Phase 1: Environment Setup & Document Loading - Walkthrough

Goal: Environment ready + documents readable
Date: 2026-02-09
Status: ✅ Complete

Overview

Phase 1 established the foundation for the RAG system by setting up the development environment, installing all necessary dependencies, configuring Ollama for LLM inference, and implementing robust PDF text extraction.

Tech Stack Decisions 🔒

Core Technologies (Locked)

Component	Technology	Rationale
Language	Python 3.10+	Industry standard for ML/AI, extensive library support
LLM Runtime	Ollama	Local inference, no API costs, privacy-focused
LLM Model	Phi-3 / TinyLlama	Lightweight models for 8GB RAM systems
Embeddings	sentence-transformers/all-MiniLM-L6-v2	384-dim, CPU-optimized, high quality
Vector DB	FAISS (CPU)	Fast similarity search, no external dependencies
PDF Processing	PyMuPDF (fitz)	Fast, reliable text extraction
Framework	LangChain	Modular RAG components, well-documented

Key Dependencies

```
langchain              # RAG framework
langchain-community    # Community integrations
langchain-huggingface  # HuggingFace embeddings
langchain-ollama       # Ollama LLM integration
faiss-cpu              # Vector similarity search
sentence-transformers  # Embedding models
pypdf                  # PDF utilities
pymupdf                # PDF text extraction
```

Environment Setup ✅

1. Python Virtual Environment

Created isolated environment to avoid dependency conflicts:

```
# Create virtual environment
python -m venv venv

# Activate (Windows)
venv\Scripts\activate

# Verify Python version
python --version  # Should be 3.10+
```

Result: ✅ Clean, isolated Python environment

2. Install Dependencies

Installed all required packages from `requirements.txt`:

```
pip install -r requirements.txt
```

Installed Packages:

- ✅ LangChain ecosystem (langchain, langchain-community, langchain-huggingface, langchain-ollama)
- ✅ Vector database (faiss-cpu)
- ✅ Embedding models (sentence-transformers)
- ✅ PDF processing (pypdf, pymupdf)

Verification: Created `check_env.py` to verify all imports work correctly.

Ollama Setup ✅

1. Install Ollama

Downloaded and installed Ollama from ollama.com

Platform: Windows
Installation: Standard installer

2. Pull LLM Models

Downloaded lightweight models suitable for 8GB RAM:

```
# Pull Phi-3 model (3.8B parameters)
ollama pull phi3

# Alternative: TinyLlama (1.1B parameters)
ollama pull tinyllama
```

Models Available:

```
NAME            SIZE    MODIFIED
phi3:latest     2.3 GB  [timestamp]
tinyllama       637 MB  [timestamp]
```

3. Test Ollama

Verified Ollama works with sample prompt:

```
ollama run phi3 "What is machine learning?"
```

Result: ✅ Ollama responding correctly with coherent answers

Configuration: Set to use `tinyllama` in `config/settings.py` for lower RAM usage:

```
LLM_MODEL = "tinyllama"  # 1.1B params, optimized for low-RAM systems
```

Repository Structure ✅

1. GitHub Repository

Created and initialized Git repository:

```
git init
git add .
git commit -m "Initial commit: RAG system foundation"
```

Repository: Clean, version-controlled codebase

2. Folder Structure

Organized modular architecture:

```
RAG project/
├── config/
│   └── settings.py          # Centralized configuration
├── data/                    # Input documents (PDFs, TXT)
├── src/
│   ├── embeddings/
│   │   └── huggingface.py   # Embedding model
│   ├── ingestion/
│   │   ├── loader.py        # Document loading
│   │   └── splitter.py      # Text chunking
│   ├── llm/
│   │   └── ollama_llm.py    # LLM interface
│   ├── retrieval/
│   │   └── retriever.py     # RAG retrieval
│   ├── utils/
│   │   └── persistence.py   # Reporting utilities
│   └── vector_store/
│       └── store.py         # FAISS vector DB
├── scripts/
│   ├── ingest_data.py       # Ingestion pipeline
│   └── test_rag.py          # RAG testing
├── vector_db/               # FAISS index storage
├── venv/                    # Virtual environment
├── .gitignore               # Git ignore rules
├── requirements.txt         # Dependencies
├── README.md                # Project documentation
└── main.py                  # Main CLI interface
```

