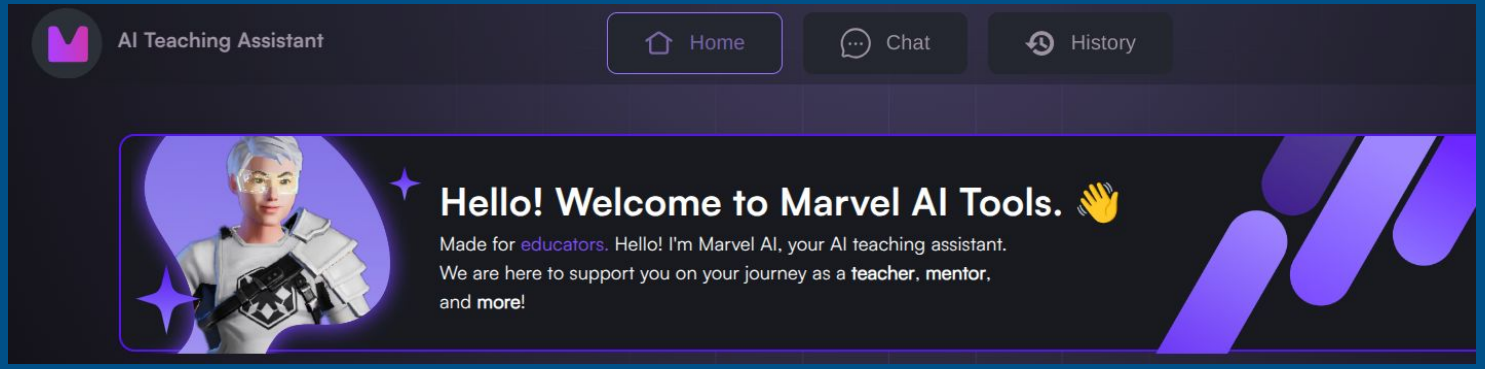
17.04.25

Reality AI Lab

Gen AI Bootcamp



Henry

🎤 Slide 1: Welcome & Icebreaker

- **Title:** “RAG: When Chatbots Go to the Library Before Answering”
- **Speaker Note:**

  “Ever gotten an answer from ChatGPT that felt like it was making stuff up? (Raise hands.) Yeah, me too—like that time I asked for my horoscope and it told me to buy toilet paper. With RAG, we give the model a library card.”

https://www.anthropic.com/engineering/building-effective-agents

📚 : What Is RAG?

- **Definition:**
  - *Retrieval-Augmented Generation* = Retrieval + Generation
- **Key Idea:**
  - First **retrieve** relevant documents (external “memory”)
  - Then **generate** a response conditioned on those docs
- **Analogy:**
  - Think of your LLM as the friend who always winges answers—and RAG is the one who actually checks Wikipedia first.

⚙️ The Two-Step Dance

1. **Retriever**
    - Searches a corpus (text, PDF, webpage)
    - Often uses embeddings + approximate nearest neighbors
2. **Generator**
    - Takes retrieved snippets as “context”
    - Feeds into an LLM (GPT-style)
    -

🔍 Under the Hood – Retriever Options

- **Sparse vs. Dense Retrieval**
  - Sparse: TF·IDF, BM25 (old school, but dependable)
  - Dense: Vector embeddings + FAISS/Pinecone
- **Pro Tip:**
  - “Dense retrieval is like remembering your ex’s birthday—way more precise than a grocery-list method.”

🤖 The Generator's Role

- **Prompt Engineering:**
  - How you stitch in retrieved docs
  - E.g., "Answer the question based ONLY on the following passages…"
- **Model Choices:**
  - GPT-3.5/4, Claude, Llama2, etc.

"If your prompt is too sloppy, your model will be the sloppy friend who shows up in pajamas."

🌟 Why RAG Matters

- **Accuracy Boost:**
  - Cuts hallucinations by grounding answers
- **Up-to-Date Knowledge:**
  - Plug in fresh data without re-training
- **Efficiency:**
  - Only retrieve what you need—no need to expand a 500B-parameter mode

Why this Matters

- **Grounded Answers**: The LLM “looks up” real text before answering, so it’s far less likely to make stuff up.
- **Up-to-Date**: You can add or replace PDFs at any time—no need to retrain the LLM.
- **Efficient**: Only a handful of chunks go into the prompt, keeping token counts low and costs down.

In a nutshell, this pipeline gives your chatbot a “library card”—it actually checks the books before it speaks!

