# Hướng dẫn customize LDR cho 100% offline

Tuyệt vời! Bây giờ tôi có đủ thông tin để hướng dẫn chi tiết. Đây là roadmap hoàn chỉnh:

### 🎯 Strategy: Biến LDR Thành 100% Offline System

### Core Changes Cần Thực Hiện:

1. ✅ **Remove all web search engines** (SearXNG, Tavily, Google, etc.)
2. ✅ **Use only Ollama** (local LLM + local embeddings)
3. ✅ **FAISS for vector storage** (already in LDR)
4. ✅ **Focus on document processing** (PDF, Word, Markdown, Images)
5. ✅ **Remove internet dependencies**

### 📦 Bước 1: Clone và Setup Project

```
# Clone repository
git clone https://github.com/LearningCircuit/local-deep-research.git
cd local-deep-research

# Tạo branch mới cho offline version
git checkout -b offline-version

# Tạo virtual environment
python -m venv venv
source venv/bin/activate  # Linux/Mac
# venv\Scripts\activate   # Windows

# Install dependencies
pip install -r requirements.txt

# Install thêm packages cho offline
pip install faiss-cpu  # hoặc faiss-gpu nếu có GPU
pip install sentence-transformers
pip install pytesseract  # OCR cho ảnh
pip install Pillow
```

## 🔧 Bước 2: Configure Ollama cho Embeddings + LLM

### 2.1. Install và Setup Ollama

```
# Install Ollama
# macOS: brew install ollama
# Linux: curl -fsSL https://ollama.com/install.sh | sh
# Windows: Download từ https://ollama.com

# Start Ollama server
ollama serve
```

### 2.2. Download Models

```
# LLM Model (chọn 1)
ollama pull llama3.1          # Tốt, 4.7GB
ollama pull qwen2.5           # Tốt cho tiếng Việt
ollama pull mistral           # Nhẹ, 4.1GB

# Embedding Model (REQUIRED cho offline)
ollama pull nomic-embed-text  # Tốt nhất, 274MB
ollama pull all-minilm        # Nhẹ hơn, 46MB
ollama pull mxbai-embed-large # Mạnh hơn, 670MB
```

### 2.3. Test Ollama Embeddings

```
# test_ollama_embeddings.py
import ollama

# Test embedding
response = ollama.embeddings(
    model='nomic-embed-text',
    prompt='This is a test sentence'
)

print(f"Embedding dimension: {len(response['embedding'])}")
print(f"First 5 values: {response['embedding'][:5]}")
```

## 🔧 Bước 3: Tạo Offline Configuration File

Tạo file `config_offline.py`:

```
"""
Offline Configuration for Local Deep Research
Loại bỏ tất cả web search, chỉ dùng local documents
"""

import os
```

```
# ==================== OFFLINE MODE ====================
OFFLINE_MODE = True  # Force offline mode

# ==================== OLLAMA SETTINGS ====================
OLLAMA_CONFIG = {
    "host": "http://localhost:11434",
    "llm_model": "llama3.1",  # hoặc "qwen2.5", "mistral"
    "embedding_model": "nomic-embed-text",  # CRITICAL cho offline
    "context_length": 4096,
    "temperature": 0.7,
    "keep_alive": "5m",
}

# ==================== VECTOR STORE SETTINGS ====================
VECTOR_STORE_CONFIG = {
    "type": "faiss",  # Local vector database
    "persist_directory": "./offline_knowledge_base",
    "collection_name": "documents",
    "embedding_dimension": 768,  # nomic-embed-text = 768 dims
    "distance_metric": "cosine",  # cosine, euclidean, or dot_product
}

# ==================== DOCUMENT PROCESSING ====================
DOCUMENT_CONFIG = {
    "chunk_size": 1000,
    "chunk_overlap": 200,
    "supported_formats": [".pdf", ".docx", ".doc", ".txt", ".md", ".png", ".jpg", ".jpeg"
    "ocr_language": "vie+eng",  # Vietnamese + English
    "max_file_size_mb": 500,  # Maximum file size
}

# ==================== DISABLED FEATURES ====================
# Disable tất cả web search engines
DISABLED_SEARCH_ENGINES = [
    "searxng",
    "tavily",
    "google",
    "brave",
    "arxiv",
    "pubmed",
    "wikipedia",
    "github",
]

# Disable cloud LLM providers
DISABLED_LLM_PROVIDERS = [
    "openai",
    "anthropic",
    "google",
    "cohere",
]

# ==================== ENABLED FEATURES ====================
ENABLED_FEATURES = {
    "local_documents_only": True,
```

```
    "document_analysis": True,
    "conversation_history": True,
    "export_pdf": True,
    "export_markdown": True,
}

# ==================== DATABASE SETTINGS ====================
DATABASE_CONFIG = {
    "type": "sqlite",  # hoặc "sqlcipher" nếu cần encryption
    "path": "./offline_data/research.db",
    "encryption_enabled": False,  # Set True nếu dùng sqlcipher
}

# ==================== PATHS ====================
DATA_DIR = os.getenv("LDR_DATA_DIR", "./offline_data")
DOCUMENTS_DIR = os.path.join(DATA_DIR, "documents")
VECTOR_DB_DIR = os.path.join(DATA_DIR, "vector_store")
CACHE_DIR = os.path.join(DATA_DIR, "cache")

# Create directories
for directory in [DATA_DIR, DOCUMENTS_DIR, VECTOR_DB_DIR, CACHE_DIR]:
    os.makedirs(directory, exist_ok=True)
```

