simple, direct API calls can be easier to understand, test, and operate. Add framework abstractions when they reduce complexity rather than merely adding layers.

Part 2 — Good-to-have skills

JavaScript

Q1. var, let, const?

Answer: var is function-scoped; let and const are block-scoped. const prevents reassignment of the binding, although referenced objects can still be mutable.

Q2. Promise vs async/await?

Answer: Promises represent eventual completion; async/await is syntax built on promises that makes asynchronous control flow easier to read.

TypeScript

Q1. Why use TypeScript?

Answer: Static typing catches many errors before runtime and improves IDE support, refactoring, and API contracts.

Q2. interface vs type?

Answer: Both describe shapes; interfaces are especially useful for extensible object contracts, while type aliases are flexible for unions, intersections, and other compositions.

Gemini

Q1. How should you evaluate Gemini against another LLM?

Answer: Use the same task dataset and metrics: correctness, groundedness, latency, cost, tool success, and safety. Do not rely only on general benchmarks.

Q2. How do multi-provider apps work?

Answer: Use a provider adapter with a common internal interface, then map prompts, tool definitions, streaming, and structured output to each provider.

Prompt Engineering

Q1. What is a good production prompt?

Answer: It clearly states the task, context, constraints, output format, and failure behavior, with examples only when they improve performance.

Q2. How do you reduce prompt injection risk?

Answer: Treat retrieved/user content as untrusted data, separate instructions from data, constrain tools, validate outputs, and apply authorization outside the model.

Function Calling

Q1. What is function/tool calling?

Answer: The model produces a structured request to invoke an application-defined tool; the application executes it and returns the result to the model.

Q2. How do you secure tools?

Answer: Enforce authorization, validate arguments with schemas, apply least privilege, rate-limit sensitive operations, and log calls. Never let the model bypass application authorization.

Structured Outputs

Q1. Why use structured output?

Answer: It reduces parsing ambiguity and makes model results easier to validate and consume programmatically.

Q2. What if the model output is invalid?

Answer: Validate against a schema, retry with a constrained repair strategy if appropriate, and fail safely when validation still fails.

JSON Schema

Q1. What is JSON Schema?

Answer: A declarative specification for validating JSON structure, types, required fields, enums, and constraints.

Q2. Why is it useful with LLMs?

Answer: It gives the application a machine-checkable contract for model-generated data.

Context Window Management

Q1. Why manage context?

Answer: Long context increases cost/latency and can dilute relevant information. Applications should send only what is needed.

Q2. How can you reduce context?

Answer: Use summarization, retrieval, relevance filtering, conversation state compression, metadata, and chunking.

Token Optimization

Q1. How do you reduce token usage?

Answer: Shorten redundant instructions, retrieve only relevant chunks, avoid repeating history, use structured compact data, and select an appropriate model.

Q2. Why does token optimization matter?

Answer: It can reduce cost and latency while making application behavior more predictable.

AI Safety

Q1. What is AI safety in an application?

Answer: Controls that reduce harmful, unsafe, unauthorized, or unreliable behavior across inputs, model outputs, tools, and downstream actions.

Q2. What layers should be used?

Answer: Input controls, model/provider safeguards, output validation, tool authorization, monitoring, human review for high-impact actions, and incident response.

Model Evaluation

Q1. What is an eval set?

Answer: A representative collection of inputs with expected or judged outcomes used to measure model/application behavior.

Q2. What metrics can be used?

Answer: Task accuracy, groundedness, relevance, refusal quality, tool-call success, latency, cost, and human preference where appropriate.

Pinecone

Q1. What is Pinecone used for?

Answer: A managed vector database/search service for semantic retrieval.

Q2. What should you store as metadata?

Answer: Identifiers, tenant/document IDs, permissions, timestamps, source type, and filterable attributes needed for retrieval and authorization.

Weaviate

Q1. What is Weaviate?

Answer: A vector database platform supporting vector search, metadata filtering, and hybrid retrieval patterns.

Q2. What is hybrid search?

Answer: Combining semantic/vector similarity with keyword or lexical matching to improve retrieval across different query types.

Qdrant

Q1. What is Qdrant?

Answer: A vector search engine/database designed for similarity search with payload metadata filtering.

Q2. What matters in production?

Answer: Index configuration, distance metric, payload filtering, collection design, backup strategy, latency, and retrieval evaluation.

Milvus

Q1. What is Milvus?

Answer: An open-source vector database designed for scalable similarity search.

