AI ENGINEER INTERVIEW CHEAT SHEET

from zero to usable genai systems

A. FROM NLP TO LLMs

1 Language models start from predicting text

Problem
How do we model language so a machine can predict what comes next?

What it is
n-gram models predict the next word from the previous (n-1) words.

How it works
Count how often words follow a context. Pick the most likely next word.

Example (bigram: n=2)

Context: "the cat" → predict next word

Next word	Count	P(next | the cat)
sat	18	0.60
is	7	0.23
on	3	0.10
and	2	0.07
...	...	...

Analogy
n-gram is like autocomplete using only the last few words.

Interview cue
"n-grams predict the next word from a short context using frequency counts."

2 From n-grams to modern language models

Problem
Short context misses meaning and long-range dependencies.

What it is
Modern LMs use more context (and better representations) to capture meaning.

How it works
Use bigger context windows → neural models (e.g., Transformers) to predict the next token.

Short context (n-gram)
Context: "bank of the"
Likely next: river / year / ... (weak) ✗

Longer context (LM)
Context: "I deposited cash at the bank on Main Street."
Likely next: yesterday / in / ... (strong) ✓

Analogy
Short memory = forgetful student.
Long memory = understands the story.

Interview cue
"Modern LMs overcome n-gram limits with larger context and neural architectures."

NLP EVOLUTION TIMELINE

n-grams (1980s–2000s) → Embeddings (2000s–2010s) → Neural / RNN / CNN (2010s) → Transformers (2017–) → LLMs (2020+)

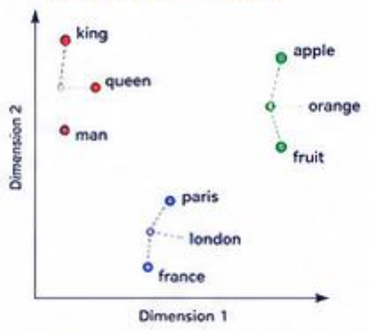
3 Embeddings / Word2Vec as meaning vectors

Problem
Words as IDs/lists do not capture meaning or similarity.

What it is
Embeddings map words to dense vectors in a continuous space.

How it works
Models learn vectors so similar words end up closer together.

Example (2D visualization)



Analogy
Vectors are like GPS coordinates for meaning.

Example math
king − man + woman ≈ queen

Interview cue
"Embeddings place semantically similar words closer together in vector space."

4 What is an LLM?

Problem
We need models that understand rich context and generate useful text at scale.

What it is
LLM = Large Language Model. Trained on massive text to predict and generate tokens.

How it works
Transformer architecture learns patterns from billions of tokens using self-attention.

LLM = Large Language Model
A neural network trained on massive text data to predict the next token given previous tokens.

Key traits
- Huge training data (web, books, code, ...)
- Billions of parameters
- Handles long-range context
- Generates, summarizes, translates, reasons

Analogy
An LLM is like a very well-read person who writes well, but does not automatically know your company's private documents or current data.

Interview cue
"An LLM predicts the next token using patterns learned from a massive dataset."

5 Tokens — pieces of text the model reads/writes

Problem
Models work on numbers, not raw text.

What it is
Tokens are the basic pieces (words, subwords, symbols) an LLM consumes/produces.

How it works
Text → tokenizer → tokens (IDs). Model operates on token IDs.

Example tokenization
Text: "AI engineers build useful systems."

Word-level

AI	engineers	build	useful	systems	.
101	8802	325	1704	2403	13

Subword (BPE)

AI	engineer	s	build	use	ful	system	s	.
101	27381	16	325	1650	478	6123	16	13

Symbols / special

<BOS>	AI	engineer	s	.	<EOS>
0	101	27381	16	13	1

Interview cue
"LLMs operate on token IDs, not raw words."

6 Prompt → generated answer

Problem
How do we get useful output from an LLM?

What it is
You provide a prompt (instructions + context). Model generates an answer token by token.

How it works
Prompt → tokens → LLM → next-token prediction → decoded text.

Pipeline: Prompt → Tokenize → LLM → Decode → Output text

Analogy
You ask a question; the model writes the best possible continuation.

Interview cue
"A prompt guides the model to generate the most relevant continuation."