### 🔧 Bước 4: Tạo Offline Embedding Handler

Tạo file `offline_embeddings.py`:

```
"""
Offline Embedding Handler sử dụng Ollama
Thay thế OpenAI/cloud embeddings
"""

import ollama
import numpy as np
from typing import List, Union
from config_offline import OLLAMA_CONFIG

class OfflineEmbeddings:
    """
    Embedding handler hoàn toàn offline sử dụng Ollama
    """

    def __init__(self, model: str = None):
        self.model = model or OLLAMA_CONFIG["embedding_model"]
        self.host = OLLAMA_CONFIG["host"]

        # Verify model exists
        self._verify_model()

    def _verify_model(self):
        """Kiểm tra model đã được pull chưa"""
        try:
            # List available models
            models = ollama.list()
```

```
            model_names = [m['name'] for m in models.get('models', [])]

            if self.model not in model_names:
                print(f"⚠  Model {self.model} chưa được tải")
                print(f" Đang tải model...")
                ollama.pull(self.model)
                print(f"✓ Model {self.model} đã sẵn sàng")
            else:
                print(f"✓ Model {self.model} đã sẵn sàng")

        except Exception as e:
            print(f"✗ Lỗi khi verify model: {e}")
            raise

    def embed_text(self, text: str) -> List[float]:
        """
        Tạo embedding cho một đoạn text

        Args:
            text: Text cần embed

        Returns:
            List[float]: Vector embedding
        """
        try:
            response = ollama.embeddings(
                model=self.model,
                prompt=text
            )
            return response['embedding']
        except Exception as e:
            print(f"✗ Lỗi khi tạo embedding: {e}")
            raise

    def embed_batch(self, texts: List[str], show_progress: bool = True) -> np.ndarray:
        """
        Tạo embeddings cho nhiều texts

        Args:
            texts: List of texts
            show_progress: Hiển thị progress bar

        Returns:
            np.ndarray: Matrix of embeddings
        """
        embeddings = []
        total = len(texts)

        for i, text in enumerate(texts):
            if show_progress and (i % 10 == 0 or i == total - 1):
                print(f" Progress: {i+1}/{total} ({(i+1)/total*100:.1f}%)")

            embedding = self.embed_text(text)
            embeddings.append(embedding)

        return np.array(embeddings)
```

```
    def embed_query(self, query: str) -> List[float]:
        """
        Embed query (giống embed_text, để tương thích với LangChain)
        """
        return self.embed_text(query)

    def embed_documents(self, documents: List[str]) -> List[List[float]]:
        """
        Embed documents (để tương thích với LangChain)
        """
        return self.embed_batch(documents, show_progress=True).tolist()


# ==================== LangChain Compatible Wrapper ====================

class OllamaEmbeddingsLangChain:
    """
    Wrapper để tương thích với LangChain
    """

    def __init__(self, model: str = None):
        self.embeddings = OfflineEmbeddings(model)

    def embed_query(self, text: str) -> List[float]:
        return self.embeddings.embed_query(text)

    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        return self.embeddings.embed_documents(texts)


# ==================== TEST FUNCTION ====================

def test_embeddings():
    """Test offline embeddings"""
    print("🧪 Testing Offline Embeddings...\n")

    embedder = OfflineEmbeddings()

    # Test single text
    text = "This is a test sentence for embedding"
    embedding = embedder.embed_text(text)
    print(f"✅ Single embedding dimension: {len(embedding)}")
    print(f"   First 5 values: {embedding[:5]}\n")

    # Test batch
    texts = [
        "First document about AI",
        "Second document about machine learning",
        "Third document about deep learning"
    ]
    embeddings = embedder.embed_batch(texts)
    print(f"✅ Batch embeddings shape: {embeddings.shape}")
    print(f"   Dimensions: {embeddings.shape[^1]}\n")

    # Test similarity
```

```
    from numpy.linalg import norm

    def cosine_similarity(a, b):
        return np.dot(a, b) / (norm(a) * norm(b))

    sim_0_1 = cosine_similarity(embeddings[^0], embeddings[^1])
    sim_1_2 = cosine_similarity(embeddings[^1], embeddings[^2])

    print(f"📊 Similarity between doc 0 and 1: {sim_0_1:.4f}")
    print(f"📊 Similarity between doc 1 and 2: {sim_1_2:.4f}")

    print("\n✅ All tests passed!")


if __name__ == "__main__":
    test_embeddings()
```

