# 📘 Local RAG Application Tutorial (MacBook M2 - Open Source Only)

## 🧱 Step 1: Set Up Your Environment

Install essential software using Homebrew and create a Python virtual environment:  
  
```bash  
brew install python@3.11  
brew install ollama  
brew install cmake  
pip install virtualenv  
  
python3 -m venv rag-env  
source rag-env/bin/activate  
pip install --upgrade pip  
```

## 📦 Step 2: Install Required Libraries

Install all open-source dependencies:  
  
```bash  
pip install langchain llama-index sentence-transformers \  
 chromadb faiss-cpu \  
 transformers accelerate \  
 bitsandbytes evaluate \  
 scikit-learn peft \  
 tqdm pandas  
```

## 🤖 Step 3: Run a Local LLM with Ollama

Install and use Mistral model with Ollama:  
  
```bash  
brew install ollama  
ollama pull mistral  
```  
  
Then in Python:  
  
```python  
from langchain.llms import Ollama  
llm = Ollama(model="mistral")  
```

## 🗃️ Step 4: Create a Vector Store

Use FAISS and SentenceTransformers to embed and store your documents:  
  
```python  
from langchain.vectorstores import FAISS  
from sentence\_transformers import SentenceTransformer  
  
texts = ["doc1 content", "doc2 content"]  
embedding\_model = SentenceTransformer('all-MiniLM-L6-v2')  
db = FAISS.from\_texts(texts, embedding\_model)  
```

## 🔍 Step 5: Retrieval-Augmented Generation

Build a RAG chain using LangChain:  
  
```python  
from langchain.chains import RetrievalQA  
  
qa\_chain = RetrievalQA.from\_chain\_type(  
 llm=llm,  
 retriever=db.as\_retriever()  
)  
  
print(qa\_chain.run("What is this document about?"))  
```

## 🧠 Step 6: Add a Reranker (Optional)

Improve results by reranking with a cross-encoder model:  
  
```bash  
pip install cross-encoder  
```  
  
```python  
from cross\_encoder import CrossEncoder  
  
re\_ranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")  
scores = re\_ranker.predict([(query, doc) for doc in retrieved\_docs])  
```

## 🧪 Step 7: Evaluate LLM Output

Use ROUGE or other metrics via HuggingFace Evaluate:  
  
```python  
import evaluate  
  
rouge = evaluate.load("rouge")  
results = rouge.compute(  
 predictions=["predicted answer"],  
 references=["expected answer"]  
)  
print(results)  
```

## 🔁 Step 8: Fine-tune the Model (Advanced)

Use PEFT + LoRA and HuggingFace Transformers for fine-tuning.  
  
Reference: https://github.com/huggingface/trl/blob/main/examples/sft\_trainer.py