Dora Project Specification

1. Project Overview

1.1 Purpose

Dora is a system that uses RAG (Retrieval-Augmented Generation) technology to retrieve knowledge from PDF documents and augment local LLM responses. By executing all processing in a local environment, it ensures privacy and security.

1.2 Version Information

- Project Name: Dora
- Version: 0.0.1
- Python Requirement: 3.10 or higher

1.3 Key Features

- Inference using local LLM (Ollama)
- Knowledge extraction from PDF documents
- Related document retrieval via vector search
- Answer generation using RAG
- Command-line interface (CLI)

2. System Architecture

2.1 Overall Structure

The system consists of the following layers:

1. **CLI Layer** (`cli.py`): Provides user interface
2. **LocalLLM Layer** (`llm.py`): LLM integration and RAG mode switching
3. **RAG Chain / Direct Query**: Two paths for query processing
4. **Knowledge Base Layer** (`knowledge_base.py`): Overall knowledge base management
5. **Foundation Layer**: DocumentProcessor, VectorStore (ChromaDB), Embeddings (HuggingFace)

Six Primary Components/Classes:

The Dora system architecture defines six primary components/classes:

1. **DocumentProcessor** (`document.py`): Handles PDF loading and text chunking
2. **VectorStore** (`vectorstore.py`): Manages vector database (ChromaDB) and embeddings
3. **KnowledgeBase** (`knowledge_base.py`): Integrates document processing and vector store
4. **RAGChain** (`rag.py`): Implements RAG chain combining retrieval and generation
5. **LocalLLM** (`llm.py`): Wrapper for local LLM with optional RAG support
6. **CLI** (`cli.py`): Command-line interface for user interaction

2.2 Data Flow

2.2.1 Document Addition Flow

1. Load PDF file using `DocumentProcessor.load_pdf()`
2. Parse PDF with `PyPDFLoader`
3. Split text using `RecursiveCharacterTextSplitter` (chunk_size=1000, chunk_overlap=200)
4. Add metadata (source, file_name)
5. Call `VectorStore.add_documents()` via `KnowledgeBase.add_document()`
6. Generate embeddings using `HuggingFaceEmbeddings` (paraphrase-multilingual-MiniLM-L12-v2)
7. Save to ChromaDB (collection name: "dora_kb")

2.2.2 Query Processing Flow (RAG Mode)

1. Receive user query via `LocalLLM.invoke()`
2. Call `RAGChain.get_answer()`
3. Retrieve related documents using `Retriever.invoke()` (k=4 documents)
4. Build context by concatenating document contents
5. Generate prompt using `PromptTemplate`
6. Generate answer using `OllamaLLM.invoke()`
7. Return final answer

2.2.3 Query Processing Flow (Normal Mode)

1. Receive user query via `LocalLLM.invoke()`
2. Generate answer directly using `OllamaLLM.invoke()`
3. Return final answer

3. Key Components

3.1 DocumentProcessor (`document.py`)

3.1.1 Role

Handles PDF file loading and text chunking.

3.1.2 Main Methods

- `load_pdf(file_path: str | Path) -> list`
  - Loads PDF file and splits into chunks
  - Returns: List of LangChain Document objects
  - Adds metadata (`source`, `file_name`) to each chunk

3.1.3 Configuration Parameters

- `chunk_size`: Default 1000 characters
- `chunk_overlap`: Default 200 characters
- Text splitting algorithm: `RecursiveCharacterTextSplitter`

3.1.4 Error Handling

- `FileNotFoundError`: When file does not exist
- `ValueError`: When file is not a PDF
- `RuntimeError`: When PDF loading fails

3.2 VectorStore (`vectorstore.py`)

3.2.1 Role

Manages vector store using ChromaDB and HuggingFace embedding model.