**Test ngay:**

```
python offline_embeddings.py
```

## 🔟 Bước 5: Tạo Offline Vector Store với FAISS

Tạo file `offline_vector_store.py`:

```
"""
Offline Vector Store sử dụng FAISS
Hoàn toàn local, không cần internet
"""

import faiss
import numpy as np
import pickle
import os
from typing import List, Dict, Tuple
from offline_embeddings import OfflineEmbeddings
from config_offline import VECTOR_STORE_CONFIG

class OfflineVectorStore:
    """
    Vector store hoàn toàn offline với FAISS
    """

    def __init__(self, persist_directory: str = None):
        self.persist_directory = persist_directory or VECTOR_STORE_CONFIG["persist_direct
        self.embedding_dim = VECTOR_STORE_CONFIG["embedding_dimension"]
        self.distance_metric = VECTOR_STORE_CONFIG["distance_metric"]

        # Initialize embedder
        self.embedder = OfflineEmbeddings()

        # Initialize or load FAISS index
```

```
        self.index = None
        self.documents = []  # Store original documents
        self.metadatas = []  # Store metadata

        os.makedirs(self.persist_directory, exist_ok=True)

        # Try to load existing index
        if self._index_exists():
            self.load()
        else:
            self._create_index()

    def _create_index(self):
        """Tạo FAISS index mới"""
        if self.distance_metric == "cosine":
            # Cosine similarity = inner product on normalized vectors
            self.index = faiss.IndexFlatIP(self.embedding_dim)
        elif self.distance_metric == "euclidean":
            self.index = faiss.IndexFlatL2(self.embedding_dim)
        else:
            self.index = faiss.IndexFlatIP(self.embedding_dim)  # default

        print(f"✅ Created new FAISS index: {self.distance_metric}")

    def _normalize_vectors(self, vectors: np.ndarray) -> np.ndarray:
        """Normalize vectors for cosine similarity"""
        norms = np.linalg.norm(vectors, axis=1, keepdims=True)
        return vectors / (norms + 1e-10)

    def add_documents(
        self,
        texts: List[str],
        metadatas: List[Dict] = None,
        show_progress: bool = True
    ) -> None:
        """
        Thêm documents vào vector store

        Args:
            texts: List of text documents
            metadatas: List of metadata dicts
            show_progress: Show progress
        """
        if not texts:
            return

        if metadatas is None:
            metadatas = [{}] * len(texts)

        print(f"\n Adding {len(texts)} documents to vector store...")

        # Generate embeddings
        embeddings = self.embedder.embed_batch(texts, show_progress=show_progress)

        # Normalize if using cosine
        if self.distance_metric == "cosine":
```

```
            embeddings = self._normalize_vectors(embeddings)

        # Add to FAISS index
        self.index.add(embeddings.astype('float32'))

        # Store documents and metadata
        self.documents.extend(texts)
        self.metadatas.extend(metadatas)

        print(f"✅ Added {len(texts)} documents. Total: {len(self.documents)}")

    def search(
        self,
        query: str,
        k: int = 5
    ) -> Dict[str, List]:
        """
        Tìm kiếm documents liên quan

        Args:
            query: Search query
            k: Number of results

        Returns:
            Dict with documents, metadatas, distances
        """
        if len(self.documents) == 0:
            return {
                "documents": [],
                "metadatas": [],
                "distances": []
            }

        # Generate query embedding
        query_embedding = np.array([self.embedder.embed_query(query)])

        # Normalize if using cosine
        if self.distance_metric == "cosine":
            query_embedding = self._normalize_vectors(query_embedding)

        # Search
        k = min(k, len(self.documents))
        distances, indices = self.index.search(query_embedding.astype('float32'), k)

        # Gather results
        results = {
