RAG Day■1 • 80/20 Code Cheatsheet (One■Pager)

RAG Day■1 • 80/20 Code Cheatsheet (One■Pager)

1) Pipeline (Executive Summary)
Input question → Embed question → Vector search (top■k) → Retrieve chunks → Prompt LLM with retrieved context → LLM answer

ASCII Map:

```
[User Q]
↓ embed(q)
[q■vec] —— similarity search (k)
↓ ■
[top■k chunks (text)] ■
↓ ■
[Prompt: system + user + retrieved context] → [LLM] → [Answer]
```

2) Vital 20% Code Blocks

```
# A. Chunking (preprocessing heart)
def chunk_text(text, chunk_size=500, overlap=50):
chunks = []
for i in range(0, len(text), chunk_size - overlap):
chunks.append(text[i:i+chunk_size])
return chunks
```

Why it matters: LLM context is limited; we split docs into retrievable pieces.
Knobs: chunk_size (precision/recall and token cost), overlap (context continuity).

```
# B. Embeddings + Vector Store (semantic index)
embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_texts(chunks, embeddings)
```

Why it matters: turns text into vectors so we can do semantic search.

```
# C. Retriever (information bridge)
retriever = vectorstore.as_retriever(search_type="similarity", search_kwargs={"k": 3})
```

Why it matters: controls how many chunks come back (precision vs recall).

```
# D. Retrieval■Augmented QA Chain (glue)
qa = RetrievalQA.from_chain_type(
llm=ChatOpenAI(),
retriever=retriever
)
```

```
# E. Final Run (end■to■end)
result = qa.run("What is this document about?")
```

3) High■Leverage Knobs (Tune These First)
• k (top■k): 2–5 for precise Q&A; 5–10 for exploratory queries.
• chunk_size / overlap: start 400–800 / 10–20% overlap. PDF■like docs may need 800–1200.
• embedding model: small/cheap vs larger/accurate (impacts retrieval quality).
• system prompt: steer style and ground rules (“answer only from provided context”).

4) Quick Debug Checklist (80/20)
■ Retrieval looks wrong? → print top■k chunks; manually read if relevant.
■ Hallucinations? → tighten system prompt; raise k slightly; increase overlap.
■ Too many tokens / slow? → reduce k; reduce chunk_size; use smaller LLM.
■ Missing answers? → broaden k; try different embedding model; adjust chunking strategy.
■ Eval smoke test: create 5–10 Q/A pairs with known answers; measure hit@k.

5) Micro■Eval (Day■1 Friendly)
• hit@k: % of questions whose gold chunk appears in top■k.
• Simple prompt test: ask 10 factual questions; count correct grounded answers.
• Log retrieved sources with scores for each answer.

6) Interview Flash Cards
Q1: Why chunking? → LLM context is limited; chunks enable targeted retrieval.
Q2: What’s an embedding? → Dense vector capturing semantic meaning for similarity search.
Q3: Where’s the “heart” of RAG? → The integration of retriever + LLM (RetrievalQA).
Q4: How do you reduce hallucinations? → Constrain to retrieved context, tune k/overlap, add citations.
Q5: How do you debug retrieval? → Inspect top■k chunks and similarity scores; adjust chunking/embeddings/k.

7) One■Minute Setup Snippet (Pseudo)

```
docs = load_documents(path)
chunks = [c for d in docs for c in chunk_text(d.text)]
vs = FAISS.from_texts(chunks, OpenAIEmbeddings())
qa = RetrievalQA.from_chain_type(llm=ChatOpenAI(), retriever=vs.as_retriever(search_kwargs={"k":3}))
qa.run("Your question here")
```

Notes:
• Focus only on the five vital blocks above on Day■1. Boilerplate (imports, logging, UI) can wait.
• Explain the pipeline in 30 seconds using the ASCII map; that’s your 80/20 interview edge.