# Generative AI Engineer – Home Assignment

## 1. Goal

Build an application that:

- Loads a dataset of academic abstracts (.jsonl).
- Indexes them locally in any reasonable way you choose.
- Answers queries using retrieved context + a local LLM.
- Runs in a browser at `http://127.0.0.1:8080`.
- Is delivered as a single Docker image.
- Works fully on-prem for indexing and text generation.

## 2. Dataset

We will provide a `.jsonl` file (e.g., `arxiv_2.9k.jsonl`) containing ~2,900 abstracts.

Do not assume the dataset can fit fully into memory.

**Example record:**

```
{
  "id": "2509.01234",
  "title": "A New Approach to Transformers",
  "authors": "John Doe, Jane Smith",
  "categories": "cs.CL",
  "abstract": "We propose a new method for training transformer
architectures..."
}
```

- Use **abstract** as the text for embeddings. Use **id + title** for citations. Abundoned the rest.

## 3. Requirements

### A) Indexing

- On startup:
  - Read the dataset from `DATA_PATH`.
  - Build or load a local vector index.
- If a new dataset file is provided, the system must discard the old one and rebuild the index.

### B) Retrieval + Generation

- When a query is submitted, the system should:
  - Understand the query.
  - Retrieve relevant entries from the dataset.
  - Produce a **coherent, natural-language answer grounded in the retrieved context**.
  - Include citations (doc id + title) and retrieved context in the response.

**Example JSON output:**

```
{
  "answer": "...",
  "citations": [{"doc_id": "2509.01234", "title": "A New Approach to
Transformers"}],
```

```
  "retrieved_context": ["..."]
}
```

- You are free to choose any **architecture, pipeline, or models** to achieve this, as long as it runs fully on-prem inside the Docker container.

### C) Web UI + API

- A simple web app with:
  - Input box for queries.
  - Display of:
    - Final answer
    - Citations
    - Retrieved context
- Bonus: provide API endpoints (`/answer`, `/stream`).

### D) Docker

- Everything must run inside a **single container**.
- The core pipeline (indexing, retrieval, LLM) must be fully offline/local.

## 4. Bonus – Image Generation

- Optionally, add the ability to generate images for answers using an **external API** (your choice).
- The UI should allow the user to **choose whether to generate images** (e.g., a checkbox/toggle).
- Images should be included in the response JSON and displayed if requested.
- This feature is not mandatory and must not slow down the main text-answering flow.

## 5. How We Run It

We will run your app as follows:

```
docker run --rm -p 8080:8080 \
  -e DATA_PATH=/data/arxiv_5k.jsonl \
  -v $(pwd)/arxiv_2.9k.jsonl:/data/arxiv_2.9k.jsonl:ro \
  yourname/genai-app:latest
```

The app must then be accessible at:
`http://127.0.0.1:8080`

## 6. Deliverables

- Dockerfile + code (indexing, retrieval, LLM, web app).
- Published image or buildable repo.
- README.md with instructions (including how to configure optional image generation).

## 7. Resources & Constraints

- The Docker image may be **a few gigabytes** in size (models included).
- A **GPU may be available**, but the solution must also run on **CPU**.

- Minimum RAM: **4 GB** (small local models).
- Recommended RAM: **8 GB** (for larger models and smoother operation).

# 8. Evaluation

We will check that:

- The system works with any `.jsonl` dataset we mount.
- Answers are **grounded in retrieved context and phrased as coherent text**.
- Citations include **doc id + title**.
- UI/API are usable and structured correctly.
- The app runs with a single Docker command.
- **Bonus:** Optional image generation works when enabled in the UI.

# 9. Example of the GUI (keep it simple) and the interaction