# Private Chatbot — Ready-to-run Code Files

This document contains all the files you asked for. Save each section into the filename shown in the header (e.g., ingest.py, index\_builder.py, etc.).

## requirements.txt

streamlit  
sentence-transformers  
faiss-cpu  
pdfplumber  
python-docx  
torch  
transformers  
regex  
tqdm  
nltk

Notes: If you don’t want local generation remove transformers and keep only the retriever-related requirements.

## ingest.py

from pathlib import Path  
from typing import List, Dict  
import pdfplumber  
import docx  
  
  
def extract\_text\_from\_pdf(path: Path) -> str:  
 texts = []  
 with pdfplumber.open(path) as pdf:  
 for p in pdf.pages:  
 texts.append(p.extract\_text() or "")  
 return "\n".join(texts)  
  
  
def extract\_text\_from\_docx(path: Path) -> str:  
 doc = docx.Document(path)  
 return "\n".join(p.text for p in doc.paragraphs)  
  
  
# Simple character-based chunker with overlap  
def chunk\_text(text: str, chunk\_size: int = 800, overlap: int = 200) -> List[Dict]:  
 chunks = []  
 start = 0  
 n = len(text)  
 i = 0  
 while start < n:  
 end = min(start + chunk\_size, n)  
 chunk = text[start:end].strip()  
 if chunk:  
 chunks.append({"id": i, "text": chunk, "start": start, "end": end})  
 i += 1  
 start += chunk\_size - overlap  
 return chunks  
  
  
def ingest\_folder(folder\_path: str) -> List[Dict]:  
 p = Path(folder\_path)  
 all\_chunks = []  
 for file in p.iterdir():  
 if file.suffix.lower() == '.pdf':  
 text = extract\_text\_from\_pdf(file)  
 elif file.suffix.lower() == '.docx':  
 text = extract\_text\_from\_docx(file)  
 else:  
 continue  
 chunks = chunk\_text(text)  
 for c in chunks:  
 c['source'] = file.name  
 all\_chunks.extend(chunks)  
 return all\_chunks

## index\_builder.py

from sentence\_transformers import SentenceTransformer  
import numpy as np  
import faiss  
import pickle  
  
MODEL\_NAME = "all-MiniLM-L6-v2"  
  
  
def build\_index(chunks, model\_name=MODEL\_NAME, index\_path="faiss\_index.bin", meta\_path="meta.pkl"):  
 model = SentenceTransformer(model\_name)  
 texts = [c['text'] for c in chunks]  
 embeddings = model.encode(texts, show\_progress\_bar=True, convert\_to\_numpy=True)  
  
 # faiss expects float32  
 embeddings = embeddings.astype('float32')  
 dim = embeddings.shape[1]  
  
 index = faiss.IndexFlatL2(dim)  
 index.add(embeddings)  
  
 faiss.write\_index(index, index\_path)  
 with open(meta\_path, 'wb') as f:  
 pickle.dump(chunks, f)  
  
 print(f"Saved FAISS index to {index\_path} and metadata to {meta\_path}")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import argparse  
 from ingest import ingest\_folder  
  
 parser = argparse.ArgumentParser()  
 parser.add\_argument('--folder', required=True, help='Folder with PDFs/DOCX to ingest')  
 parser.add\_argument('--index', default='faiss\_index.bin')  
 parser.add\_argument('--meta', default='meta.pkl')  
 args = parser.parse\_args()  
  
 chunks = ingest\_folder(args.folder)  
 build\_index(chunks, index\_path=args.index, meta\_path=args.meta)

## search.py

from sentence\_transformers import SentenceTransformer  
import faiss  
import pickle  
  
MODEL\_NAME = "all-MiniLM-L6-v2"  
  
class Retriever:  
 def \_\_init\_\_(self, index\_path='faiss\_index.bin', meta\_path='meta.pkl', model\_name=MODEL\_NAME):  
 self.model = SentenceTransformer(model\_name)  
 self.index = faiss.read\_index(index\_path)  
 with open(meta\_path, 'rb') as f:  
 self.meta = pickle.load(f)  
  
 def retrieve(self, query: str, top\_k: int = 5):  
 q\_emb = self.model.encode([query], convert\_to\_numpy=True)  
 q\_emb = q\_emb.astype('float32')  
 D, I = self.index.search(q\_emb, top\_k)  
 results = []  
 for idx, dist in zip(I[0], D[0]):  
 meta = self.meta[idx]  
 results.append({"score": float(dist), "text": meta['text'], "meta": meta})  
 return results  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 r = Retriever()  
 q = input('Query: ')  
 res = r.retrieve(q, top\_k=5)  
 for i, r in enumerate(res, 1):  
 print('---')  
 print(i, r['score'], r['meta'].get('source'))  
 print(r['text'])