            "documents": [self.documents[i] for i in indices[^0]],
            "metadatas": [self.metadatas[i] for i in indices[^0]],
            "distances": distances[^0].tolist()
        }

        return results

    def save(self) -> None:
        """Lưu index và metadata xuống disk"""
        # Save FAISS index
```

```
        index_path = os.path.join(self.persist_directory, "faiss_index.bin")
        faiss.write_index(self.index, index_path)

        # Save documents and metadata
        data_path = os.path.join(self.persist_directory, "documents.pkl")
        with open(data_path, 'wb') as f:
            pickle.dump({
                'documents': self.documents,
                'metadatas': self.metadatas
            }, f)

        print(f"💾 Saved vector store to {self.persist_directory}")

    def load(self) -> None:
        """Load index và metadata từ disk"""
        try:
            # Load FAISS index
            index_path = os.path.join(self.persist_directory, "faiss_index.bin")
            self.index = faiss.read_index(index_path)

            # Load documents and metadata
            data_path = os.path.join(self.persist_directory, "documents.pkl")
            with open(data_path, 'rb') as f:
                data = pickle.load(f)
                self.documents = data['documents']
                self.metadatas = data['metadatas']

            print(f"✅ Loaded vector store from {self.persist_directory}")
            print(f"   Total documents: {len(self.documents)}")
        except Exception as e:
            print(f"⚠️  Could not load existing index: {e}")
            print(f"   Creating new index...")
            self._create_index()

    def _index_exists(self) -> bool:
        """Check if saved index exists"""
        index_path = os.path.join(self.persist_directory, "faiss_index.bin")
        data_path = os.path.join(self.persist_directory, "documents.pkl")
        return os.path.exists(index_path) and os.path.exists(data_path)

    def get_stats(self) -> Dict:
        """Get vector store statistics"""
        return {
            "total_documents": len(self.documents),
            "embedding_dimension": self.embedding_dim,
            "distance_metric": self.distance_metric,
            "persist_directory": self.persist_directory
        }

    def clear(self) -> None:
        """Xóa toàn bộ dữ liệu"""
        self._create_index()
        self.documents = []
        self.metadatas = []
        print("🗑️  Cleared all documents from vector store")
```

```
# ==================== TEST FUNCTION ====================

def test_vector_store():
    """Test offline vector store"""
    print("🧪 Testing Offline Vector Store...\n")

    # Create vector store
    vector_store = OfflineVectorStore(persist_directory="./test_vector_db")

    # Add documents
    documents = [
        "Python is a programming language",
        "Machine learning is a subset of AI",
        "Deep learning uses neural networks",
        "Natural language processing handles text",
        "Computer vision processes images"
    ]

    metadatas = [
        {"source": "doc1.txt", "topic": "programming"},
        {"source": "doc2.txt", "topic": "AI"},
        {"source": "doc3.txt", "topic": "AI"},
        {"source": "doc4.txt", "topic": "NLP"},
        {"source": "doc5.txt", "topic": "CV"}
    ]

    vector_store.add_documents(documents, metadatas)

    # Test search
    print("\n🔍 Testing search...")
    query = "What is AI?"
    results = vector_store.search(query, k=3)

    print(f"\nQuery: {query}")
    print(f"Top {len(results['documents'])} results:")
    for i, (doc, meta, dist) in enumerate(zip(
        results['documents'],
        results['metadatas'],
        results['distances']
    )):
        print(f"\n{i+1}. Distance: {dist:.4f}")
        print(f"   Document: {doc}")
        print(f"   Metadata: {meta}")

    # Save
    vector_store.save()

    # Test stats
    print(f"\n📊 Stats: {vector_store.get_stats()}")

    print("\n✅ All tests passed!")


if __name__ == "__main__":
    test_vector_store()
```