Design Principles:

- ✅ Modular components (easy to test/modify)
- ✅ Clear separation of concerns
- ✅ Scalable architecture

3. Git Ignore Configuration

Created `.gitignore` to exclude:

```
venv/           # Virtual environment
__pycache__/    # Python cache
*.pyc           # Compiled Python
vector_db/      # Generated FAISS indices
.env            # Environment variables
```

Result: ✅ Clean repository without unnecessary files

PDF Text Extraction ✅

Implementation: `loader.py`

Created robust document loader with PyMuPDF (fitz):

Key Features

1. Page-wise Text Extraction

```python
def load_pdf_with_pymupdf(file_path: os.PathLike) -> List[Dict]:
    """
    Loads PDF and extracts text page-by-page.
    Returns list of dictionaries with content and metadata.
    """
    doc_data = []
    doc = fitz.open(file_path)

    for page_num in range(len(doc)):
        page = doc.load_page(page_num)
        text = page.get_text("text")

        page_data = {
            "content": text,
            "metadata": {
                "source": file_name,
                "page": page_num + 1,
                "is_scanned": detect_scanned_pdf(page),
                "total_pages": len(doc)
            }
        }
        doc_data.append(page_data)

    return doc_data
```

2. Scanned PDF Detection

Automatically detects scanned pages (images) vs text-based PDFs:

```
def detect_scanned_pdf(page) -> bool:
    """
    Detects if page is likely scanned by checking text presence.
    Returns True if text length < 50 characters.
    """
    text = page.get_text().strip()
    return len(text) < 50
```

Benefits:

- ✅ Warns users about scanned pages
- ✅ Helps identify OCR requirements
- ✅ Improves data quality awareness

3. Multi-Format Support

Handles both PDF and TXT files:

```
def load_document(file_path: os.PathLike) -> List[Dict]:
    """Main entry point for document loading."""
    ext = os.path.splitext(file_path)[1].lower()

    if ext == ".pdf":
        return load_pdf_with_pymupdf(file_path)
    elif ext == ".txt":
        # Handle TXT files
        with open(file_path, 'r', encoding='utf-8') as f:
            content = f.read()
        return [{
            "content": content,
            "metadata": {"source": os.path.basename(file_path), "page": 1
        }]
    else:
        print(f"Unsupported format: {ext}")
        return []
```

4. Metadata Preservation

Each extracted page includes:

- **source**: Original filename
- **page**: Page number (1-indexed)
- **is_scanned**: Boolean flag for scanned pages
- **total_pages**: Total pages in document

Testing

Created `test_extraction.py` to verify extraction:

```
from src.ingestion.loader import load_document

# Test PDF loading
docs = load_document("data/sample.pdf")
print(f"Extracted {len(docs)} pages")
print(f"First page preview: {docs[0]['content'][:200]}...")
```

Results:

- ✅ Successfully extracts text from PDFs
- ✅ Preserves page numbers and metadata
- ✅ Detects scanned pages
- ✅ Handles TXT files

Verification & Testing ✅

Environment Check

Created check_env.py to verify setup:

```
import sys

def check_imports():
    """Verify all critical imports work."""
    try:
        import langchain
        import faiss
        import sentence_transformers
        import fitz  # PyMuPDF
        print("✅ All dependencies installed correctly")
        return True
    except ImportError as e:
        print(f"❌ Missing dependency: {e}")
        return False

if __name__ == "__main__":
    success = check_imports()
    sys.exit(0 if success else 1)
```

