# DataForge 2nd Round Pipeline

Team: Nikhil24380

Members: Nikhil Agrawal

## Problem Statement Brief

**Problem:** Most Retrieval-Augmented Generation (RAG) systems lack transparency, failing to explicitly show which retrieved information supports the generated answer.

- **Consequence:** This lack of clarity makes the system's output difficult to trust and verify.
- **Objective:** Design a simple, transparent RAG pipeline that delivers both a natural language answer and a structured explanation.
- **Desired Explanation Format:** Entities and relationships (triples) derived directly from the retrieved documents.
- **Emphasis:** Interpretability and clarity over complex graph reasoning or large-scale optimization.

# Comparative Summary of Existing RAG Approaches

- **Basic RAG** relies on semantic retrieval followed by answer generation, but provides no visibility into whether retrieved documents actually influenced the output; citations are often post-hoc, and there is no reasoning trace or interpretability.

- **Query Decomposition (Multi-Query) RAG** improves retrieval coverage by splitting a complex question into sub-queries, but does not explain how those sub-queries contribute to the final answer, which evidence was used, or why some retrieved information was discarded.

- **Graph-Based RAG (GraphRAG)** introduces structured entity–relation representations that improve factual consistency and multi-hop reasoning, yet the reasoning remains implicit: users cannot inspect which graph paths supported specific claims, how individual documents contributed, or which retrieved sources were ultimately unused.

- **Overall Limitation** across all three approaches is the absence of faithful, user-verifiable interpretability—none clearly expose what questions were asked, what evidence was used, and how that evidence directly supports the final answer.

# The Proposed Pipeline

### The core questions the proposed pipeline answers

1) What knowledge was used
2) How the different sources contributed towards the output
3) Why and which sources were unused after retrieving
4) How the core reasoning was structured for the given input query

The challenge is to provide faithful verifiable and interpretable steps, reasoning data sources etc.
I plan the pipeline to go from “Trust the model and here is the ans” to “Here are the exact questions asked, domains explored, evidence used, and how much each document contributed.”

# Possible Technologies to be used

1. Langchain, Langgraph, Langsmith for orchestration
2. An vector based document store
3. Python, json, React fastapi programming languages for building full stack system
4. Spacy/LLM for entity and relation extraction
5. FAISS/Milvus for vector retrieval, indexing and similarity search

# Stage 1: Data Storage and Chunking

To the addition of normal token based chucking i would like chucks to be stored with rich metadata that could be topics the data relates to the document it was a part of the main entities of the data contained in the chunk. In this system the chucks are embedded and retrieved but the metadata remains unchanged for any of the chucks retrieved this allows us to map if the document was actually used later or just retrieved as a part of the query decomposition.
This system would also let us quote citations for any form of data that is outputted and not just a mere answer to the query. The below is just an example of how the data could be thought to be stored to save all the metadata with the data actually used to generate the output.

**Example: Evidence-Aware Chunk Schema**

```
{
  "chunk_id": "DOC3_CH5",
  "document_id": "DOC3",
  "text": "Jakob Bernoulli introduced the Law of Large Numbers in his work Ars Conjectandi.",
  "entities": ["Jakob Bernoulli", "Law of Large Numbers", "Ars Conjectandi"],
  "relations": [
    ["Jakob Bernoulli", "introduced", "Law of Large Numbers"]
  ],
  "domain": "Probability Theory"
}
```

# Stage 2: User query decomposition

Using the input query directly for retrieval might limit the scope of output the user actually wanted and even for the generator LLM. Breaking down the query to sub smaller queries and linking them to the domain the queries belong to make the retrieval much more robust, can give a look of which queries fetches the most relevant results and which queries the least relevant. This can help us for a visible chain of queries and evidence used by the system of the LLM or the system decomposing and combining the documents for an output.

