MedicalGPT – What I Built & Why (Personal Notes)

1. Goal (Why I started this project)

- I already knew ML/DL theory but lacked **hands-on LLM + RAG experience**
- Interviewers ask *"Have you worked with RAG, embeddings, vector DBs?"*
- Instead of learning theory first, I decided to **build first, understand while building**
- Goal was **learning + resume**, not production product

---

2. First step – LLM API connectivity

What I did

- Created a simple `main.py`
- Called an LLM using LangChain + OpenRouter
- Asked a basic question like *"Explain what anatomy is"*

Why I did it

- To confirm:
  - API key works
  - Provider works
  - LangChain → LLM pipeline works
- If this fails, nothing else matters

Interview takeaway

- Always isolate **LLM connectivity** before adding complexity

---

3. PDF loading (Document ingestion)

What I did

- Used a real medical textbook PDF (Anatomy & Physiology)
- Loaded it using `PyPDFLoader`

Why I did it

- RAG always starts with **external knowledge**
- LLMs don't magically know your documents
- PDFs need to be converted into text before use

Interview takeaway

- RAG = *LLM + external documents*
- First step is **document ingestion**

---

4. Chunking (Very important concept)

What I did

- Split large PDF pages into smaller chunks
- Used:
  - `chunk_size = 500`
  - `chunk_overlap = 100`
- 500+ pages → ~2000–4000 chunks

Why chunking is required

- LLMs have **context length limits**
- You cannot pass an entire book to the model
- Smaller chunks:
  - Fit into context
  - Improve relevance
  - Reduce noise

Why chunk overlap is used

- Prevents cutting meaning in half
- Keeps semantic continuity between chunks

Engineering tradeoff

- Small chunks → better precision, more chunks
- Large chunks → fewer chunks, more noise
- 500/100 is a **balanced industry-standard choice**

Interview takeaway

- Chunking is like **batching text semantically**
- Chunk size directly affects retrieval quality

---

5. Embeddings (Key interview topic)

What I learned

- **Embeddings ≠ LLM**
- LLMs generate text
- Embeddings convert text into **numerical vectors**

Why embeddings are needed

- Computers don't understand meaning directly
- Embeddings allow:
  - similarity search
  - semantic matching
- Similar meaning → vectors closer together

How the system understands similarity

- Question → embedding
- Chunks → embeddings
- Compare vectors using distance (cosine / dot product)

Interview takeaway

- Embeddings power **retrieval**
- LLMs power **generation**
- They serve different purposes

---

6. Vector database (Conceptual understanding)

What I attempted

- Used ChromaDB to store chunk embeddings
- Persisted embeddings locally

What happened

- Faced tooling issues on Windows + free API tier
- Retrieval calls blocked or hung

Engineering decision

- Stopped fighting the tool
- Switched to **manual RAG** for learning

Interview takeaway

- Tools change, **architecture matters**
- Engineers adapt instead of forcing unstable setups

---

7. Manual RAG (Key learning moment)

What I did

- Selected a **small subset of chunks**
- Passed them directly as context to the LLM
- Asked questions based on that context

Why this still counts as RAG

- Retrieval doesn't have to be vector-based
- Core idea:
  - External context
  - Grounded answers
- Vector DB is an optimization, not a requirement

Interview takeaway

- RAG = retrieve → augment → generate
- Retrieval can be manual or automated

---

8. Prompt design & safety

What I enforced

- "Answer strictly from provided context"
- "If not present, say I don't know"
- No diagnosis
- No prescriptions

Why this matters

- Prevents hallucinations
- Important for medical domain
- Shows safety awareness

Interview takeaway

- Prompting is **control logic**, not magic
- Constraints matter more than fancy prompts

---

9. Interactive chat loop (Final product)

What I built

- Terminal-based MedicalGPT chatbot
- User can ask multiple questions
- Exit using `exit` or `quit`

Why terminal chat

- No UI complexity
- Focus on core logic
- Easy to debug
- Interviewers like simple demos

Interview takeaway

- UI is optional
- Core system design is what matters

---

10. Key engineering lessons (Very important)

- Always validate components **step by step**
- Separate:
  - API testing
  - PDF loading
  - chunking
  - retrieval
  - generation
- Don't fight tools endlessly
- Make reasonable tradeoffs
- Build first, optimize later

---

11. How I explain this project in interviews (short)

"I built a MedicalGPT chatbot using LangChain and OpenRouter.
I ingest medical PDFs, chunk them, construct context dynamically, and answer user questions grounded strictly in the document.
I explored embeddings and vector databases, understood their role in semantic retrieval, and adapted my design based on tooling constraints."