## local\_gen.py (optional — only if you want a local generation step)

from transformers import AutoTokenizer, AutoModelForCausalLM  
import torch  
  
MODEL = "distilgpt2" # small example; replace with your local model  
DEVICE = 'cuda' if torch.cuda.is\_available() else 'cpu'  
  
  
def generate\_answer(context: str, question: str, model\_name=MODEL, max\_new\_tokens: int = 150):  
 tokenizer = AutoTokenizer.from\_pretrained(model\_name)  
 model = AutoModelForCausalLM.from\_pretrained(model\_name).to(DEVICE)  
  
 prompt = f"Context:\n{context}\n\nQuestion: {question}\nAnswer:"  
 inputs = tokenizer(prompt, return\_tensors='pt').to(DEVICE)  
 outputs = model.generate(\*\*inputs, max\_new\_tokens=max\_new\_tokens, do\_sample=False)  
 text = tokenizer.decode(outputs[0], skip\_special\_tokens=True)  
 return text[len(prompt):].strip()

Notes: from\_pretrained will download weights the first time. Place the model in a local cache to avoid repeated downloads.

## streamlit\_app.py

import streamlit as st  
import tempfile  
import os  
from ingest import extract\_text\_from\_pdf, extract\_text\_from\_docx, chunk\_text  
from index\_builder import build\_index  
from search import Retriever  
  
INDEX\_PATH = 'faiss\_index.bin'  
META\_PATH = 'meta.pkl'  
  
st.set\_page\_config(page\_title='Private Document Chatbot')  
st.title('Private Document Chatbot (Local)')  
  
st.markdown('Upload PDF or DOCX files. All indexing/search stays local.')  
  
uploaded\_files = st.file\_uploader('Upload PDF or DOCX', accept\_multiple\_files=True)  
if uploaded\_files:  
 all\_chunks = []  
 for uf in uploaded\_files:  
 suffix = uf.name.split('.')[-1].lower()  
 with tempfile.NamedTemporaryFile(delete=False, suffix='.' + suffix) as tmp:  
 tmp.write(uf.getbuffer())  
 tmp\_path = tmp.name  
 if suffix in ['pdf']:  
 text = extract\_text\_from\_pdf(tmp\_path)  
 elif suffix in ['docx']:  
 text = extract\_text\_from\_docx(tmp\_path)  
 else:  
 st.warning(f"Unsupported file type: {suffix}")  
 continue  
 chunks = chunk\_text(text)  
 for c in chunks:  
 c['source'] = uf.name  
 all\_chunks.extend(chunks)  
  
 if all\_chunks:  
 st.info('Building index — this may take a moment')  
 build\_index(all\_chunks, index\_path=INDEX\_PATH, meta\_path=META\_PATH)  
 st.success('Index built and saved locally.')  
  
# Load existing index if present  
if os.path.exists(INDEX\_PATH) and os.path.exists(META\_PATH):  
 try:  
 retriever = Retriever(INDEX\_PATH, META\_PATH)  
 except Exception as e:  
 st.error(f"Failed to load index: {e}")  
 retriever = None  
  
 if retriever:  
 query = st.text\_input('Ask a question about your documents:')  
 top\_k = st.slider('Top K', min\_value=1, max\_value=10, value=5)  
  
 if st.button('Search') and query:  
 results = retriever.retrieve(query, top\_k=top\_k)  
 st.write('Top passages:')  
 for i, r in enumerate(results, 1):  
 st.markdown(f"\*\*Result {i} — score {r['score']:.4f} — source: {r['meta'].get('source','unknown')}\*\*")  
 st.write(r['text'])  
  
 if st.checkbox('Generate concise answer (local LLM)'):  
 try:  
 from local\_gen import generate\_answer  
 context = "\n\n".join([r['text'] for r in results])  
 with st.spinner('Generating...'):  
 answer = generate\_answer(context, query)  
 st.subheader('Generated Answer')  
 st.write(answer)  
 except Exception as e:  
 st.error(f"Generation failed: {e}")  
  
else:  
 st.info('No index found yet. Upload files above to create an index.')

## How to run

1. Create a virtual environment and install dependencies:

python -m venv venv  
source venv/bin/activate # linux/mac  
venv\Scripts\activate # windows  
pip install -r requirements.txt

1. Index a folder (optional) or use Streamlit uploader:

python index\_builder.py --folder ./my\_docs

1. Run Streamlit app:

streamlit run streamlit\_app.py

## Notes & caveats

* The first run of sentence-transformers or transformers will download model weights — you need internet for the initial download. Once the model is cached, the app is offline.
* For scanned PDFs you need OCR (Tesseract) — not included by default.
* For large corpora consider FAISS IVF and persisting embeddings on disk.

If you’d like, I can now: - package these into a ZIP in the sandbox so you can download them directly, or - adapt the app to strictly **avoid** any model downloads by using a pre-downloaded local model path, or - remove the local LLM pieces and optimize for speed and small memory footprint.

Tell me which of those you’d like next and I’ll prepare it.