3.2.2 Main Methods

- `add_documents(documents: list) -> list[str]`
    - Vectorizes documents and adds to ChromaDB
    - Returns: List of added document IDs
- `similarity_search(query: str, k: int = 4) -> list`
    - Searches for documents similar to query
    - Returns: List of similar documents
- `similarity_search_with_score(query: str, k: int = 4) -> list[tuple]`
    - Searches with similarity scores
    - Returns: List of (document, score) tuples
- `as_retriever(**kwargs) -> Any`
    - Returns LangChain Retriever object
- `get_collection_info() -> dict`
    - Gets collection information
    - Returns: `{"count": int, "exists": bool, "collection_name": str}`

3.2.3 Configuration Parameters

- `persist_directory`: Default `.dora/kb` (relative to project root)
    - This is the default directory path where vector database data is stored
    - The full path structure: `<project_root>/.dora/kb/`
- `embedding_model_name`: Default `paraphrase-multilingual-MiniLM-L12-v2`
- `collection_name`: Fixed value `"dora_kb"`
- Embedding settings:
    - `device`: "cpu"
    - `normalize_embeddings`: True

3.2.4 Technical Details

- ChromaDB: Persistent vector database
- Embedding model: Via sentence-transformers library

- Auto-persistence: ChromaDB 0.4.x and later automatically saves data

3.3 KnowledgeBase (`knowledge_base.py`)

3.3.1 Role

Integrates document processing and vector store to manage the overall knowledge base.

3.3.2 Main Methods

- `add_document(file_path: str | Path) -> int`
    - Adds PDF file to knowledge base
    - Returns: Number of chunks added
- `search(query: str, k: int = 4) -> list`
    - Searches the knowledge base
    - Returns: List of related documents
- `get_retriever(**kwargs) -> Any`
    - Gets Retriever object for searching
- `clear() -> None`
    - Deletes all documents in the knowledge base
- `get_info() -> dict`
    - Gets knowledge base information
    - Returns: `{"count": int, "exists": bool, "collection_name": str}`
- `list_documents() -> list[str]`
    - Gets list of document sources in knowledge base
    - Returns: List of file paths

3.3.3 Configuration Parameters

- `kb_directory`: Default `.dora/kb` (relative to project root)
    - This is the default directory path where the knowledge base data is stored
    - The full path structure: `<project_root>/.dora/kb/`
- `chunk_size`: Default 1000 characters
- `chunk_overlap`: Default 200 characters
- `embedding_model_name`: Default `paraphrase-multilingual-MiniLM-L12-v2`

3.4 RAGChain (`rag.py`)

3.4.1 Role

Implements RAG chain that integrates knowledge base retrieval and LLM generation.

3.4.2 Main Methods

- `invoke(query: str) -> dict[str, Any]`
  - Executes RAG chain
  - Returns: `{"result": str, "source_documents": list}`
- `get_answer(query: str) -> str`
  - Gets answer only
  - Returns: Generated answer text

3.4.3 Prompt Template

```
Use the following pieces of context to answer the question at the end.
If you don't know the answer, just say that you don't know, don't try to
make up an answer.

Context:
{context}

Question: {question}

Answer:
```

3.4.4 Configuration Parameters

- `k`: Default 4 (number of documents to retrieve)

3.4.5 Processing Flow

1. Receive query
2. Retrieve related documents using Retriever (k documents)
3. Build context by concatenating document contents
4. Embed context and query into prompt template
5. Generate answer using LLM
6. Return answer and source documents

3.5 LocalLLM (`llm.py`)

3.5.1 Role

Wrapper class for local LLM using Ollama. Integrates RAG functionality.

3.5.2 Main Methods

- `invoke(prompt: str) -> str`
  - Generates answer for prompt

    - When RAG is enabled: Uses RAGChain
    - When RAG is disabled: Queries LLM directly

- `invoke_with_sources(prompt: str) -> dict[str, Any]`
    - Generates answer with source document information (RAG mode only)
    - Returns: `{"result": str, "source_documents": list}`

3.5.3 Configuration Parameters

- `model_name`: Default `"llama3.2"`
- `use_rag`: Default `False`
- `knowledge_base`: Default `None`
- `rag_k`: Default `4`

3.5.4 Error Handling

- `ConnectionError`: When cannot connect to Ollama or model is unavailable
- `ValueError`: When RAG is enabled but knowledge_base is None
- `RuntimeError`: When answer generation fails

3.6 CLI (`cli.py`)

3.6.1 Role

Provides command-line interface.