Q2. How do you evaluate a vector database?

Answer: Measure recall/precision of retrieval, latency at target load, filtering performance, memory/storage needs, operational complexity, and cost.

LlamaIndex

Q1. What is LlamaIndex?

Answer: A framework for connecting LLM applications with external data through indexing, retrieval, and data connectors.

Q2. When is it useful?

Answer: When a project needs reusable ingestion/retrieval abstractions over documents and data sources.

Hugging Face Transformers

Q1. What is Transformers?

Answer: A library/ecosystem for pretrained transformer models and tooling for NLP, vision, audio, and multimodal workloads.

Q2. What is fine-tuning?

Answer: Updating model parameters on task/domain data to adapt behavior. It differs from retrieval, which leaves model weights unchanged and supplies external context.

Sentence Transformers

Q1. What are Sentence Transformers?

Answer: Models and tooling for producing embeddings of sentences, passages, and documents for semantic similarity and retrieval.

Q2. How do you build semantic search?

Answer: Embed documents and queries, index document vectors, retrieve nearest neighbors, apply metadata filters/reranking, and evaluate against labeled queries.

Pydantic AI

Q1. Why use typed AI-agent frameworks?

Answer: They can provide structured agent inputs/outputs, dependency injection, validation, and clearer application contracts.

Q2. What should remain outside the model?

Answer: Authorization, business rules, transaction integrity, secrets, and safety-critical decisions should be enforced by application code.

Conversational AI

Q1. What are the components of a conversational AI system?

Answer: UI/voice channel, session management, prompt/context layer, model, retrieval/tools, memory, safety controls, observability, and evaluation.

Q2. How do you handle conversation memory?

Answer: Separate short-term context from durable user data, store only necessary information, apply access controls, and define retention rules.

Voice AI

Q1. What makes voice AI different from text AI?

Answer: Audio introduces streaming, latency, turn-taking, interruption, noise, transcription errors, and speech synthesis concerns.

Q2. What is barge-in?

Answer: It allows the user to interrupt the assistant while it is speaking; the system must stop/cancel current audio generation and continue with the new turn.

AI Assistants

Q1. What is an AI assistant architecture?

Answer: A model plus conversation state, tools, retrieval, authorization, orchestration, observability, and a user interface.

Q2. Agent vs assistant?

Answer: An assistant may mainly respond and call controlled tools; an agent often has a more autonomous loop of planning, tool use, observation, and iteration.

Workflow Automation

Q1. How can AI improve workflow automation?

Answer: Use AI for classification, extraction, summarization, routing, and natural-language interfaces, while deterministic code enforces business rules and side effects.

Q2. How do you make automation reliable?

Answer: Use idempotency, retries, queues, validation, audit logs, dead-letter handling, and human approval for risky actions.

MCP

Q1. What is MCP?

Answer: Model Context Protocol is a protocol for connecting AI applications/models with tools and contextual resources through standardized interfaces.

Q2. What is the security concern?

Answer: Treat tools and resources as privileged integrations: authenticate, authorize, validate arguments, restrict data access, and audit actions.

AI Guardrails

Q1. What are guardrails?

Answer: Controls around an AI system that constrain inputs, outputs, tools, and workflows to acceptable behavior.

Q2. Are guardrails only prompts?

Answer: No. Strong guardrails are enforced in application code, schemas, permissions, filters, moderation, and monitoring; prompts alone are insufficient.

AI Evaluation Frameworks

Q1. What does an evaluation framework provide?

Answer: Reusable datasets, evaluators, experiments, scoring, traces, and regression tracking for AI application quality.

Q2. How do you prevent regressions?

Answer: Run a fixed evaluation suite in CI/release checks and compare important metrics before changing prompts, models, retrieval, or tools.

streaming audio

Q1. Why stream audio?

Answer: Streaming reduces perceived latency because processing and playback can start before the full response is complete.

Q2. What can go wrong?

Answer: Buffer underruns, format mismatch, packet loss, clock drift, delayed cancellation, and synchronization issues. Use clear framing, buffering, sequence handling, and metrics.

voice activity detection

Q1. What is VAD?

Answer: Voice Activity Detection identifies when speech is present in an audio stream, helping detect turn boundaries and reduce unnecessary processing.

Q2. Why is VAD important in voice AI?

Answer: It supports natural turn-taking, lower cost, and faster response by detecting when the user starts/stops speaking.

embeddings

Q1. Why can duplicate 'embeddings' skills appear?