7 Why the LLM is not the full product

Problem
LLMs don't automatically know your private data, policies, or tools.

What it is
An LLM alone lacks context, permissions, up-to-date data, and actions.

How it works
You must connect data, APIs, tools, and safety layers around the LLM.

LLMs can do...
- ✓ Generate text
- ✓ Summarize
- ✓ Reason on text

... but NOT automatically:
- ✗ Access private company data
- ✗ Obey your permissions & policies
- ✗ Call tools / APIs reliably
- ✗ Stay up-to-date with internal systems

Interview cue
"An LLM is a powerful brain, not a business system."

8 Hallucination and grounding

Problem
LLMs can produce plausible but wrong answers.

What it is
Hallucination = confident but false. Grounding = anchor answers to real data/sources.

How it works
Provide relevant contexts (RAG), citations, and verification to ground outputs.

Hallucination (bad)
Q: Who is the CEO of ACME Inc? (Internal data not in training)
A: John Smith is the CEO of ACME Inc. ✗
(Plausible but may be false)

Grounded (good)
Q: Who is the CEO of ACME Inc?
A: According to ACME HR doc (2024-03), the CEO is Jane Doe.
Source: HR_People_2024-03.pdf

Best practices
- ✓ Use RAG with trusted documents
- ✓ Cite sources
- ✓ Verify critical facts
- ✓ Verify critical facts

Interview cue
"Grounding reduces hallucinations by anchoring the model to trusted data and showing sources."

9 What an AI Engineer really does

Problem
A model alone ≠ value. Businesses need reliable, secure, useful systems.

What it is
AI Engineer turns a model into a usable, safe, and scalable application.

How it works
Design data flows, RAG, tools, APIs, evals, safety, monitoring, and deploy.

LLM alone	vs	Enterprise GenAI system
• Knows public data only		✓ Connected to company data (RAG)
• No access to private data		✓ Tools & APIs (search, CRM, ...)
• No tools or actions		✓ Permissions & policies enforced
• No policies / permissions		✓ Safety, guardrails, observability
• Hard to evaluate		✓ Evaluated & monitored
• Not production-ready ☹		✓ Reliable, secure, useful ☺

KEY TAKEAWAY
An AI Engineer does not just use ChatGPT. An AI Engineer turns an AI model into a usable enterprise system.

NLP EVOLUTION AT A GLANCE

n-grams (Local statistics) → Embeddings (Meaning vectors) → RNN / CNN (Sequence models) → Transformers (Attention at scale) → LLMs (General text intelligence)

MINI EXAMPLE: NEXT-WORD PREDICTION

Prompt: "The chef cooked a"

Model predicts next token:

meal (0.52)	delicious (0.21)	wonderful (0.08)	new (0.05)	...

Model picks highest probability (sampling/decoding may vary).

TOKEN FLOW (UNDER THE HOOD)

AI ENGINEER INTERVIEW CHEAT SHEET

from zero to usable genai systems

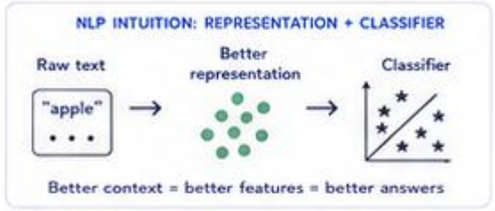
C. CONTEXT: RAG, CAG, CRAG

19 WHY CONTEXT MATTERS

Problem
LLMs can produce confident but wrong answers without the right context.

What it is
Context provides the facts and constraints the model should use to answer.

How it works
Better context → better representations inside the model → better classification / generation.



Analogy
Asking a doctor with the right chart and test results beats asking with no background.

Interview cue
LLMs don't know by default.
We must give them the right evidence.

Common mistake
Relying on model memory instead of providing reliable, up-to-date context.

20 RAG (RETRIEVAL-AUGMENTED GENERATION)

Problem
Model training data is static and incomplete; it may not contain your latest or private domain knowledge.

What it is
Retrieve relevant information at runtime, add it to the prompt, then generate a grounded answer.

How it works
Find → Filter → Attach → Generate. The model cites the retrieved facts.

Analogy
Searching the library in real time and reading the most relevant pages before answering.

Interview cue
RAG reduces hallucination and keeps answers current by grounding them in retrieved facts.