3.6.2 Main Functions

- `run_test_llm() -> None`
    - Runs LLM operation verification test
- `interactive() -> None`
    - Starts interactive mode
    - Also handles subcommands (`add-doc`, `list-docs`, `clear-kb`)
- `add_document() -> None`
    - Adds PDF document to knowledge base
    - Arguments: `sys.argv[2]` or `sys.argv[1]` (subcommand format)
- `list_documents() -> None`
    - Displays list of documents in knowledge base
- `clear_knowledge_base() -> None`
    - Clears knowledge base (with confirmation prompt)

3.6.3 Internal Functions

- `_initialize_knowledge_base() -> tuple[KnowledgeBase | None, bool]`
  - Initializes knowledge base and determines RAG availability
- `_initialize_llm(use_rag: bool, kb: KnowledgeBase | None) -> LocalLLM`
  - Initializes LLM
- `_handle_subcommands() -> bool`
  - Handles subcommands

3.6.4 CLI Command List

Command	Entry Point	Description
`dora-test`	`dora.cli:run_test_llm`	LLM operation verification
`dora`	`dora.cli:interactive`	Start interactive mode
`dora add-doc <file>`	`dora.cli:interactive` (subcommand)	Add document
`dora list-docs`	`dora.cli:interactive` (subcommand)	List documents
`dora clear-kb`	`dora.cli:interactive` (subcommand)	Clear knowledge base
`dora-add-doc <file>`	`dora.cli:add_document`	Add document (standalone command)
`dora-list-docs`	`dora.cli:list_documents`	List documents (standalone command)
`dora-clear-kb`	`dora.cli:clear_knowledge_base`	Clear knowledge base (standalone command)

4. Technology Stack

4.1 Core Libraries

- **LangChain**: LLM application development framework
  - `langchain-ollama`: Ollama integration
  - `langchain-community`: Community components
  - `langchain-text-splitters`: Text splitting
  - `langchain_core`: Core functionality
- **ChromaDB**: Vector database
  - Version: 0.4.0 or higher
  - Persistent storage: SQLite-based

- **sentence-transformers**: Embedding model
  - Model: `paraphrase-multilingual-MiniLM-L12-v2`
  - Multilingual support (including Japanese)
- **PyPDF**: PDF processing
  - Version: 3.0.0 or higher

4.2 External Services

- **Ollama**: Local LLM runtime environment
  - Default model: `llama3.2`
  - Communication via HTTP API

4.3 Development Tools

- **uv**: Package manager
- **ruff**: Linter/Formatter
- **mypy**: Type checker
- **pytest**: Test framework

5. Data Structures

5.1 Document Chunk Structure

```
{
    "page_content": str,  # Text content of chunk
    "metadata": {
        "source": str,       # File path
        "file_name": str,    # File name
        "page": int          # Page number (added by PyPDFLoader)
    }
}
```

5.2 Vector Store Structure

- **Collection name**: `"dora_kb"` (fixed)
- **Storage location**: `.dora/kb/` directory
- **Default directory path**: `.dora/kb` (relative to project root)
  - This is the default path where ChromaDB stores vector database data
  - Example: If project root is `/home/user/Dora`, data is stored in `/home/user/Dora/.dora/kb/`
- **Embedding dimensions**: Model-dependent (`paraphrase-multilingual-MiniLM-L12-v2` is 384 dimensions)

5.3 Metadata

Each document chunk has the following metadata:

- `source`: Full path of original PDF file
- `file_name`: PDF file name
- `page`: Page number (auto-added by PyPDFLoader)

6. Configuration and Parameters

6.1 Default Configuration Values

DocumentProcessor

- `chunk_size`: 1000 characters
- `chunk_overlap`: 200 characters

VectorStore

- `persist_directory`: `.dora/kb`
- `embedding_model_name`: `paraphrase-multilingual-MiniLM-L12-v2`
- `collection_name`: `"dora_kb"`
- `device`: `"cpu"`
- `normalize_embeddings`: `True`

KnowledgeBase

- `kb_directory`: `.dora/kb`
- `chunk_size`: 1000 characters
- `chunk_overlap`: 200 characters
- `embedding_model_name`: `paraphrase-multilingual-MiniLM-L12-v2`

RAGChain

- `k`: 4 (number of documents to retrieve)

LocalLLM

- `model_name`: `"llama3.2"`
- `use_rag`: `False`
- `rag_k`: 4

6.2 Customizable Parameters

All major classes can have parameters customized during initialization. When using programmatically, parameters can be specified in each class constructor.

