Document Analysis and Q&A System for "The Enigmatic Research of Dr. X"

Technical Documentation

---

1. Introduction and Overview

This documentation covers a comprehensive NLP pipeline created for processing, analyzing, and interacting with "The Enigmatic Research of Dr. X". The system extracts content from varied document formats, processes it into searchable chunks, and enables advanced capabilities including semantic search, translation, summarization, and retrieval-augmented question answering (RAG).

Core Capabilities

- Document extraction from multiple formats
- Text chunking and embedding generation
- Vector database creation for semantic search
- Question-answering with conversation support
- Translation between multiple languages
- Text summarization with evaluation metrics
- Specialized evaluation metrics for quality assessment

Required Libraries

Document and Data Processing

- **python-docx**: Reading/writing Microsoft Word files
- **pdfplumber**: Extracting text and tables from PDFs

- **pymupdf (fitz)**: Alternative PDF extraction tool
- **openpyxl**: Reading/writing Excel files
- **pandas**: Data manipulation and analysis

Language Model & NLP

- **langchain**: Framework for LLM applications
- **tiktoken**: OpenAI's tokenization library
- **openai**: Official Python SDK for OpenAI APIs
- **transformers**: Hugging Face's library for pre-trained models
- **huggingface_hub**: Access to hosted models and datasets
- **langdetect**: Automatic language detection
- **rouge-score**: Evaluation metrics for text generation

Vector Search & Embedding

- **faiss-cpu**: Efficient vector similarity search
- **nomic**: Tools for high-dimensional embedding visualization
- **numpy**: Numerical computations and vector operations

---

2. Document Processing Pipeline

Task 1: Reading and Extracting Text from Publications

Objective: Extract readable text and table content from various file types (.docx, .pdf, .csv, .xlsx, etc.)

Input Types:

- Word files (.docx): Text and tables

- PDF files (.pdf): Text and tables
- Tabular files (.csv, .xlsx, .xls, .xlsm): Tables

Key Functions:

- `extract_text_from_file(file_path)`: Main entry point that routes to format-specific handlers
- `extract_text_from_docx(file_path)`: Processes Word documents
- `extract_text_and_tables_from_pdf(file_path)`: Handles PDF extraction
- `extract_all_tables(file_path)`: Processes CSV and Excel files
- `extract_tables_from_sheet(...)`: Formats tabular data into readable text

Output Format: Content blocks as dictionaries containing:

- type: "text" or "table"
- source: File path
- page_number: Page or sheet number
- content: Text or table content
- table: Raw table data (for Word documents)

Performance Tracking:

- Total tokens processed
- Total processing time
- Tokens per second (TPS)

Task 7: Smart Table-Aware Chunking

Objective: Extract, process, and chunk text and tables while preserving table integrity and semantic coherence

Components:

1. **Tokenizer Initialization**
   - Uses `tiktoken.get_encoding("cl100k_base")`
   - Ensures chunks don't exceed specified token limit (default: 1000)
2. **Semantic Chunking Logic**
   - `tokenize_and_chunk_semantic()`: Divides text into chunks while maintaining sentence boundaries
   - Preserves tables as whole units with `is_table` flag
3. **Multiple Format Support**
   - Handles .docx, .pdf, and .xlsx/.csv files
   - Format-specific extraction functions
4. **Output Format**
   - Two CSV formats: detailed (with is_table flag) and simplified
   - Includes source, page_number, chunk_number, content fields

Key Improvements:

- Sentence-based chunking vs. token-based chunking
- Table preservation with is_table flagging
- More natural chunk boundaries following sentence structure

---

3. Vector Processing and RAG System

Task 2: Breaking Down Publications

Objective: Divide large texts into manageable chunks and generate embeddings

Process:

1. Read pre-chunked text from CSV (source, page_number, chunk_number, content)
2. Count tokens using cl100k_base tokenizer
3. Generate embeddings using nomic-embed-text-v1 model
4. Record error-resilient processing with comprehensive logging
5. Track performance statistics (tokens, time, TPS)

Output: CSV file with chunk_number, page_number, content, embedding, token_count

Task 3: Building a Vector Database

Objective: Create a searchable vector database from document chunks

Steps:

1. Load chunk data and embeddings from CSV
2. Convert string embeddings to numeric format with NumPy
3. Create FAISS IndexFlatL2 for similarity search
4. Add all embeddings to the index
5. Store chunk metadata separately for retrieval

Task 4: Creating a RAG Q&A System

Objective: Build a question-answering system using document context

Process Flow:

1. Convert user question to vector embedding
2. Search FAISS index for most similar chunks
3. Select and format top k chunks as context

4. Build prompt with context, conversation history, and question
5. Generate answer using LLaMA-2 model
6. Update conversation history for follow-up questions
7. Track token usage and performance metrics