Common mistake
Retrieving too much / irrelevant context and overloading the prompt.

21 RAG PIPELINE

User asks a question.	Convert the question to a vector.	Search vector DB for relevant chunks.	Rerank for relevance and diversity; keep top-k.	Compose prompt with chunks + instructions.	LLM reads and produces the final answer.

Analogy Ask → look up → pick the best pages → give to expert → get answer.

Interview cue RAG = Retrieve at runtime, then Generate with grounded context.

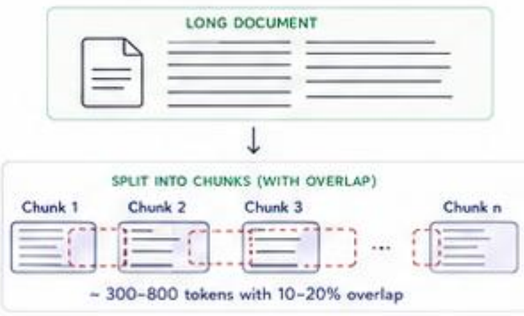
Common mistake *Ignoring reranking or using too small/large chunk sizes.*

22 CHUNKS

Problem
Long documents are too big to embed and retrieve precisely.

What it is
Documents are split into smaller, overlapping pieces (chunks) for retrieval.

How it works
Chunk → embed → store. Retrieve the chunks most relevant to the query.



Analogy
Cut a long book into pages so you can find the right pages fast.

Interview cue
Good chunking improves recall without wasting context window.

Common mistake
Chunks too big (miss precision) or too small (lose meaning).

23 EMBEDDINGS AND VECTOR DATABASE

Problem
Keyword search misses meaning; synonyms and paraphrases fail.

What it is
Embeddings convert text into dense vectors. Vector DB finds similar vectors (semantic search).

How it works
Text → embedding model → vector DB index → cosine similarity search.

VECTOR DB FEATURES
- ✓ Fast ANN search (e.g., HNSW)
- ✓ Metadata filters
- ✓ Top-k results with scores

Analogy Instead of matching words, you match meaning.

Interview cue *Embeddings + Vector DB enable relevance beyond keywords.*

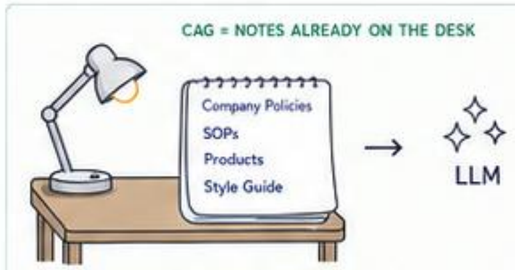
Common mistake *Mixing embedding models or not normalizing vectors.*

24 CAG (CACHE-AUGMENTED GENERATION)

Problem
Some knowledge is stable and used frequently. Retrieving it every time is wasteful and slow.

What it is
Preload stable, reusable knowledge into a local cache or prompt template so the model can use it directly.

How it works
Cache → attach directly to prompt → generate. No retrieval round-trip.



Analogy Keep your most-used notes on your desk, no need to go to library.

Interview cue *CAG reduces latency and cost for stable, high-reuse knowledge.*

Common mistake *Caching data that changes often → stale / incorrect answers.*

25 COLD DATA vs HOT DATA

COLD / STATIC DATA
- Changes rarely (days / months / years)
- Examples: policies, SOPs, manuals, product docs, company knowledge
- Best for: *CAG (cache)*
- Goal: *fast, consistent, low cost*

→ *Keep in cache*

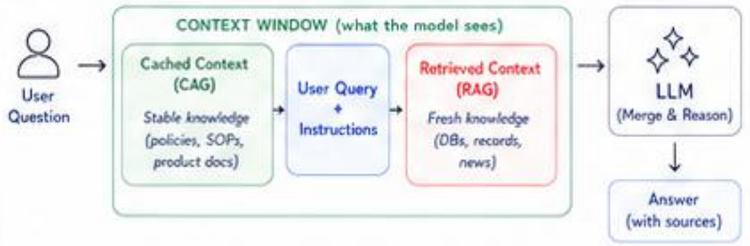
HOT / DYNAMIC DATA
- Changes frequently (seconds / minutes)
- Examples: customer records, orders, financial data, live DBs, news
- Best for: *RAG (retrieve)*
- Goal: *freshness and accuracy*

→ *Retrieve in real time*

Interview cue *Different data, different strategy. Choose based on volatility.*

Common mistake *Caching hot data or retrieving static data.*

26 CRAG = CAG + RAG (HYBRID ARCHITECTURE)



Why it works
- ✓ Cache takes care of stability and speed.
- ✓ Retrieve brings freshness and specificity.
- ✓ LLM merges both for accurate answers.