**Test:**

```
python offline_vector_store.py
```

### 📄 Bước 6: Document Processor với OCR Support

Tạo file `offline_document_processor.py`:

```
"""
Document Processor với OCR support cho ảnh
Xử lý PDF, Word, Markdown, Text, và Images
"""

import os
import fitz  # PyMuPDF
from docx import Document
from PIL import Image
import pytesseract
from typing import List, Dict
from config_offline import DOCUMENT_CONFIG

class OfflineDocumentProcessor:
    """
    Xử lý các loại documents khác nhau
    """

    def __init__(self):
        self.chunk_size = DOCUMENT_CONFIG["chunk_size"]
        self.chunk_overlap = DOCUMENT_CONFIG["chunk_overlap"]
        self.supported_formats = DOCUMENT_CONFIG["supported_formats"]
        self.ocr_lang = DOCUMENT_CONFIG["ocr_language"]
        self.max_size_mb = DOCUMENT_CONFIG["max_file_size_mb"]

    def process_file(self, file_path: str) -> Dict:
        """
        Xử lý file và trả về text + metadata

        Returns:
            Dict with 'text', 'chunks', 'metadata'
        """
        # Check file exists
        if not os.path.exists(file_path):
            raise FileNotFoundError(f"File not found: {file_path}")

        # Check file size
        size_mb = os.path.getsize(file_path) / (1024 * 1024)
        if size_mb > self.max_size_mb:
            raise ValueError(f"File too large: {size_mb:.1f}MB (max: {self.max_size_mb}MB

        # Get extension
        ext = os.path.splitext(file_path)[^1].lower()

        if ext not in self.supported_formats:
```

```
            raise ValueError(f"Unsupported format: {ext}")

        print(f"\n Processing: {os.path.basename(file_path)} ({size_mb:.1f}MB)")

        # Extract text
        if ext == '.pdf':
            text = self._process_pdf(file_path)
        elif ext in ['.docx', '.doc']:
            text = self._process_word(file_path)
        elif ext == '.txt':
            text = self._process_txt(file_path)
        elif ext == '.md':
            text = self._process_markdown(file_path)
        elif ext in ['.png', '.jpg', '.jpeg']:
            text = self._process_image(file_path)
        else:
            raise ValueError(f"Unsupported format: {ext}")

        # Create chunks
        chunks = self.chunk_text(text)

        # Metadata
        metadata = {
            "source": file_path,
            "filename": os.path.basename(file_path),
            "extension": ext,
            "size_mb": size_mb,
            "num_chunks": len(chunks),
            "text_length": len(text)
        }

        print(f"✅ Extracted {len(text)} characters, {len(chunks)} chunks")

        return {
            "text": text,
            "chunks": chunks,
            "metadata": metadata
        }

    def _process_pdf(self, file_path: str) -> str:
        """Extract text from PDF"""
        text = ""
        with fitz.open(file_path) as doc:
            for page_num, page in enumerate(doc):
                page_text = page.get_text()
                text += f"\n\n--- Page {page_num + 1} ---\n\n{page_text}"
        return text

    def _process_word(self, file_path: str) -> str:
        """Extract text from Word document"""
        doc = Document(file_path)
        paragraphs = [para.text for para in doc.paragraphs if para.text.strip()]
        return "\n\n".join(paragraphs)

    def _process_txt(self, file_path: str) -> str:
        """Read text file"""
```

```
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
            return f.read()

    def _process_markdown(self, file_path: str) -> str:
        """Read markdown file"""
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
            return f.read()

    def _process_image(self, file_path: str) -> str:
        """Extract text from image using OCR"""
        try:
            image = Image.open(file_path)
            text = pytesseract.image_to_string(image, lang=self.ocr_lang)
            return text
        except Exception as e:
            print(f"⚠  OCR failed: {e}")
            return f"[Image: {os.path.basename(file_path)} - OCR failed]"

    def chunk_text(self, text: str) -> List[str]:
        """
        Chia text thành chunks với overlap
        """
        if len(text) <= self.chunk_size:
            return [text]

        chunks = []
        start = 0
        text_len = len(text)

        while start < text_len:
            end = start + self.chunk_size

            # Try to break at sentence boundary
            if end < text_len:
                # Look for sentence endings
                for delimiter in ['. ', '.\n', '! ', '!\n', '? ', '?\n']:
                    last_delim = text[start:end].rfind(delimiter)
                    if last_delim != -1:
                        end = start + last_delim + len(delimiter)
                        break

            chunk = text[start:end].strip()
            if chunk:
                chunks.append(chunk)

            start += (self.chunk_size - self.chunk_overlap)

        return chunks

    def process_folder(self, folder_path: str) -> List[Dict]:
        """
        Xử lý tất cả files trong folder
        """
        results = []

        print(f"\n Scanning folder: {folder_path}")
```

```
        for root, dirs, files in os.walk(folder_path):
            for file in files:
                file_path = os.path.join(root, file)
                ext = os.path.splitext(file)[^1].lower()

                if ext in self.supported_formats:
                    try:
                        result = self.process_file(file_path)
                        results.append(result)
                    except Exception as e:
                        print(f"✗ Error processing {file}: {e}")

        print(f"\n✓ Processed {len(results)} files")
        return results


# ==================== TEST FUNCTION ====================

def test_processor():
    """Test document processor"""
    print("🧪 Testing Document Processor...\n")

    processor = OfflineDocumentProcessor()

    # Test text chunking
    sample_text = "This is sentence one. This is sentence two. " * 100
    chunks = processor.chunk_text(sample_text)

    print(f"✓ Chunking test:")
    print(f"   Original length: {len(sample_text)}")
    print(f"   Number of chunks: {len(chunks)}")
    print(f"   First chunk length: {len(chunks[^0])}")

    print("\n✓ All tests passed!")


if __name__ == "__main__":
    test_processor()
```

