Project Title:

The Enigmatic Research of Dr. X – NLP Pipeline with Local LLMs

1: Introduction & Objective

Objective:

This project aims to process and analyze a set of research documents left behind by the fictional researcher Dr. X, who mysteriously disappeared. The main goals were:

- Extract and structure text from various file types
- Build a semantic search (RAG) Q&A system
- Enable summarization and translation
- Measure pipeline efficiency and performance
- Ensure all components work locally and offline

Key Constraints:

- Only local models and vector DBs were allowed
- No cloud APIs or external CV-based tools

- Focus on text-based NLP tasks

2. File Handling, Preprocessing, and Chunking

Supported File Types:

- `.docx`, `.pdf`, `.csv`, `.xlsx`, `.xls`, `.xlsm`

Text Extraction Strategy:

Used format-specific parsers:

- `python-docx` for Word documents
- `PyMuPDF` for PDFs with page number tags
- `pandas` for spreadsheet content
- Tables were flattened into a readable, text-friendly format

Tokenization & Chunking:

- Used `tiktoken`'s `cl100k_base` tokenizer (compatible with GPT models)
- Long documents were split into manageable chunks with overlapping windows

- Each chunk includes metadata: `filename`, `page_number`, and `chunk_number`

3: Vector Database & Semantic Search (RAG System)

Embeddings:

- Initially planned to use Nomic, but replaced with `sentence-transformers` (`all-MiniLM-L6-v2`) for fully local, efficient embeddings

Vector Store:

- Used **ChromaDB (PersistentClient)** to store chunks and metadata locally
- Queryable via cosine similarity for fast retrieval

RAG Q&A Pipeline:

- User question → embedded

- ChromaDB returns top `k` relevant chunks
- Local **LLaMA model via Ollama** (e.g., `llama2`, `tinyllama`) used to generate answers from context
- **Follow-up ready**: prompt could include past QA pairs for conversational memory

Page 4: Translation & Summarization

Step 5: Translation

- Integrated `argos-translate` to convert English → Arabic (offline)
- Automatically translates RAG answers
- Preserves structure/formatting and supports high fluency

Step 6: Summarization

- Summarizes long documents using `Falconsai/text_summarization` (very lightweight model ~350MB)
- Truncates long inputs
- Evaluates quality with **ROUGE-1, ROUGE-2, and ROUGE-L** metrics

Example Output:

- English Answer: "Dr. X researched zero-point energy..."
- Arabic: "أجرى الدكتور إكس أبحاثًا حول طاقة النقطة الصفرية..."

5: Performance Metrics & Optimization

Metrics Captured:

- Tokens per second for:
    - Embedding
    - RAG response
    - Summarization
    - Translation
- Logged with Python's `time.perf_counter()` and `tiktoken` for token counting

Optimization Techniques:

- Chose small, efficient models (`tinyllama`, `MiniLM`, `Falconsai`)
- Reused tokenizer across modules
- Batching used in summarization for speed

- Modular design avoids reloading models unnecessarily

Sample TPS:

Task	Tokens	Duration	TPS
RAG	620	1.20s	~516
Summary	1500	2.85s	~526

6: Creativity, Innovation & Final Thoughts

Innovation Highlights:

- Local multilingual RAG system with auto-translation
- Token-efficient chunking + table flattening
- Lightweight models that run on CPU
- ROUGE evaluation baked into summarization

Extra Features:

- Spreadsheet handling with multi-sheet support
- Optional translation of summaries
- Output logs support performance tuning

Challenges:

- Handling Nomic model fallback
- Keeping everything under memory limits for CPU
- Translation quality tuning without fine-tuning LLMs

Conclusion:

This pipeline meets all functional, technical, and performance goals with a clean, reproducible, and local-only approach. It is production-ready for offline environments, such as secure research labs or regulatory-sensitive domains.