Interview cue
CRAG uses the best of both worlds: speed + freshness.

Common mistake *Overfilling the context window or giving low-quality retrieved chunks.*

27 DECISION RULE

Problem
Engineers must decide what to cache, what to retrieve, or both.

What it is
A simple rule to choose CAG, RAG, or CRAG based on data characteristics.

How it works
Assess stability, reuse frequency, and freshness requirements.

WHEN TO USE WHAT

Stable + Reused (rarely changes, used often)	→	CAG (Cache)	Keep it nearby. Preload in prompts or local store.
Dynamic + Changing (changes often, time-sensitive)	→	RAG (Retrieve)	Fetch at runtime from latest sources.
Mixed / Enterprise (both stable & dynamic)	→	CRAG (CAG + RAG)	Cache the stable part, retrieve the fresh part.

Interview cue
"Not all enterprise knowledge changes at the same speed, so I would separate what should be cached from what should be retrieved."

Common mistake
One-size-fits-all approach. It hurts latency, cost, or accuracy.

AI ENGINEER INTERVIEW CHEAT SHEET

from zero to usable genai systems

B. MAKING AI USABLE

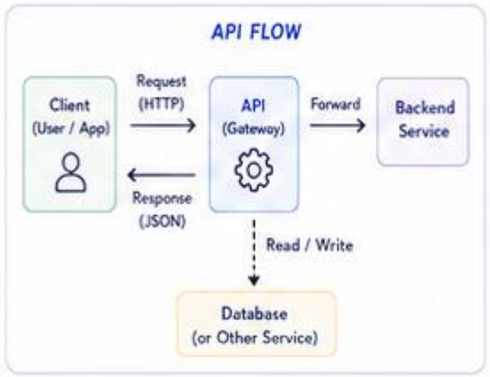
10 API

Problem
Systems can't talk safely or in a controlled way.

What it is
A controlled way for systems to talk to each other.

How it works
A client sends a request to an API. The API validates, processes, and returns a response.

Analogy
A waiter between customer and kitchen.



Interview cue
"Explain an API like I'm five."

Common mistake / warning
Exposing secrets in URLs or returning too much data.

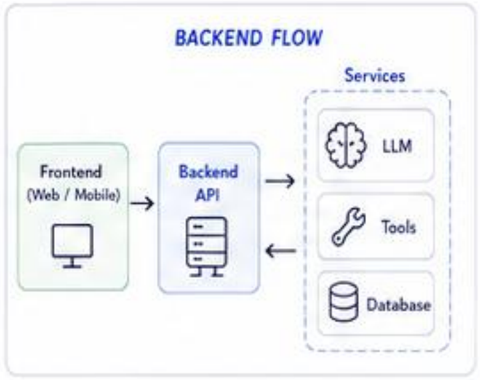
11 BACKEND API

Problem
User requests need to be handled, secured and routed.

What it is
The backend receives the user request and coordinates the system.

How it works
It authenticates, validates, routes to services / tools / DB, and returns the result.

Analogy
The control center of your app.



Interview cue
"How does the backend fit in the system?"

Common mistake / warning
Skipping validation, auth, or rate limiting.

12 TOOLS

Problem
LLM alone can't access live data or take actions.

What it is
Tools are functions the LLM can ask the system to run.

How it works
The model decides it needs a tool, asks for it, and uses the result.

Analogy
If the LLM is the brain, tools are the hands.

Interview cue
"What kinds of tools would you build?"

Common mistake / warning
Giving too many tools or unclear descriptions.

