Project 6 – "Ask Dr Bot": Building a Healthcare FAQ Retrieval Engine

1. Business scenario

Hospitals get thousands of repeated "What should I do if …?" questions through patient-portal chat. A retrieval engine that surfaces an existing answer (or triages to a nurse when no good match exists) can cut response time dramatically.

2. Data options (pick one)

MedQuAD (NIH) – 47 k question–answer pairs across 12 medical sites.

HealthTap-FAQ mini-dump on Kaggle (~20 k Q-A).

If those are hard to access, **Quora Question Pairs** can substitute (non-domain but shows duplicates).

3. Suggested steps (feel free to modify/choose another approach)

- **Build** Static (Word2Vec) and contextual (spaCy en_core_sci_lg or en_core_web_trf) embeddings in a semantic-search stack.
- **Measure** retrieval quality with industry metrics (Recall@k, MAP).
- **Optimize** Explore ways of improving the metrics by means such as
  - FAISS IndexFlatIP works fine for cosine if you L2-normalise the vectors.
  - TF-IDF weighting often boosts Word2Vec averages by 2-3 % Recall.
- **Deploy (optional)** a minimal API or Streamlit front-end that takes a user question and returns the most relevant FAQ answer.
  Command line is ok

4. Key Steps

a) **Data wrangling**

   a. Clean HTML, lower-case, remove PHI tokens if present.

   b. Split into train (70 %) / validation (15 %) / test (15 %). Make sure questions from the same answer cluster stay in the same split.

b) **Static-embedding baseline**

   a. Load pre-trained GoogleNews Word2Vec (or GloVe) vectors.

   b. Represent each question as the *weighted average* of its word vectors (TF-IDF weighting or plain mean).

   c. Build a FAISS index; implement cosine-similarity search.

c) **Contextual-embedding model**

   a. Use spaCy's transformer pipeline (en_core_web_trf or en_core_sci_lg).

   b. Extract the pooled sentence vector (.vector attribute).

   c. Re-index with FAISS; keep identical search logic.

d) **Retrieval evaluation**

   a. For every test question, treat its ground-truth answer as the "positive".

   b. Report **Recall@1, Recall@5, and MAP@10** for both embedding types.

   c. Plot a bar chart and provide a short error-analysis paragraph (where static failed but contextual worked, and vice versa).

e) **Mini-application**

f) Build **one** of:

   a. a **Streamlit app** with a text box, or

   b. a **FastAPI endpoint** (/ask) returning the top-3 answers as JSON.

   c. A simple command line prompt where user can ask question and get a response

5. Stretch goals

- Add **sentence-BERT** fine-tuning on your train split and compare performance.
- Implement **RAG-style summarisation**: after retrieving top-k answers, feed them plus the user's question into GPT-3.5 Turbo (or Llama 3) to generate a one-paragraph personalised response.
- Push the app to **Render, HuggingFace Spaces or Fly.io** with a one-click deploy script.

6. Deliverables

1. **Notebook / script** with: data prep, both embedding pipelines, evaluation, and plots.
2. **app.py** (Streamlit or FastAPI or command line) plus a requirements.txt.
3. **Presentation deck**: Template provided. Must include business case, solution approach and choices considered, architecture diagram, metric table, latency numbers, and reflections on when contextual embeddings shine.
4. **README** with setup steps and how to run local queries (screenshots welcome).