Assignment 15 — Solutions

LLMOps, Deployment & Safety

Part C — Building GenAI Applications · 15 questions with answers

Q1. How do you deploy an LLM application to production? Describe the architecture. ★ · Asked: Google

Answer

Typical production architecture:

- **Frontend** – web/mobile chat UI or integration (Slack, WhatsApp, internal portal).
- **API layer** – FastAPI/Node service behind an API gateway: authentication, rate limiting, request validation, streaming responses (SSE/WebSockets).
- **Orchestration** – LangChain/LangGraph application logic: prompt templates, RAG chain or agent, tools.
- **Guardrails** – input checks (PII, prompt injection, topic filters) and output checks (toxicity, PII leakage, format).
- **Model layer** – LLM API (OpenAI/Azure OpenAI/Bedrock/Vertex) or self-hosted models on GPUs (vLLM/TGI on Kubernetes), via a **model gateway** (LiteLLM) for routing, fallbacks and retries.
- **Data layer** – vector DB, document ingestion pipeline (scheduled re-indexing), cache (Redis), conversation store (Postgres).
- **Observability** – tracing (LangSmith/Langfuse), logs, metrics (latency, tokens, cost), evaluation and feedback.
- **CI/CD** – Docker containers, automated eval tests on prompt/model changes, secrets management, versioned prompts/configs.

Start simple (one service + managed APIs) and scale components as needed.

Q2. How do you reduce latency and cost (caching, smaller models, streaming, batching, prompt compression, routing)? ★ · Asked: TCS, Google

Answer

- **Right-size the model:** use smaller/cheaper models for simple tasks; **route** only hard queries to large models (model cascade/router).
- **Caching:** exact-match and **semantic caching** of responses; **prompt caching** offered by providers for repeated long prefixes (system prompt, documents) – big savings on cost and time-to-first-token.
- **Streaming** – show tokens as they arrive (perceived latency drops even if total time is the same).
- **Reduce tokens:** shorter prompts, fewer/better retrieved chunks (re-ranking), summarize history, set max_tokens, ask for concise output.
- **Batching** – batch offline jobs (Batch APIs are ~50% cheaper); continuous batching for self-hosted models.
- **Parallelize** independent calls (retrieval + classification at the same time).
- **Self-hosting optimizations:** quantization, vLLM, speculative decoding, GPU right-sizing.
- Fine-tune a small model to replace a large one for high-volume narrow tasks.
- Monitor cost per request/feature and set budgets/alerts.

Q3. What are vLLM, TGI and Ollama? What is continuous batching and PagedAttention?

Answer

- **vLLM** – open-source, high-throughput LLM **inference/serving engine** (from UC Berkeley). Uses PagedAttention and continuous batching, exposes an **OpenAI-compatible API**, supports quantization, tensor parallelism, LoRA adapters. Production choice for self-hosted serving.
- **TGI (Text Generation Inference)** – Hugging Face's production serving toolkit (Rust/Python) with continuous batching, streaming, quantization, tensor parallelism.
- **Ollama** – easy tool to **run LLMs locally** (laptop/desktop/server) with one command (`ollama run llama3.1`); uses llama.cpp and GGUF quantized models; great for development and small-scale/private use, not for high-throughput serving.
- **Continuous (in-flight) batching** – instead of waiting for a whole batch to finish (static batching), the server **adds new requests to the batch and removes finished ones at every generation step**, so the GPU stays fully utilized → several times higher throughput.
- **PagedAttention** – stores the KV cache in **fixed-size blocks ("pages")** that need not be contiguous, like virtual memory in an operating system. Removes memory fragmentation and over-reservation, allows sharing of common prefixes between requests → many more concurrent sequences per GPU.

Q4. What is Groq / LPU inference and why is its latency so low?

Answer

Groq is a company that built a custom AI inference chip called the **LPU (Language Processing Unit)** and offers a cloud API (GroqCloud) running open models (Llama, Mixtral, Qwen, Whisper) at **very high speed** (hundreds to over a thousand tokens/second).

Why latency is so low:

- LLM token generation is **memory-bandwidth bound** – a GPU must repeatedly read the weights from external HBM. Groq's LPU keeps model weights in large amounts of **on-chip SRAM**, which is far faster than HBM.
- **Deterministic, compiler-scheduled architecture** – the compiler plans every operation and data movement in advance, so there is no runtime scheduling, caches or contention overhead.
- Many chips are linked with a fast, synchronous interconnect so a model is spread across chips and runs as a pipeline.

Trade-off: needs many chips per model (limited SRAM per chip), supports a selected set of models.

Q5. What is semantic caching? · Asked: Google, Wipro

Answer

Semantic caching stores previous queries and their responses with **embeddings**. For a new query, compute its embedding and search the cache; if a stored query is **semantically similar above a threshold** (e.g. cosine > 0.95), return the cached answer instead of calling the LLM.

Example: "How do I reset my password?" and "Steps to change my login password" → same cached answer.

Benefits: lower cost and latency for repetitive questions (FAQs, support).