### 🔟 Bước 7: Offline Chat System với RAG

Tạo file `offline_rag_chat.py`:

```
"""
Offline RAG Chat System
Kết hợp Ollama LLM + FAISS Vector Store
"""

import ollama
from offline_vector_store import OfflineVectorStore
from offline_document_processor import OfflineDocumentProcessor
from config_offline import OLLAMA_CONFIG
from typing import List, Dict
```

```
class OfflineRAGChat:
    """
    Chat AI hoàn toàn offline với RAG
    """

    def __init__(
        self,
        llm_model: str = None,
        vector_store: OfflineVectorStore = None
    ):
        self.llm_model = llm_model or OLLAMA_CONFIG["llm_model"]
        self.vector_store = vector_store or OfflineVectorStore()
        self.doc_processor = OfflineDocumentProcessor()
        self.conversation_history = []

        self._verify_llm()

    def _verify_llm(self):
        """Verify LLM model exists"""
        try:
            ollama.show(self.llm_model)
            print(f"✓ LLM Model {self.llm_model} ready")
        except:
            print(f"⚠  Model {self.llm_model} not found. Pulling...")
            ollama.pull(self.llm_model)
            print(f"✓ Model {self.llm_model} ready")

    def ingest_file(self, file_path: str) -> None:
        """
        Đọc và lưu file vào knowledge base
        """
        # Process file
        result = self.doc_processor.process_file(file_path)

        # Create metadata for each chunk
        base_metadata = result["metadata"]
        metadatas = [
            {
                **base_metadata,
                "chunk_id": i,
                "chunk_text_length": len(chunk)
            }
            for i, chunk in enumerate(result["chunks"])
        ]

        # Add to vector store
        self.vector_store.add_documents(
            result["chunks"],
            metadatas,
            show_progress=True
        )

        # Save immediately
        self.vector_store.save()
```

```
    def ingest_folder(self, folder_path: str) -> None:
        """
        Đọc tất cả files trong folder
        """
        results = self.doc_processor.process_folder(folder_path)

        for result in results:
            base_metadata = result["metadata"]
            metadatas = [
                {
                    **base_metadata,
                    "chunk_id": i
                }
                for i in range(len(result["chunks"]))
            ]

            self.vector_store.add_documents(
                result["chunks"],
                metadatas,
                show_progress=False
            )

        self.vector_store.save()

    def query(
        self,
        question: str,
        use_context: bool = True,
        k: int = 5,
        temperature: float = 0.7
    ) -> Dict:
        """
        Trả lời câu hỏi với RAG

        Returns:
            Dict with 'answer', 'sources', 'context_used'
        """
        context_docs = []
        context_str = ""

        if use_context:
            # Search for relevant context
            search_results = self.vector_store.search(question, k=k)
            context_docs = search_results["documents"]

            if context_docs:
                context_str = "\n\n---\n\n".join(context_docs)
                print(f"\n Found {len(context_docs)} relevant passages")

        # Build prompt
        if context_str:
            prompt = f"""Dựa trên thông tin sau đây, hãy trả lời câu hỏi của người dùng m

THÔNG TIN TỪ TÀI LIỆU:
{context_str}
```

```
CÂU HỎI: {question}

Hãy trả lời dựa trên thông tin được cung cấp. Nếu thông tin không đủ để trả lời đầy đủ, h
        else:
            prompt = question

        # Call LLM
        response = ollama.chat(
            model=self.llm_model,
            messages=[
                {
                    'role': 'system',
                    'content': 'Bạn là trợ lý AI hữu ích, trả lời chính xác dựa trên thôn
                },
                {
                    'role': 'user',
                    'content': prompt
                }
            ],
            options={
                'temperature': temperature,
                'num_ctx': OLLAMA_CONFIG["context_length"]
            }
        )

        answer = response['message']['content']

        # Save to history
        self.conversation_history.append({
            'question': question,
            'answer': answer,
            'context_used': bool(context_str),
            'num_sources': len(context_docs)
        })

        return {
            'answer': answer,
            'sources': context_docs,
            'context_used': bool(context_str)
        }

    def stream_query(
        self,
        question: str,
        use_context: bool = True,
        k: int = 5
    ) -> str:
        """
        Stream response từ LLM
        """
        context_str = ""

        if use_context:
            search_results = self.vector_store.search(question, k=k)
            context_docs = search_results["documents"]
```

```
            if context_docs:
                context_str = "\n\n---\n\n".join(context_docs)
                print(f"\n Sử dụng {len(context_docs)} đoạn văn liên quan\n")

        if context_str:
            prompt = f"""Dựa trên thông tin sau, trả lời câu hỏi:

THÔNG TIN:
{context_str}

CÂU HỎI: {question}"""
        else:
            prompt = question

        # Stream
        stream = ollama.chat(
            model=self.llm_model,
            messages=[
                {'role': 'system', 'content': 'Bạn là trợ lý AI. Trả lời bằng tiếng Việt.
                {'role': 'user', 'content': prompt}
            ],
            stream=True,
            options={'num_ctx': OLLAMA_CONFIG["context_length"]}
        )

        full_response = ""
        for chunk in stream:
            content = chunk['message']['content']
            print(content, end='', flush=True)
            full_response += content

        print()
        return full_response

    def get_stats(self) -> Dict:
        """Get system statistics"""
        vector_stats = self.vector_store.get_stats()

        return {
            **vector_stats,
            "llm_model": self.llm_model,
            "conversation_turns": len(self.conversation_history)
        }


# ==================== MAIN APPLICATION ====================

def main():
    """Main application"""
    print("\n" + "="*60)
    print(" OFFLINE RAG CHAT SYSTEM")
    print("="*60 + "\n")

    # Initialize system
    chat = OfflineRAGChat()
```

```
    while True:
        print("\n" + "="*60)
        print("MENU:")
        print("1. Thêm file vào knowledge base")
        print("2. Thêm toàn bộ folder")
        print("3. Bắt đầu chat")
        print("4. Xem thống kê")
        print("5. Thoát")
        print("="*60)

        choice = input("\nChọn (1-5): ").strip()

        if choice == '1':
            file_path = input("Đường dẫn file: ").strip()
            try:
                chat.ingest_file(file_path)
            except Exception as e:
                print(f"✗ Lỗi: {e}")

        elif choice == '2':
            folder_path = input("Đường dẫn folder: ").strip()
            try:
                chat.ingest_folder(folder_path)
            except Exception as e:
                print(f"✗ Lỗi: {e}")

        elif choice == '3':
            print("\n CHAT MODE (gõ 'exit' để thoát)\n")
            while True:
                question = input(" Bạn: ").strip()

                if question.lower() == 'exit':
                    break

                if not question:
                    continue

                print("\n AI: ", end='')
                chat.stream_query(question)
                print()

        elif choice == '4':
            stats = chat.get_stats()
            print(f"\n Thống kê hệ thống:")
            for key, value in stats.items():
                print(f"   - {key}: {value}")

        elif choice == '5':
            print("\n Tạm biệt!")
            break


if __name__ == "__main__":
    main()
```