Conversational Support:

- Maintains last 2 Q&A pairs as history
- Enables understanding of follow-up queries
- Preserves context across conversation

---

4. Translation Systems

Task 5: Translation System

Objective: Translate publications between languages while preserving structure

Features:

- Multi-format support (.pdf, .docx, .txt)
- Automatic source language detection
- Translation to English or Arabic
- Two-step translation for non-English to Arabic
- Grammar correction for English outputs
- Paragraph structure preservation

Models Used:

- Translation (multilingual → English): Helsinki-NLP/opus-mt-mul-en
- Translation (English → Arabic): Helsinki-NLP/opus-mt-en-ar

- Grammar correction: vennify/t5-base-grammar-correction

Process Flow:

1. Read text from input file
2. Detect source language
3. Split text into manageable chunks
4. Translate chunks to target language
5. Apply grammar correction for English outputs
6. Save as new DOCX with appropriate suffix

Task 8: Improved Translation Pipeline

Objective: Enhanced document processing and translation pipeline

Key Improvements:

1. **Device Configuration**: Forced CPU usage to prevent memory issues
2. **Improved Text Chunking**: More accurate sentence splitting using regex
3. **Modular Structure**: Better organized functions for maintenance
4. **Fluent Grammar Correction**: Structured application of grammar improvement

Pipeline Components:

1. **Text Extraction**: Format-specific extraction methods
2. **Chunking**: Sentence-boundary-aware text splitting
3. **Translation Process**:
    - Language detection
    - Translation to intermediary English if needed

    - Translation to target language
    - Grammar correction for English outputs
4. **Output Saving**: DOCX format with preserved structure

---

5. Summarization Systems

Task 6: Text Summarization

Objective: Summarize long texts using transformer-based models

Implementation:

- Model: facebook/bart-large-cnn
- Function: `summarize_text(input_text, max_length=150)`
- Performance tracking: tokens, time, TPS

Summarization Experiment Tool:

- Compares multiple models: BART, PEGASUS, T5
- Tests different prompt styles:
    - No prefix (default)
    - "summarize: " prefix
    - "tl;dr: " prefix
- Comprehensive performance logging

Summary Evaluation with ROUGE:

- Evaluates generated summaries against reference summaries
- Calculates ROUGE-1, ROUGE-2, and ROUGE-L scores

- Returns comprehensive scoring dictionary
- Provides basis for quality assessment

---

6. Evaluation Metrics

Translation Preservation Score (TPS)

Definition: Measures semantic meaning preservation after translation cycles

Methodology:

1. Embed original and back-translated text
2. Calculate cosine similarity between embeddings
3. Score ranges from -1 to 1

Interpretation:

Score Range	Interpretation
0.90 - 1.00	Excellent preservation
0.75 - 0.89	Good preservation, minor changes
0.50 - 0.74	Partial meaning shift
< 0.50	Major semantic loss

Semantic Overlap Evaluation

Definition: Measures continuity between adjacent text chunks

Key Components:

- Embedding model: all-MiniLM-L6-v2
- Functions:

- `load_embeddings_from_csv(file_path)`
- `calculate_semantic_overlap(embeddings)`

Output: Average similarity score between adjacent chunks

Factual Consistency Score

Definition: Assesses factual alignment between summaries and source texts

Process:

1. Generate fact-based questions about the content
2. Use QA model to answer questions from both original and summary
3. Compare answers for matching information
4. Calculate score: (matching answers) / (total questions)

Applications:

- Evaluating AI summaries
- Auditing human summaries
- Benchmarking summarization systems

Answer Grounding Score

Definition: Quantifies how well an answer aligns with supporting context

Technical Specifications:

- Model: all-MiniLM-L6-v2
- Similarity: Cosine via util.pytorch_cos_sim
- Inputs: answer (str), retrieved_chunks (list of str)

Processing:

1. Encode answer and chunks as embeddings
2. Calculate similarity between answer and each chunk
3. Return maximum similarity as grounding score

Interpretation:

Score Range	Interpretation
0.90 - 1.00	Excellent grounding
0.75 - 0.89	Good grounding
0.50 - 0.74	Partial grounding
< 0.50	Weak or no grounding

Data Extraction Utilities

Purpose: Extract structured data from various file formats

Supported Formats:

- PDF (.pdf)
- Word (.docx)
- Excel (.xls, .xlsx)
- CSV (.csv)

Core Functions:

- `extract_tables_from_pdf(pdf_path)`
- `extract_tables_from_docx(docx_path)`
- `extract_text_from_csv(file_path)`

- `extract_text_from_excel(file_path)`
- `extract_structured_text(file_path)`

Error Handling:

- UnicodeDecodeError for encoding issues
- NotImplementedError for unsupported formats
- ValueError for unknown extensions