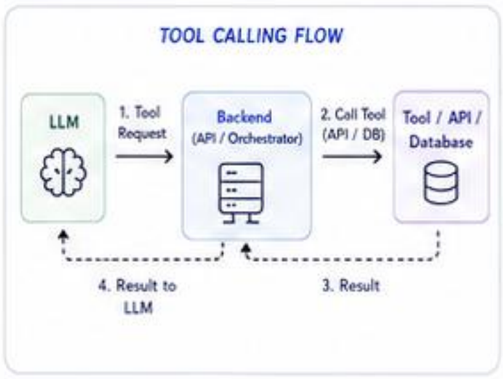
13 TOOL CALLING

Problem
Model needs info or action outside its knowledge.

What it is
The model asks to call a function or API. Backend executes it and returns the result.

How it works
LLM decides → sends tool request → backend runs it → result goes back → LLM continues.

Analogy
Asking a teammate to look something up and report back.



Interview cue
"Walk me through tool calling."

Common mistake / warning
Not validating tool outputs before sending to LLM.

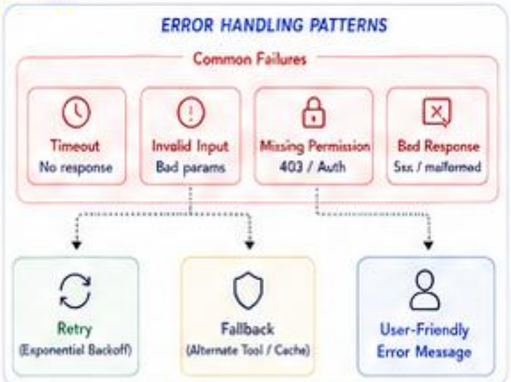
14 TOOL ERROR HANDLING

Problem
Tools can fail. The system must handle it gracefully.

What it is
Detect failures, recover if possible, and give useful messages to users.

How it works
Detect error → decide: retry, fallback, or show user an (helpful) error.

Analogy
Like a smart auto-retry with a backup plan.



Interview cue
"How do you handle tool failures?"

Common mistake / warning
Silent failures or empty responses to the user.

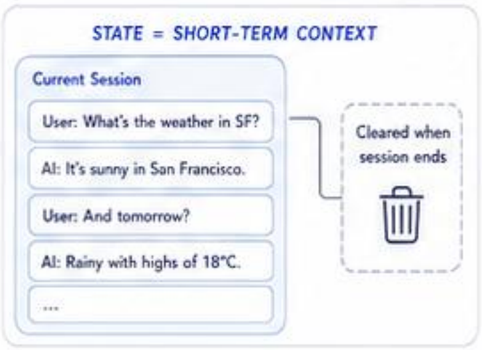
15 STATE

Problem
How do we keep track of what's happening right now?

What it is
Temporary information about the current conversation or workflow.

How it works
Stored in memory / session. Cleared when the session ends.

Analogy
Sticky notes on a whiteboard in a meeting.



Interview cue
"What's the difference between state and memory?"

Common mistake / warning
Storing long-term info in state (it gets lost).

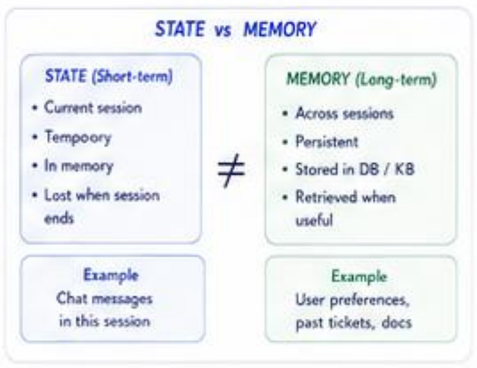
16 MEMORY

Problem
We need to remember important info across sessions.

What it is
Persistent information the system can reuse later.

How it works
Stored in a DB / vector store / knowledge base and retrieved when needed.

Analogy
A notebook you keep for yourself.



Interview cue
"When would you use memory instead of state?"

Common mistake / warning
Storing sensitive data forever without retention policies.

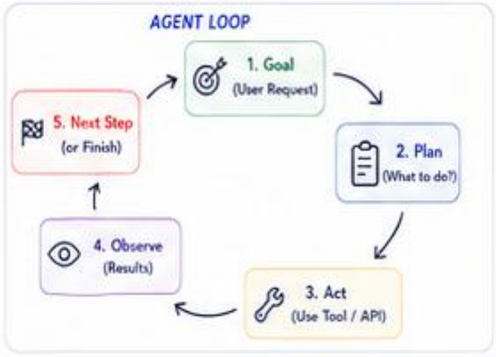
17 AGENT

Problem
Complex tasks need multiple steps and decisions.