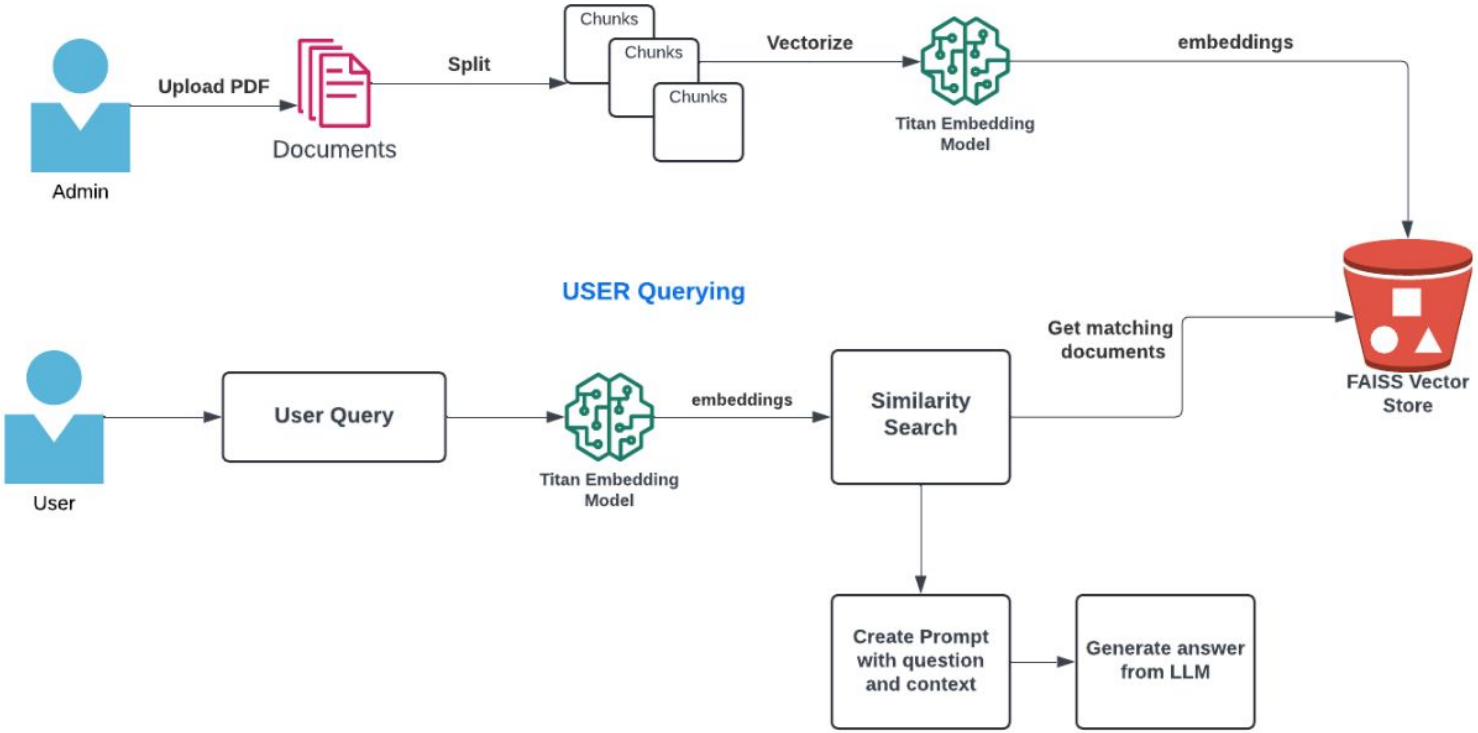
🚧 Slide 7: Common Pitfalls

- **Retriever Fails:**
  - Garbage in → garbage out
- **Context Window Overload:**
  - Too many tokens → performance dips
- **Latency Trade-off:**
  - “Faster than Googling yourself, slower than just winging it.”
- **Mitigation Tips:**
  - Cache frequent queries
  - Chunk & filter aggressively

🔧 Hands-On Example (Demo Outline)

1. **Indexing:**
    - Take your docs (e.g., product manual PDFs)
    - Embed + store in FAISS
2. **Query Pipeline:**
    - User asks “How do I reset my password?”
    - Retriever fetches top-5 relevant chunks
    - LLM generates a crisp answer
3. **Live Demo**

Our Demo


Admin
Upload PDF
Documents
Split
Chunks
Chunks
Chunks
Vectorize
Titan Embedding Model
embeddings
FAISS Vector Store
USER Querying
User
User Query
Titan Embedding Model
embeddings
Similarity Search
Get matching documents
Create Prompt with question and context
Generate answer from LLM

1. **Admin Uploads PDF**
   An administrator (you) uploads one or more PDF documents into the system.

2. **Split into Chunks**
   Each PDF is automatically split into smaller “chunks” or passages (e.g. a few sentences or a paragraph each). This makes retrieval more precise than working with whole documents.

3. **Vectorize with Titan Embedding Model**
   Each chunk is passed through the Titan Embedding Model, which converts text into a high-dimensional vector (an embedding) that captures its semantic meaning.

4. **Store Embeddings in FAISS**
   Those embeddings are stored in a FAISS vector store (an efficient similarity-search index).

User Submits Query
A user asks a question (e.g. “How do I reset my password?”).

Query Embedding
The query is sent through the **same** Titan Embedding Model to produce its embedding.

Similarity Search
FAISS performs a nearest-neighbor search between the query embedding and the stored chunk embeddings. It returns the top-k most semantically relevant chunks.

Create Prompt with Context
Those retrieved chunks are stitched into a prompt template along with the user’s question.