AI Engineer - NLP Skills Assessment

This document details a modular and extensible Natural Language Processing (NLP) pipeline developed for processing scientific documents. The system integrates text extraction, chunking, embedding, retrieval-augmented generation (RAG), translation, summarization, and performance measurement, utilizing state-of-the-art tools such as FAISS, LangChain, and Ollama.

1. Setup Instructions

1. Install Python dependencies:
```
pip install -r requirements.txt
```

2. Pull models and start Ollama:
```
ollama pull llama3
ollama pull nomic-embed-text
```

3. Ensure Ollama is running and data files are placed in the `data/` directory.

2. Functional Overview

- Text Extraction from .pdf, .docx, .txt, .csv, .xlsx
- Token-based Chunking using tiktoken with configurable size and overlap
- Embedding using nomic-embed-text via Ollama and stored in FAISS vector index
- RAG (Retrieval-Augmented Generation) chat interface using LangChain and llama3
- Translation with automatic language detection and fluency enhancement using LLMs
- Summarization with both abstractive and extractive modes using llama3
- Token-level performance logging for benchmarking (tokens/sec)
- Support for interactive and automated modes
- CLI support for configurable pipelines (data-dir, input-file, summarize, translate, etc.)

3. Creative Extensions

- Performance logger that records tokens per second to a JSON or JSONL format with timestamping.
- Smart handling of large text using recursive chunking and overlap strategy to avoid LLM context limit issues.
- Integrated language detection with fallback to error messages using langdetect.
- Post-processing of translations to enhance fluency with a second LLM pass.
- Automatic folder creation and structured JSON output for reproducibility and auditing.
- Interactive mode (RAG chatbot) and batch mode (document pipeline) coexist in a single script.
- Resilient chunking with tokenizer-aware decoding and character-position-based page estimation.
- Pipeline automatically avoids reprocessing if vector DB already exists.

4. Pipeline Components

• Extraction: Uses PyMuPDF, python-docx, and pandas to support multiple file types.
• Chunking: Uses cl100k_base tokenizer (OpenAI's tiktoken) with chunk size and overlap.
• Embedding: nomic-embed-text via Ollama into FAISS vector store.
• Summarization: LLM summarization with ROUGE evaluation metrics.
• Translation: Multi-step process using LLM with structure retention and improvement.
• RAG: Retrieves relevant vector-matching chunks and generates answer from LLM.
• Logging: JSON-based performance tracking per task for benchmarking.

5. Output Structure

```
outputs/
├── chunks/       # JSON files of document chunks
├── summaries/    # Summarized outputs
├── translated/   # Translated texts
├── metadata.json   # FAISS metadata
├── vector_db.faiss # FAISS vector index
├── performance.json# Performance logs
└── pipeline.log    # Execution logs
```

6. Models Used

• LLM: llama3 (via Ollama)

• Embeddings: nomic-embed-text (via Ollama)

7. Author

Created by Muhammad Ashiq Ameer
Powered by LangChain, Ollama, FAISS, and friends.