Result: ✅ All imports successful

Sample Document Loading

Tested with sample PDFs in `data/` directory:

```
python test_extraction.py
```

Output:

```
Opening PDF with PyMuPDF: sample.pdf
Extracted 5 pages
Page 1: [content preview]
Page 2: [content preview]
...
✅ All pages extracted successfully
```

Deliverables ✅

1. Clean Repository ✅

- ✅ Git repository initialized
- ✅ Proper `.gitignore` configuration
- ✅ Modular folder structure
- ✅ README documentation

Repository Status: Clean, organized, version-controlled

2. Working PDF Text Extraction ✅

- ✅ PyMuPDF integration
- ✅ Page-wise text extraction
- ✅ Metadata preservation (source, page numbers)
- ✅ Scanned PDF detection
- ✅ Multi-format support (PDF, TXT)

Extraction Status: Fully functional and tested

Configuration Files

`config/settings.py`

Centralized configuration:

```
import os

# Paths
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
DATA_DIR = os.path.join(BASE_DIR, "data")
DB_DIR = os.path.join(BASE_DIR, "vector_db")

# Models
LLM_MODEL = "tinyllama"  # 1.1B params for low-RAM systems
EMBEDDING_MODEL_NAME = "sentence-transformers/all-MiniLM-L6-v2"

# RAG Parameters
CHUNK_SIZE = 500
```

```
CHUNK_OVERLAP = 50
RETRIEVAL_K = 3  # Number of documents to retrieve
```

Benefits:

- ✅ Single source of truth for configuration
- ✅ Easy to modify parameters
- ✅ Consistent across all modules

Key Achievements

Technical Setup

- ✅ Python virtual environment configured
- ✅ All dependencies installed and verified
- ✅ Ollama installed with Phi-3 and TinyLlama models
- ✅ Git repository initialized with proper structure

Document Processing

- ✅ PyMuPDF integration for fast PDF extraction
- ✅ Page-wise text extraction with metadata
- ✅ Scanned PDF detection
- ✅ Multi-format support (PDF, TXT)

Code Quality

- ✅ Modular architecture
- ✅ Type hints for better code clarity
- ✅ Error handling for robust operation
- ✅ Testing scripts for verification

Next Steps

With Phase 1 complete, the foundation is ready for:

1. **Phase 2**: Text chunking and vectorization (✅ Complete)
2. **Phase 3**: Vector database creation (✅ Complete)
3. **Phase 4**: RAG query implementation

4. **Phase 5**: User interface and deployment

Usage Example

Loading Documents

```
from src.ingestion.loader import load_document

# Load a PDF
pdf_data = load_document("data/research_paper.pdf")

# Access extracted content
for page in pdf_data:
    print(f"Page {page['metadata']['page']}")
    print(f"Content: {page['content'][:100]}...")
    print(f"Scanned: {page['metadata']['is_scanned']}")
```

Running Environment Check

```
# Activate virtual environment
venv\Scripts\activate

# Check all dependencies
python check_env.py

# Test document extraction
python test_extraction.py
```

Troubleshooting

Common Issues

Issue: `ModuleNotFoundError` for dependencies
Solution: Ensure virtual environment is activated and run `pip install -r requirements.txt`

Issue: Ollama not responding
Solution: Check Ollama service is running: `ollama list`

Issue: PDF extraction returns empty text

Solution: Check if PDF is scanned (image-based). Consider OCR tools if needed.

Summary

Phase 1 successfully established a solid foundation for the RAG system:

✅ **Environment**: Clean Python virtual environment with all dependencies
✅ **LLM**: Ollama configured with lightweight models (Phi-3, TinyLlama)
✅ **Repository**: Well-organized, version-controlled codebase
✅ **Document Loading**: Robust PDF extraction with PyMuPDF
✅ **Metadata**: Page numbers, source tracking, scanned page detection

Status: Production-ready document loading pipeline 🚀

This walkthrough documents the completed Phase 1 implementation for future reference and onboarding.