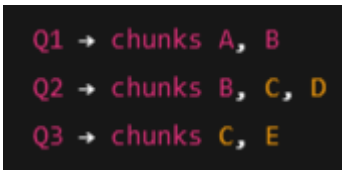
**Example: Query decomposition**

```
{
  "main_query": "What did Jakob Bernoulli contribute to mathematics?",
  "sub_queries": [
    {"id": "Q1", "question": "What are some key contributions of Jakob Bernoulli?", "domain": "mathematics"},
    {"id": "Q2", "question": "Who was Jakob Bernoulli?", "domain": "mathematics"},
    {"id": "Q3", "question": "What is the Bernoulli principle?", "domain": "physics"},
    {"id": "Q4", "question": "What is the significance of the Bernoulli numbers in mathematics?", "domain": "mathematics"}
  ]
}
```

# Stage 3: Multi-Query Retrieval

Using a single retrieval step can bias results toward a narrow subset of relevant information. In the proposed pipeline, each decomposed sub-query is used to perform **independent retrieval**, allowing the system to explore multiple semantic aspects of the user intent in parallel. This multi-query retrieval approach improves recall and reduces the risk of missing supporting evidence for specific sub-components of the question. By keeping retrieval results separated per sub-query before merging, the system can explicitly observe which sub-queries retrieve highly relevant information and which contribute little. This makes the retrieval process transparent and enables a clear mapping between sub-queries, retrieved evidence, and their influence on the final output. For the retrieval we use a hybrid multi stage retrieval i.e. use vector and keyword search to reduce false negatives and then have a second stage grader to verify if the retrieved chuck is accurate while maintaining the logs to again justify the reasoning and if the retrieved docs were relevant.

### Example for multiquery retrieval




# Stage 4: The local graph creation

It is best explained by example the user query is “How did Jakob Bernoulli influenced probability theory” After query decomposition suppose it goes into 2 questions and retrieved docs are as follows “Jakob Bernoulli introduced the Law of Large Numbers in his work Ars Conjectandi.” and “The Law of Large Numbers is a foundational result in probability theory.”. The nodes for the following 2 docs can be while maintaining the references and chunks from metadata

1. Jakob Bernoulli
2. Law of Large Numbers
3. Ars Conjectandi
4. Probability Theory

Using these nodes we can craft the nodes for the documents such as

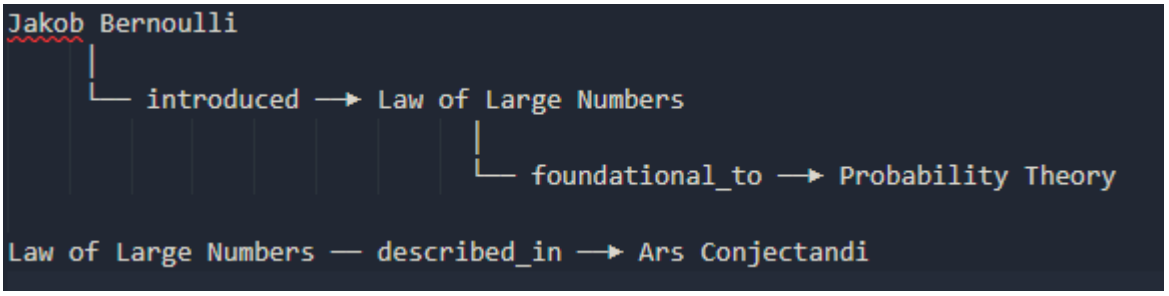
From **Chunk 1**:

- *“Jakob Bernoulli introduced the Law of Large Numbers”*
  → Edge: `introduced`
- *“introduced … in his work Ars Conjectandi”*
  → Edge: `described_in`

From **Chunk 2**:

- *“Law of Large Numbers is a foundational result in probability theory”*
  → Edge: `foundational_to`

`This leads to the graph of form`




Now if the output is of the form “Jakob Bernoulli influenced probability theory by introducing the Law of Large Numbers.” we can justify it using the above graph by saving Jakob introduced the Law which was foundational in Theory. Why this works as it only constructs it from retrieved evidence where the nodes and edges are corresponding to the text and the system can easily explain how the thought of processing worked.