What it is
An agent plans, uses tools, observes results, and decides the next step.

How it works
Loop until the goal is reached or stopped.

Analogy
A project manager that gets things done.



Interview cue
"How does an agent decide the next step?"

Do not use agents when a simple API call or simple RAG pipeline is enough.

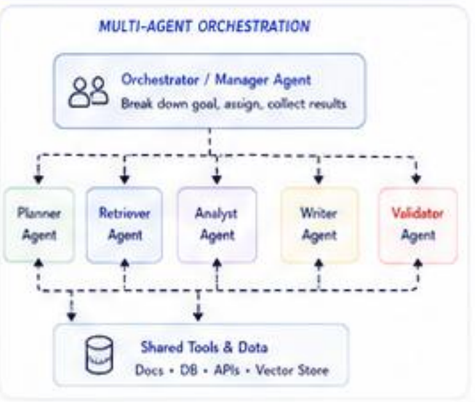
18 MULTI-AGENT SYSTEMS

Problem
One agent can't be best at everything.

What it is
Several specialized agents coordinate to complete a complex task.

How it works
A manager/orchestrator assigns tasks, collects results, and orchestrates the flow.

Analogy
A team where each person has a role.



Interview cue
"When would you use multiple agents?"

Common mistake / warning
Too much coordination cost for simple tasks.

AI ENGINEER INTERVIEW CHEAT SHEET

from zero to usable genai systems

D. PRODUCTION + INTERVIEW ANSWERS

28 Guardrails

Problem
Models can hallucinate, output unsafe content, or go off track.

What it is
Safety and quality controls around the model.

How it works
Validate inputs, constrain behavior, check outputs, block or fix risky content.

Analogy
Road guardrails keep the car on the road and prevent it from falling off a cliff.

Interview cue
What guardrails would you add for this use case?

Common mistake
Relying only on the base model without checks.

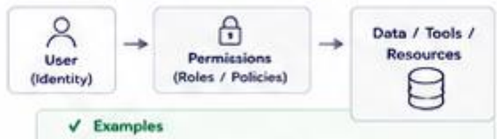
29 Security / permissions

Problem
The system might access data the user should not see.

What it is
The system enforces access only to what the user is allowed to see.

How it works
AuthN (who you are) + Auth2 (what you can do). Row/field-level filters, tenant isolation, encryption.

Analogy
Like office keycards: you can only enter rooms you have permission to access.



✓ **Examples**
- ✓ User A can see only their documents.
- ✓ Finance role can access budgets.
- ✓ Tenant A cannot see Tenant B's data.

Interview cue
How do you make sure users only see their own data?

Common mistake
Hard-coding access rules instead of using policies.

30 Logs

Problem
When something goes wrong, we don't know what happened.

What it is
Logs record what happened: prompt, retrieval, tool call, error, answer, latency, cost.

How it works
Log every important step with timestamps and IDs. Store in a searchable place.

Analogy
Airplane black box: records the flight so we can understand what happened.

Log fields
timestamp, user_id, session_id, model, prompt, retrieval_ids, tool_calls, output, error, latency_ms, cost_usd

Interview cue
What would you log for this system?

Common mistake
Logging too little (can't debug) or too much (too noisy/costly).

31 Observability

Problem
Hard to understand why the system behaved a certain way or where it failed.

What it is
The ability to understand what the system did and why.

How it works
Logs + metrics + traces + dashboards + alerts.

Analogy
Car dashboard: shows speed, fuel, engine status so you know what's happening.

Key metrics
- ✓ Answer quality
- ✓ Retrieval quality
- ✓ Latency (p50/p95/p99)
- ✓ Error rate
- ✓ Cost per request
- ✓ User satisfaction

Interview cue
How would you debug a bad answer in production?

Common mistake
Building without visibility; debugging becomes guesswork.

32 Evaluation

Problem
How do we know if the system is good?

What it is
Measure answer quality, grounding, retrieval quality, safety, latency, and cost.