## 🧪 Bước 8: Test Toàn Bộ Hệ Thống

```
# Test embeddings
python offline_embeddings.py

# Test vector store
python offline_vector_store.py

# Test document processor
python offline_document_processor.py

# Run main application
python offline_rag_chat.py
```

## 📦 Bước 9: Package Requirements

Tạo file `requirements_offline.txt`:

```
# Core dependencies
ollama>=0.1.0
faiss-cpu>=1.7.4  # hoặc faiss-gpu
numpy>=1.24.0

# Document processing
pymupdf>=1.23.0  # PDF
python-docx>=1.0.0  # Word
Pillow>=10.0.0  # Images
pytesseract>=0.3.10  # OCR

# Optional
tqdm>=4.65.0  # Progress bars
```

## 🚀 Bước 10: Các Tối Ưu Hóa

### 10.1. Tăng Tốc Embedding với GPU

```
# Trong offline_embeddings.py, có thể dùng sentence-transformers
from sentence_transformers import SentenceTransformer

class FastOfflineEmbeddings:
    def __init__(self):
        # Sử dụng GPU nếu có
        self.model = SentenceTransformer(
            'sentence-transformers/all-mpnet-base-v2',
            device='cuda'  # hoặc 'mps' cho Mac M1/M2
        )

    def embed_batch(self, texts):
        return self.model.encode(texts, convert_to_numpy=True)
```

### 10.2. Cache Embeddings

```
import hashlib
import pickle

class CachedEmbeddings:
    def __init__(self, cache_dir="./embedding_cache"):
        self.cache_dir = cache_dir
        os.makedirs(cache_dir, exist_ok=True)

    def _get_cache_key(self, text):
        return hashlib.md5(text.encode()).hexdigest()

    def embed_text(self, text):
        cache_key = self._get_cache_key(text)
        cache_file = os.path.join(self.cache_dir, f"{cache_key}.pkl")

        if os.path.exists(cache_file):
            with open(cache_file, 'rb') as f:
                return pickle.load(f)

        # Generate embedding
        embedding = self.embedder.embed_text(text)

        # Cache it
        with open(cache_file, 'wb') as f:
            pickle.dump(embedding, f)

        return embedding
```

