Spaced Retrieval Cards: Day 7 Deployment and MLOps

Cut along the card borders. Cover the amber answer section, recall your answer, then check.

Section A: Serialization and APIs

A1. Why serialize a trained model with `joblib`?

The credit-scoring MLP is a Python object full of fitted weights. The notebook closes; the object is gone.

So you can **train once and load the frozen model forever after**. `joblib.dump` writes it to disk; `joblib.load` reads it back, on any machine, into the same object. Re-training on every API restart wastes compute and is not reproducible unless every seed and library version is pinned.

A2. What must travel with the model when you serialize it?

The API gets one applicant at a time, as JSON, with no scaler attached. What does the model need to predict correctly?

The fitted preprocessing (the `StandardScaler`, or bundle model and scaler in one file), **the feature list and order** (income, debt_ratio, credit_score, employment_years, loan_amount), and **version metadata**. A model without its preprocessing and provenance is not auditable.

A3. What is the trust caveat when loading a `.joblib` file?

A teammate sends you a model file from the internet. You call `joblib.load`. What is the risk?

Deserializing a pickled object **can execute arbitrary code**. A malicious file can run anything on your machine. Treat a model artifact like executable code: load only files your own training pipeline produced, ideally with a known hash.

A4. What is a REST API, and which HTTP method serves a prediction?

A loan officer wants a decision back from the model. How does software ask for one?

A REST API is a **contract over HTTP**: send a request shaped like this, get a response shaped like that. A prediction carries input features in its body, so it is a **POST** (GET has no body and suits a health check). Data travels as **JSON** both ways.

A5. In FastAPI, what does Pydantic do, and what sta

A request arrives with `income = -5000` and `credit_score = 9000`. What happens?

Pydantic declares the request **shape and bounds**; FastAPI rejects a violating request with **HTTP 422** *before* your endpoint runs. The model only ever sees physically possible applicants. Also: load the model **once at module scope**, never per request (loading can cost hundreds of ms).

Section B: Streamlit and RAG

B1. What does Streamlit turn a Python script i

A risk officer does not write HTTP requests. They want a screen with sliders and charts.

An **interactive web app**: every `st.slider` becomes a control, every `st.pyplot` a live chart, every `st.dataframe` a sortable table. The mental model is "a script that re-runs on every interaction". No HTML, no JavaScript, no callbacks to write.

B2. Why does a Streamlit app feel sluggish witho

The user drags a dropdown; the whole script re-runs, including `pd.read_csv` and `KMeans().fit`.

Streamlit re-runs the **entire script** on every interaction, so without caching it re-loads data and re-fits models that have not changed. `@st.cache_data` says "run this once, remember the result, reuse it unless the inputs change". Then only the cheap re-draw happens on each click.

B3. What is RAG, and which half can you trust?

Ask a bare language model "what was Acme's Q3 revenue growth?" and it produces a confident, possibly invented number.

RAG = **Retrieval-Augmented Generation**: retrieve the relevant text, stuff it into the prompt, then generate from that text. **Retrieval** is the trusted half (cosine similarity over embeddings, no hallucination possible). **Generation** is the half that needs grounding (a model writes the answer, but only from the retrieved chunks).

B4. Why retrieve top-k chunks instead of stuffing all c

Tempting: dump every earnings-call passage into the prompt and let the model sort it out.

Two failures. **The context window is finite**: many documents do not fit, so the prompt is truncated, often dropping the relevant part. **Relevance gets diluted**: burying the few relevant passages among hundreds of irrelevant ones makes the model's job harder. Top-k (k around 3 to 5) keeps the prompt small and on-topic. Start at 3; tune by inspection.

B5. Why must a RAG answer carry a citation?

The model says "revenue grew 12 percent". An analyst is about to act on it.

So the user can **verify against the real source text** instead of trusting an unverifiable claim. The retrieval step already knows which chunk it returned, so carry that identifier into the displayed answer. A RAG answer with no citation is back to the hallucination problem with extra steps.

Section C: Containers, Cloud, and MLOps

C1. What is a container, and what problem doe

Your API runs fine on your laptop. A colleague runs it; it breaks. Different Python version, a missing library.

A container **bundles your code, its dependencies, and a minimal OS into one image**. Run that image anywhere and you get exactly the environment you built. "Works on my machine" becomes true everywhere, because the machine ships with the code.

C2. What is a `Dockerfile` versus a Docker image?

You want the bank's platform team to run your service without asking which Python version you used.

A `Dockerfile` is the **recipe**: start from this base, copy in this code, install these packages, run this command. `docker build` bakes the recipe into an **image**; `docker run` starts a container from it. The same image runs identically on your laptop, a teammate's, or a cloud server.

C3. Which free-tier host fits which workload?

You need to put a class project online with no credit card and (for a private course repo) no public GitHub repo.

Hugging Face Spaces: drag `app.py` and `requirements.txt` into the browser, get a live URL. Best for dashboards and chatbots. **Render / Railway**: free web services, good for FastAPI endpoints (the app sleeps when idle). **Streamlit Community Cloud**: needs a public GitHub repo. All are demo-grade, not production traffic.

C4. Data drift versus concept drift: what is the differe

A credit model trained on 2015 to 2019 applicants is quietly mis-ranking applicants in 2021. Why?

Data drift: the *inputs* change distribution (a new product brings a different customer mix). The model's mapping is still fine; it is just seeing a new population. Detect by tracking input feature distributions. **Concept drift**: the *relationship* between inputs and outcome changes (the same score means something different post-COVID). The model is now wrong. Detect by tracking accuracy against actual outcomes. Concept drift is the emergency.

C5. What do you monitor on a live model, and why lo

The API still returns probabilities. Nothing crashed. How would you know the model has gone stale?

Watch **input distributions**, **prediction distributions**, **latency and error rates**, and **accuracy against actuals** once outcomes arrive. The enabling habit is **logging every prediction with its inputs and a timestamp**: then you can compute all of the above retrospectively. Without that log you are blind. When drift appears, re-train, pass a CI/CD gate that checks metrics did not regress, promote, and keep a one-command rollback.