Answer: Different job platforms may list the same concept at different abstraction levels. Treat them as one core competency: embedding generation, indexing, retrieval, and evaluation.

Q2. What is cosine similarity?

Answer: It measures the angle between vectors; higher similarity generally indicates closer direction in embedding space, assuming the embedding model and metric are appropriate.

Part 3 — Practical AI/ML & System-Design Questions

1. Design an AI document Q&A system.

Strong answer: Ingest documents, parse and clean text, chunk with overlap where appropriate, generate embeddings, store vectors plus metadata/permissions, retrieve top candidates, optionally rerank, build a grounded prompt, generate an answer, cite sources, validate output, and log retrieval/model metrics.

2. Design a real-time AI call-center assistant.

Strong answer: Capture audio, use VAD and streaming STT, maintain session state, retrieve relevant customer/knowledge context, call an LLM with controlled tools, stream TTS, support barge-in, and emit events to a shared event bus. Enforce tenant/agent permissions and record trace IDs/latency.

3. How would you handle 100 concurrent AI requests?

Strong answer: Measure first. Use multiple API workers, async I/O, bounded concurrency, queues for heavy jobs, connection pooling, caching, model/provider rate limits, backpressure, timeouts, and horizontal scaling. Load-test p50/p95/p99 latency and error rates.

4. How would you implement RAG?

Strong answer: Create an ingestion pipeline, chunk documents, embed them, store vectors and metadata, retrieve relevant chunks, optionally rerank, pass only useful context to the model, and evaluate retrieval separately from generation.

5. How would you stop an AI assistant from deleting data?

Strong answer: The model never gets unrestricted database access. Expose a narrowly scoped tool, enforce authorization in backend code, require confirmation for destructive actions, validate arguments, and audit every execution.

6. How do you debug a hallucination?

Strong answer: Trace the request, retrieved context, prompt, model response, and tool results. Determine whether retrieval was missing/wrong, context was ignored, instructions conflicted, or the model invented unsupported content. Add a regression test.

7. How would you choose between Redis and Kafka?

Strong answer: Redis is useful for low-latency caching, short-lived state, locks, and lightweight pub/sub/streams. Kafka is designed for durable, partitioned event streaming and replay at scale. The correct choice depends on durability, replay, throughput, ordering, and operational requirements.

8. How would you design multi-tenant AI data isolation?

Strong answer: Every document, vector, conversation, and tool action carries tenant identity. Enforce tenant filters server-side, use separate credentials/namespaces where appropriate, test authorization boundaries, and never trust a tenant ID supplied only by the client.

9. How do you test an AI API?

Strong answer: Unit-test deterministic code; contract-test schemas; integration-test provider/retrieval/tool paths; run evaluation datasets for quality; load-test concurrency; and security-test prompt injection, data leakage, unauthorized tools, and malformed inputs.

10. What would you say when you don't know an AI interview answer?

Strong answer: State what you know, identify the uncertainty, explain how you would verify it, and reason from first principles. Avoid inventing an answer.

Part 4 — Rapid Revision Checklist

Topic	Know these before the interview
Python	decorators, generators, async, exceptions, memory, typing, testing
FastAPI	routers, Pydantic, dependencies, auth, async, background jobs, OpenAPI
REST	HTTP methods, status codes, idempotency, pagination, versioning, auth
Async & WebSockets	event loop, blocking calls, reconnects, heartbeats, scaling
Redis	cache, TTL, locks, pub/sub, rate limiting, failure handling
LLMs	tokens, context, temperature, hallucinations, evaluation, latency/cost
Prompting	role/task/context/constraints, examples, injection resistance
RAG	chunking, embeddings, vector search, metadata filters, reranking, citations
Tools	function calling, JSON Schema, validation, auth, least privilege, audit
Voice AI	STT, VAD, TTS, streaming, barge-in, latency, audio formats
Vector DBs	ANN, cosine/distance, filtering, recall, latency, backups
Hugging Face	Transformers, inference, embeddings, fine-tuning vs RAG
Agents	tool loop, state, guardrails, deterministic business logic
Evaluation	golden dataset, correctness, groundedness, tool success, regression
Production	timeouts, retries, queues, backpressure, observability, security

Final practice rule: Do not memorize these answers word-for-word. For each question, practice a 30–60 second explanation, then add one concrete example from a project you have actually built. For system-design questions, practice drawing the architecture and explaining failure cases.