7. File Structure

```
Dora/
├── src/
│   └── dora/
│       ├── __init__.py          # Package initialization, main class
```

```
exports
│       ├── cli.py              # CLI interface
│       ├── document.py         # Document processing
│       ├── knowledge_base.py   # Knowledge base management
│       ├── llm.py              # Local LLM wrapper
│       ├── rag.py              # RAG chain implementation
│       └── vectorstore.py      # Vector store management
├── tests/                      # Test files
├── .dora/
│   └── kb/                     # Knowledge base data (auto-generated)
│       ├── chroma.sqlite3      # ChromaDB database
│       └── [UUID]/             # ChromaDB internal data
├── pyproject.toml              # Project configuration
├── LICENSE                     # MIT License
└── README.md                   # Project documentation
```

8. Error Handling

8.1 Error Types

File-Related

- `FileNotFoundError`: File does not exist
- `ValueError`: Invalid file format (not PDF)

System-Related

- `ConnectionError`: Cannot connect to Ollama, model unavailable
- `RuntimeError`: Runtime errors (PDF loading failure, answer generation failure, etc.)
- `OSError`: File system related errors

Configuration-Related

- `ValueError`: Invalid configuration (RAG enabled but knowledge_base is None, etc.)

8.2 Error Messages

All error messages include specific information to help users understand the problem and execute solutions.

9. Performance Characteristics

9.1 Processing Time

Based on evaluation results (using llama3.2 model):

- **PDF loading**: Depends on file size (typically a few seconds)
- **Embedding generation**: Depends on chunk count (approximately 0.1-0.5 seconds per chunk)
- **Vector search (Retrieval)**: Average 0.014 seconds (very fast)
- **LLM answer generation**:

    - Without RAG: Average 2.448 seconds
    - With RAG: Average 1.213 seconds (faster due to more focused context)
- **Total response time**:
    - Without RAG: Average 2.448 seconds
    - With RAG: Average 1.227 seconds (includes retrieval + generation)
    - **Target**: < 10 seconds (✓ Achieved)

9.1.1 Time to First Token (TTFT)

- **Without RAG**: Average 0.245 seconds
- **With RAG**: Average 0.121 seconds
- RAG mode shows faster TTFT due to more focused context

9.1.2 RAG Overhead Analysis

- **Retrieval time**: Average 0.014 seconds (negligible overhead)
- **RAG overhead**: Negative (-1.221 seconds average), meaning RAG actually improves response time
- This is because RAG provides more focused context, leading to faster generation

9.2 Memory Usage

- Embedding model: Approximately 500MB (on initial load)
- ChromaDB: Depends on database size
- LLM: Depends on model size (llama3.2 3B is approximately 2GB)

9.3 Storage

- Knowledge base: Depends on document size and chunk count
- Embedding vector per chunk: Approximately 1.5KB (384 dimensions × 4 bytes)

10. Security and Privacy

10.1 Data Processing

- All processing runs in local environment
- No internet connection required (except for initial model download)
- Data is not sent externally

10.2 Data Storage

- **Default storage location**: Knowledge base data is saved in `.dora/kb/` (relative to project root)
    - The default directory path is `.dora/kb`
    - This directory contains ChromaDB database files and vector embeddings
- Persists until user explicitly deletes
- Follows file system permissions

11. Extensibility

11.1 Customizable Elements

- Embedding model: Can be changed via `embedding_model_name` parameter
- LLM model: Can be changed via `model_name` parameter (Ollama-compatible models)
- Chunk size: Can be adjusted via `chunk_size` and `chunk_overlap`
- Number of retrieved documents: Can be adjusted via `rag_k` parameter
- Prompt template: Can be modified within `RAGChain` class

11.2 Future Extensibility

- Support for other document formats (Word, Markdown, etc.)
- Support for other vector databases
- Integration with other LLM providers
- Web UI addition
- API server addition

12. Limitations

12.1 Supported Formats

- PDF files only (PDFs containing text)
- Image-only PDFs or scanned PDFs require OCR processing

12.2 Model Requirements

- Only models supported by Ollama can be used
- Models must be downloaded in advance

12.3 Platform

- Python 3.10 or higher required
- Environment where Ollama runs is required

13. Testing

13.1 Test Structure

- `test_cli.py`: CLI functionality tests
- `test_document.py`: Document processing tests
- `test_knowledge_base.py`: Knowledge base management tests
- `test_llm.py`: LLM functionality tests
- `test_rag.py`: RAG chain tests
- `test_vectorstore.py`: Vector store tests
- `test_init.py`: Package initialization tests

13.2 Running Tests

```
uv run pytest
```

14. License

Document Created: 2026 **Last Updated**: 2026 **Version**: 0.0.1