Risks: wrong answers if the threshold is too low (questions that look similar but differ – "credit card limit" vs "debit card limit"); stale answers (set TTL/invalidate when data changes); personalization and access control (don't share user-specific answers across users).

Tools: GPTCache, Redis vector cache, LangChain `RedisSemanticCache`.

Q6. What are guardrails? How do you handle PII, toxic content and jailbreaks? ★ · Asked: Google

Answer

Guardrails are checks and controls around the LLM that enforce safety, security, compliance and quality rules on **inputs and outputs**.

- **PII:** detect and **mask/redact** personal data (names, phone numbers, Aadhaar/PAN, card numbers) before sending to the LLM and before logging (Microsoft **Presidio**, regex, NER); prevent PII leakage in outputs; data retention policies.
- **Toxic/harmful content:** moderation models/APIs (OpenAI Moderation, **Llama Guard**, Azure Content Safety) on input and output.
- **Jailbreaks & prompt injection:** classifiers (e.g. Prompt Guard), strong system prompts, separating instructions from data, least-privilege tools, human approval for sensitive actions.
- **Topic and policy rails:** keep the bot on allowed topics (NeMo Guardrails), refuse financial/medical advice if required.
- **Output validation:** JSON schema validation, grounding/faithfulness checks, citation requirements (Guardrails AI library).
- Logging, rate limiting and red-team testing.

Q7. How do you monitor an LLM app in production (latency, tokens, cost, feedback, drift)? · Asked: TCS, Infosys, Google

Answer

- **Operational metrics:** request volume, latency (p50/p95/p99, time-to-first-token), error rates, timeouts, rate-limit hits, GPU utilization (if self-hosted).
- **Usage and cost:** input/output tokens per request, cost per user/feature/day, cache hit rate.
- **Quality:** user feedback (thumbs up/down), LLM-judge scores on sampled traffic, faithfulness/hallucination rate, refusal rate, escalation to humans, task success.
- **Retrieval health:** empty or low-score retrievals, top sources used.
- **Drift:** changes in query topics/distribution (cluster query embeddings), new unanswered question types, document freshness, provider model version changes.
- **Safety:** guardrail triggers, prompt-injection attempts, PII detections.
- Tools: LangSmith, Langfuse, Arize Phoenix, Helicone, Datadog/Grafana + OpenTelemetry; dashboards with alerts; feed failures back into the test set.

Q8. What is Gradio / Streamlit used for? Build a quick demo UI for your RAG app.

Answer

Gradio and **Streamlit** are Python libraries for building **quick web UIs/demos** for ML and LLM apps without frontend code. Gradio is especially simple for model demos (and Hugging Face Spaces); Streamlit is better for data apps/dashboards.

```
# pip install gradio
import gradio as gr

def answer(message, history):
    result = rag_chain.invoke(message)          # the RAG chain built earlier
    sources = ", ".join({d.metadata.get("source", "?") for d in result["docs"]})
    return f"{result['answer']}\n\nSources: {sources}"

demo = gr.ChatInterface(fn=answer, title="Company Policy Assistant",
                        examples=["What is the leave policy?", "How do I claim travel expenses?"])
demo.launch(share=True)          # share=True gives a temporary public link

# Streamlit equivalent (app.py; run: streamlit run app.py)
```

```
# import streamlit as st
# q = st.chat_input("Ask a question")
# if q: st.chat_message("assistant").write(rag_chain.invoke(q)["answer"])
```

Q9. Which speech-to-text and text-to-speech models have you used?

Answer

Answer from your own experience. A good structure: **which model, why, what result.**

Speech-to-text (ASR): OpenAI **Whisper** (open-source; tiny → large-v3; multilingual incl. Hindi/Marathi; faster-whisper / whisper.cpp for speed), OpenAI gpt-4o-transcribe API, Google Speech-to-Text, Azure AI Speech, AWS Transcribe, Deepgram, AssemblyAI, AI4Bharat IndicConformer (Indian languages).

Text-to-speech (TTS): OpenAI TTS, ElevenLabs, Azure Neural TTS, Google TTS, Amazon Polly; open-source Coqui XTTS, Piper, Bark, Kokoro, AI4Bharat Indic-TTS.

Example answer: "I used Whisper-small fine-tuned on call-centre audio to transcribe customer calls, measured with WER (reduced from 28% to 15%), then summarized the transcripts with an LLM." Mention evaluation (WER), latency and handling of accents/noise.

Q10. What is the difference between a POC and a production-grade GenAI architecture? ★ · Asked: PwC

Answer

POC: proves feasibility fast – a notebook/Streamlit app, one LLM API, local Chroma/FAISS, sample documents, manual testing, no auth, hard-coded prompts.

