LLM-Based Document & Data Analyzer

1️⃣ What exactly is an LLM-based Document & Data Analyzer?

It is a system that:

- Reads **documents** (PDF, DOCX, TXT)
- Understands **tabular data** (CSV, Excel)
- Accepts **natural language questions**
- Returns:
  - Correct answers
  - Logical explanations
  - (Optionally) source references

⚠️ Important:

> LLM **does NOT store knowledge —** it reasons on top of **your uploaded data**.

2️⃣ Problem this project solves (VERY IMPORTANT)

Real-world problem:

- Documents are **long**
- Data is **scattered**
- Humans:
  - Miss details
  - Make calculation mistakes
  - Waste time

Example:

200-page report + 5 CSV files
Question:
“What are the main risks and how did revenue change YoY?”

Manual → ❌
LLM-based system → ✅

3️⃣ What LLM actually does in this project

LLM is used for **THINKING**, not storing data.

Task	Done by
Understanding question	LLM
Breaking question into parts	LLM
Finding relevant text	Vector DB
Doing calculations	Python
Writing final answer	LLM

4️⃣ Why we cannot directly give documents to LLM

❌ Token limit
❌ Slow
❌ Hallucination
❌ Expensive

Solution:

We **index documents** instead of sending everything.

5️⃣ Core Concept: RAG (Retrieval-Augmented Generation)

Simple meaning:

> “Search first → then think → then answer”

Flow:

```
User Question

   ↓

Relevant Document Retrieval

   ↓

LLM Reasoning on Retrieved Content

   ↓

Final Answer
```

⚠️ But we will go **beyond basic RAG** later.

6️⃣ Types of Data in this Project

A. Unstructured Data (Text)

- PDFs
- DOCX
- TXT

Handled using:

- Text extraction
- Chunking
- Embeddings
- Vector search

B. Semi-Structured Data (Tables)

- CSV
- Excel
- Tables inside PDFs

Handled using:

- Pandas
- DuckDB / SQL
- Python execution

7️⃣ Why we separate TEXT and TABLE data

Because:

- LLMs are **bad at math**
- Python is **perfect at math**

Data Type	Engine

Text reasoning	LLM
Numerical reasoning	Python
Final explanation	LLM

This makes system **accurate + reliable**.

8️⃣ What happens when user asks a question

Example question:

> "Compare Q2 revenue growth and explain risks."

Step-by-step:

1. LLM understands question
2. Splits into:
   - Numerical part (revenue growth)
   - Textual part (risks)
3. Retrieves:
   - Tables for numbers
   - Text for risks
4. Python calculates numbers
5. LLM explains results

9️⃣ What is Hallucination (and why we fight it)

Hallucination = LLM invents facts ❌

Why it happens:

- Missing data
- Weak prompts
- Overconfidence

Our prevention:

- Retrieval-only answers
- Source verification
- Validation step

🔟 Minimum Skills Needed (You already have most)

✓ Python basics
✓ Pandas
✓ API usage
✓ Basic NLP understanding

❌ No deep ML training needed
❌ No GPU needed

1️⃣ 1️⃣ High-Level System Components

Component	Purpose
File Loader	Read documents
Chunker	Break text
Embedding Model	Convert text to vectors

Vector DB	Semantic search
Query Analyzer	Understand question
Reasoning Engine	Logic + math
Answer Generator	Final response

1️⃣ 2️⃣ What makes THIS project interview-grade

- Multi-modal reasoning (text + data)
- Real-world relevance
- Anti-hallucination design
- Clean architecture
- Extensible system

1️⃣ 3️⃣ What this project is NOT

❌ Simple chatbot
❌ ChatGPT wrapper
❌ One-file script

This is a **SYSTEM**, not a demo.

1️⃣ 4️⃣ Project Outcome (End Goal)

By end, your system can answer:

- “Summarize this report”
- “Calculate average growth”
- “Compare two datasets”
- “Explain risks from documents”
- “Give source-backed insights”

🏗️ STEP 2: SYSTEM ARCHITECTURE

(Document & Data Analyzer – Reasoning-Based LLM System)

1️⃣ High-Level Architecture (Bird’s Eye View)

Think of the system as **6 clear layers**:

User

↓

UI Layer

↓

Query Understanding Layer

↓

Retrieval Layer

↓

Reasoning Layer

↓

Answer Generation Layer

2️⃣ UI Layer (Input & Output)

Responsibilities:

- Upload documents (PDF, CSV, Excel)
- Accept natural language questions
- Display answers + sources

Tools:

- Streamlit (best for start)
- OR FastAPI + frontend later

⚠️ UI does **no thinking**.

3️⃣ Document Ingestion Layer (Knowledge Creation)

What happens here:

- Files are loaded
- Text & tables are extracted
- Data is structured

For TEXT:

- PDF → text
- DOCX → text
- Chunked into small pieces

For TABLES:

- CSV / Excel → Pandas DataFrames

- Stored separately

📌 Important:

> Ingestion happens **once**, not per question.

4️⃣ Query Understanding Layer (MOST IMPORTANT)

Purpose:

Understand **what kind of question** user is asking.

LLM decides:

- Is it:
    - Summary?
    - Numerical calculation?
    - Comparison?
    - Explanation?
- Which data is needed:
    - Text?
    - Tables?
    - Both?

Output

json Copy code

```json
{
  "question_type": "comparison",
  "needs_text": true,
  "needs_table": true,
  "metrics": ["revenue", "growth"],
  "time_period": "Q2"
}
```

🔥 This is what makes project HARD.

5️⃣ Retrieval Layer (Finding Evidence)

A. Text Retrieval

- Convert question → embedding
- Vector search in DB
- Return top-k chunks

B. Table Retrieval

- Identify relevant DataFrame
- Filter rows/columns
- Send to reasoning layer

⚠️ LLM NEVER sees full data.

6️⃣ Reasoning Layer (Brain of System)

Split into 2 engines:

🔹 Numerical Reasoning

- Python executes:
  - Averages
  - Growth rates
  - Comparisons

🔹 Textual Reasoning

- LLM reads retrieved text
- Extracts insights
- Identifies causes, risks, explanations

7️⃣ Answer Validation Layer (ANTI-HALLUCINATION)

Before final output:

- Check:
  - Are numbers computed?
  - Are claims supported by retrieved text?
- If not:
  - Regenerate answer
  - OR say "Insufficient data"

🔥 This layer is rare & impressive.

8️⃣ Answer Generation Layer

LLM generates:

- Clear explanation
- Structured output
- Optional citations

Example:

```
Revenue grew by 12% in Q2, driven by increased sales in Region A.
Key risks include supply chain instability (Report.pdf, Page 18).
```

Project structure

```
doc_analyzer/
│
├── data/
│   ├── raw_docs/
│   └── processed/
│
├── ingestion/
│   ├── load_text.py
│   ├── load_tables.py
│   └── chunking.py
│
├── embeddings/
│   └── embed_store.py
│
├── query_engine/
│   ├── query_parser.py
│   └── intent_detector.py
│
├── reasoning/
│   ├── text_reasoner.py
│   ├── table_reasoner.py
│   └── validator.py
│
├── app.py
└── requirements.txt
```

🔟 Data Flow (IMPORTANT FOR EXPLANATION)

1. Upload docs → ingestion
2. User asks question
3. Query parser classifies intent
4. Retriever fetches evidence
5. Reasoning engine processes
6. Validator checks correctness
7. Final answer shown

1️⃣ 1️⃣ Why this architecture is STRONG

✓ Scalable
✓ Low hallucination
✓ Clear separation
✓ Extendable (images, audio later)