### 10.3. Parallel Document Processing

```
from concurrent.futures import ProcessPoolExecutor
from multiprocessing import cpu_count

def process_folder_parallel(self, folder_path: str):
    files = []
    for root, dirs, files_list in os.walk(folder_path):
        for file in files_list:
            ext = os.path.splitext(file)[^1].lower()
            if ext in self.supported_formats:
                files.append(os.path.join(root, file))

    # Process in parallel
    with ProcessPoolExecutor(max_workers=cpu_count()) as executor:
        results = list(executor.map(self.process_file, files))

    return results
```

### 🔐 Bước 11: (Optional) Add Encryption với SQLCipher

Nếu muốn encryption như LDR:

```
pip install pysqlcipher3
```

```
from pysqlcipher3 import dbapi2 as sqlite

def create_encrypted_db(db_path, password):
    conn = sqlite.connect(db_path)
    conn.execute(f"PRAGMA key = '{password}'")
    conn.execute("PRAGMA cipher_page_size = 4096")
    conn.execute("PRAGMA kdf_iter = 256000")
    return conn
```

### 📊 So Sánh: LDR Original vs Offline Version

| Feature | LDR Original | Offline Version |
|---|---|---|
| **Web Search** | ✅ Multiple engines | ❌ Disabled |
| **Local LLM** | ✅ Ollama | ✅ Ollama |
| **Local Embeddings** | ⚠️ Cloud default | ✅ Ollama embeddings |
| **Vector DB** | ✅ FAISS | ✅ FAISS |
| **Document Processing** | ✅ | ✅ Enhanced |
| **OCR** | ⚠️ Limited | ✅ Full support |
| **Internet Required** | ✅ Yes | ❌ No |
| **Privacy** | ⚠️ Depends on config | ✅ 100% local |
| **Setup Complexity** | High (Docker, SearXNG) | Medium (Ollama only) |
| **Dependencies** | Heavy | Light |

### ✅ Final Checklist

- [ ] Ollama installed và running
- [ ] Models downloaded (LLM + embedding)
- [ ] All Python files created
- [ ] Dependencies installed
- [ ] Test embeddings passed
- [ ] Test vector store passed
- [ ] Test document processor passed
- [ ] Main application running

- [ ] Documents ingested successfully
- [ ] Chat working with context

## Kết Luận

Bạn đã có hệ thống **100% offline RAG** với:

✓ **Ollama** - LLM và embeddings local
✓ **FAISS** - Vector database
✓ **Document processing** - PDF, Word, Markdown, Images với OCR
✓ **No internet** - Hoàn toàn offline sau khi setup
✓ **Privacy** - Dữ liệu không rời máy

**Performance Expected:**

- Embedding: ~100-500 docs/minute (tùy hardware)
- Search: <100ms cho 10k documents
- Chat response: 20-50 tokens/second (tùy model và hardware)

**Hardware Recommendations:**

- **Minimum:** 8GB RAM, CPU
- **Recommended:** 16GB RAM, GPU (RTX 3060+)
- **Optimal:** 32GB RAM, GPU (RTX 4090 hoặc Mac M2 Max)

Bạn có câu hỏi gì về implementation không?

⁂


1. https://www.youtube.com/watch?v=ilWbhwLyDQQ
2. https://docs.mem0.ai/components/vectordbs/dbs/faiss
3. https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/convert-text-string-embedding-locally-ollama.html
4. https://python.langchain.com/v0.1/docs/use_cases/question_answering/local_retrieval_qa/
5. https://www.youtube.com/watch?v=Oa3ORRbI7SA
6. https://www.youtube.com/watch?v=ii6K0IzKamk
7. https://collabnix.com/ollama-embedded-models-the-complete-technical-guide-to-local-ai-embeddings-in-2025/
8. https://stackoverflow.com/questions/76945728/langchain-offline-only
9. https://www.pluralsight.com/resources/blog/ai-and-data/langchain-local-vector-database-tutorial
10. https://docs.koog.ai/embeddings/
11. https://stackoverflow.com/questions/76848054/in-langchain-with-conversationalretrievalchain-is-it-possible-to-use-only-loc
12. https://www.youtube.com/watch?v=rNhwqOps6cg

13. https://docs.anythingllm.com/setup/embedder-configuration/local/ollama