How it works
Use test sets and automated metrics + human review.

Analogy
Like grading a student with tests and a rubric.

Dimension	Example metrics (NLP roots)
Answer quality	Accuracy, F1, Exact Match, LLM-as-Judge score
Grounding	Context precision/recall, Attribution score
Retrieval quality	Recall@k, MRR, NDCG
Safety	Toxicity, PII leak, Policy violations
Latency	p50 / p95 / p99 latency
Cost	Cost per 1K tokens / request

Interview cue
What metrics would you track for this use case?

Common mistake
Only checking vibe; no real metrics or test set.

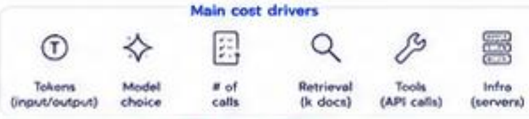
33 Cost

Problem
Costs can explode if we don't control usage.

What it is
LLM cost depends on tokens, model choice, number of calls, retrieval, tools, and infrastructure.

How it works
Estimate, measure, and optimize cost continuously.

Analogy
Like cloud bills: depends on usage, instance type, and how often you run it.



How to reduce cost
- ✓ Use smaller/cheaper models where possible.
- ✓ Cache results and reuse answers.
- ✓ Limit context length and retrieved docs.
- ✓ Batch requests and avoid extra tool calls.

Interview cue
How would you optimize cost without hurting quality?

Common mistake
Optimizing only latency and ignoring cost.

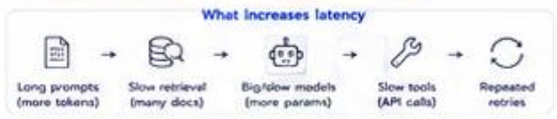
34 Latency

Problem
Users get frustrated when they wait too long.

What it is
Latency = how long the user waits for an answer.

How it works
Many factors add up: prompt size, retrieval, model speed, tools, infra.

Analogy
Like a restaurant: long menu, busy kitchen, or slow delivery makes you wait.



How to reduce latency
- ✓ Shorten prompts and context.
- ✓ Use fast retrieval and fewer docs.
- ✓ Choose faster models / small models.
- ✓ Parallelize tool calls and use caching.

Interview cue
What would you do if latency is too high?

Common mistake
Throwing bigger models at the problem.

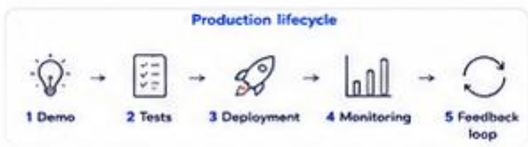
35 Deployment

Problem
A system is useless if it's not in production.

What it is
Deployment = putting the system into production.

deploy = put into production
deployment = release / production setup

How it works
Follow a safe release process and monitor.



✓ **Good practices**
- ✓ Feature flags and canary releases.
- ✓ Rollback plan if something breaks.
- ✓ Health checks and alerts.
- ✓ Monitor, learn, and improve.

Interview cue
How would you deploy and monitor this system?

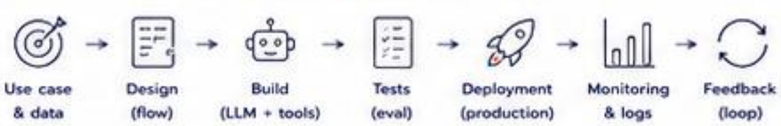
Common mistake
Shipping the first demo straight to production.

36 Final interview answer

For me, an enterprise GenAI system is not just calling an LLM.

It is designing a flow with context, tools, permissions, guardrails, evaluation, observability, and cost/latency control so the system is useful and safe in production.

Production pipeline (end-to-end)



Safety & quality flow (summary)

Risk → Mitigation

Risk	Mitigation
Hallucinations	RAG + grounding + eval
Data leaks / PII	Permissions + guardrails + logs
Toxic / unsafe output	Safety filters + policies
High cost	Optimize tokens + model + cache
High latency	Optimize retrieval + model + cache
Low quality answers	Better context + eval + feedback

If I get stuck, say this

I would start simple:
- define the use case,
- connect reliable context,
- add guardrails,
- measure quality,
- and only then scale with agents or more complex architecture.