Production adds:

- **Scalable ingestion pipeline** (incremental sync, parsing, OCR, retries, idempotency) and a managed/scalable vector DB.
- **API backend** (FastAPI), auth/SSO, **RBAC on documents**, rate limiting, multi-tenancy.
- **Reliability:** timeouts, retries with backoff, fallbacks between models/providers, caching, queues for async work.
- **Security & compliance:** guardrails, PII redaction, prompt-injection defences, secrets management, audit logs, data residency.
- **Evaluation & LLMOps:** golden dataset, CI regression tests, prompt/model versioning, tracing, monitoring of cost/latency/quality, feedback loop.
- **Infrastructure:** containers, Kubernetes/serverless, autoscaling, CI/CD, IaC, cost controls.

Interview tip: the model code is ~10% of a production GenAI system.

Q11. How would you host an LLM application to keep latency low during high traffic spikes? ★ · Asked: TCS, Google

Answer

- **Stateless API layer** (FastAPI in containers) behind a load balancer with **autoscaling** (Kubernetes HPA / Cloud Run) on concurrency/latency.
- **Self-hosted models: vLLM/TGI** with continuous batching and PagedAttention, quantized models, GPU autoscaling with warm minimum replicas (cold GPU starts are slow), KV-cache/prefix caching.
- **API models:** spread load across regions/deployments and providers; request higher rate limits; **fallback model** on 429/timeouts.
- **Queue + async processing** for non-interactive work; back-pressure and rate limiting per user to protect the system.

- **Caching:** semantic cache and prompt caching for frequent queries.
- **Streaming** responses for perceived latency; smaller models/routing for easy queries.
- Load testing (Locust/k6) and monitoring p95/p99 latency, queue depth and error rates.

Q12. How do you handle flaky LLM APIs – timeouts, rate limits (429) and server errors? ★ · Asked: Wipro, Google

Answer

- **Timeouts** on every call (connect + read) so requests don't hang.
- **Retries only for transient errors** (timeouts, 429, 500/502/503) with **exponential backoff + jitter** and a max number of attempts; respect the `Retry-After` header; never blindly retry non-idempotent actions.
- **Fallbacks:** secondary deployment/region or alternative model/provider (LangChain `.with_fallbacks()`, LiteLLM router).
- **Circuit breaker:** after repeated failures, stop calling the failing service for a cool-down period and fail fast/fall back.
- **Rate limiting & queuing** on your side to stay under provider quotas; token budgeting.
- **Caching** to avoid repeated calls; **graceful degradation** message to users.
- Monitor error rates and alert. (See the backoff code in Assignment 18.)

Q13. What is prompt caching, and how is it different from semantic caching? How do you invalidate a cache? ★ · Asked: Google, Wipro

Answer

- **Prompt caching (provider-side):** the LLM provider caches the processed **prefix** of a prompt (long system prompt, tool definitions, large documents). Later requests with the **same prefix** reuse it → lower input cost (often 50–90% cheaper on cached tokens) and faster time-to-first-token. The **answer is still generated fresh**. Put static content first, variable content last.
- **Semantic caching (application-side):** you store **question → final answer** pairs with embeddings; a new question that is semantically similar enough returns the **stored answer without calling the LLM**.
- **Exact-match caching:** hash of the exact prompt/parameters → stored response.

Invalidation: TTL/expiry; invalidate entries when the underlying documents or prompt/model version change (include the doc version and prompt version in the cache key); don't cache personalized or sensitive answers across users; similarity threshold tuning to avoid wrong hits.

Q14. What is the OWASP Top 10 for LLM applications? Explain excessive agency and data exfiltration. ★ · Asked: Google

Answer

OWASP Top 10 for LLM Applications is a list of the most critical security risks in LLM apps, including: **prompt injection**, **sensitive information disclosure**, supply-chain risks, data and model poisoning, **improper output handling** (e.g. executing model output as code/SQL), **excessive agency**, system-prompt leakage, vector/embedding weaknesses, misinformation (hallucination), and unbounded consumption (cost/DoS).

- **Excessive agency:** the LLM/agent has **more functionality, permissions or autonomy than needed** (e.g. a support bot with delete-account and database-write access, acting without approval). Fix: least privilege, minimal tools, scoped credentials, human approval for high-impact actions.

- **Data exfiltration:** sensitive data leaves the system through the AI – e.g. an injected instruction makes the agent send confidential data to an external URL/email, or render a markdown image link with data in the URL. Fix: restrict outbound tools/domains, sanitize outputs, PII/DLP filters, treat retrieved content as untrusted, monitoring.

Q15. How do you deploy a GenAI change safely, and how do you roll back a bad AI deployment? · Asked: Google

Answer

- **Version everything:** prompts, model versions, retriever/chunking configs, embedding model, code – as configuration in Git.
- **Pre-deployment:** offline evaluation and regression tests in CI; security tests.
- **Shadow testing** on live traffic, then **canary release** (e.g. 5% of users) with feature flags, watching quality (feedback, judge scores), latency, cost and error metrics.
- **Gradual rollout** if healthy.
- **Rollback:** switch the feature flag/config back to the previous prompt/model version or redeploy the previous container image (blue-green deployment); keep the previous vector index version available when changing embeddings/chunking (re-indexing is not instant).
- Post-incident: add the failing cases to the golden dataset.