# Project Context-Engine: *Building a Proprietary Knowledge Retrieval Pipeline.*

Build a **Naive RAG (Retrieval-Augmented Generation) Pipeline**. The goal is to create a system that can answer questions based on a specific, "private" corpus of documents that the model was not trained on.

- **Dataset:** You will use the ebook **https://www.gutenberg.org/ebooks/55695** and your application should answer questions from this book

---

## Technical Requirements

You must implement the following components from scratch or using standard libraries like LangChain.

### 1. The Ingestion Pipeline [3 Marks]

- **Chunking Strategy:** Implement a fixed-size window chunking mechanism with a defined overlap
- **Vectorization:** Use a pre-trained embedding model (e.g., `all-MiniLM-L6-v2` or OpenAI's `text-embedding-3-small`) to convert chunks into dense vectors.
- **Storage:** Store these in a vector store. For this assignment, an in-memory store like `FAISS` or a lightweight `ChromaDB` instance is sufficient.

### 2. The Retrieval Engine [3 Marks]

- **Similarity Search:** Implement a function that takes a user query, embeds it, and performs a **Cosine Similarity** search to retrieve the top k (k=3 is recommended) relevant chunks.

### 3. The Generation Component [3 Marks]

- **Prompt Engineering:** Design a prompt template that explicitly instructs the LLM to *only* use the provided context to answer the question.
- **LLM Integration:** Feed the retrieved chunks and the user query into an LLM (e.g., GPT-3.5/4 via API or Llama-3 via Ollama/HuggingFace).

### 4. Evaluation [1 Mark]

- Propose a robust evaluation framework for the two critical stages of your RAG pipeline: the **Retriever** and the **Generator**. Specifically, detail how you would measure **Contextual**

**Precision** and **Faithfulness**. How do you theoretically and practically ensure the model's outputs are strictly grounded in the provided source material to eliminate the risk of hallucinations?

**Submission Requirements & Technical Specifications:**

Your submission must adhere to the following criteria:

- **A. Execution-Ready Notebook:** Submit a single `.ipynb` file containing all code and outputs. Ensure the notebook is saved in a "cleared" state of success—every cell must show a valid output without execution errors.
- **B. Evaluation Proposal:** Include a dedicated **Markdown cell** corresponding to Part 4. This section should provide a rigorous theoretical and practical proposal for evaluating your RAG system's performance (e.g., using the RAG Triad or RAGAS framework).
- **C. LLM Flexibility:** You have full autonomy in selecting your inference engine. You may utilize proprietary APIs (e.g., **OpenAI GPT-4, Google Gemini, Anthropic Claude**) or deploy local instances via **Ollama** or **Hugging Face Transformers**.
- **D. Embedding Model Choice:** There are no restrictions on the vectorization model. You may use industry standards (OpenAI `text-embedding-3`) or open-source BERT-based variants (e.g., `BGE-M3`, `ModernBERT`, or `MiniLM`).
- **E. Verification Results:** Your notebook must demonstrate the system's efficacy by answering **at least five distinct queries** derived from your provided dataset. The retrieved context and the final generated response must be clearly formatted and legible within the output cells.