## Stage 5: Answer generation and validation

For the generation from the documents we use Gemini or ChatGPT api where the LLM sees only the evidence stated chunks the KG nodes and edges and the sub queries(from query decomposition) having a constraint to have the sentence supported by at least 1 evidence from the KG. For the validation we match the claims to the evidence from the graph, attach the chuck and document ID for citation while flagging the unsupported claims from the output this whole process ensures faithfulness.

## Stage 6: Document Scoring and Transparency

We can simply map how much a document is related to the output via, helping us with interpretability of the output and which document was used and if the document was helpful or not, In a similar fashion we can say if a document contribution was below a threshold or 0 we can say it was retrieved and never used for the final output.

document contribution = (chunks used from the doc) / (total chunks retrieved from the doc)

## Stage 7: The User Visible output

The user now sees the following components

**Reasoning components**

1. Sub queries (query decomposition)
2. Domains explored (entity extraction using spacy or other tech)

**Evidence**

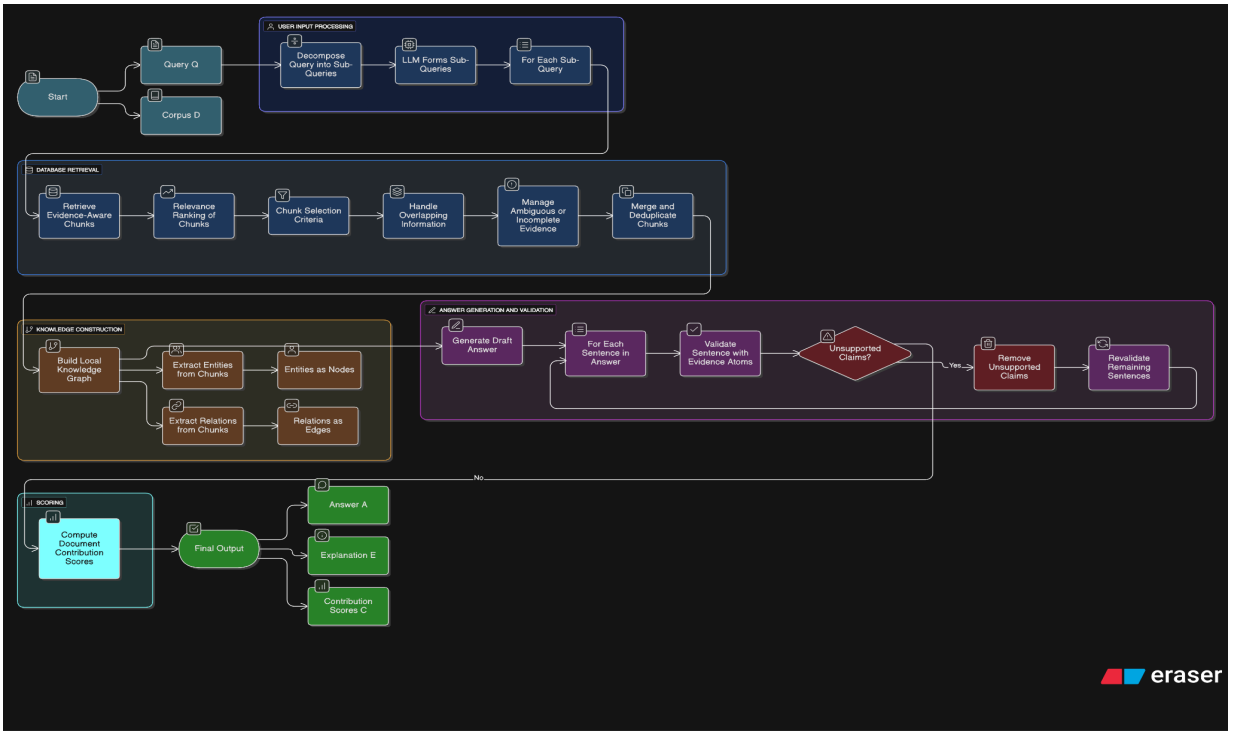
1. Chuck ID
2. Document ID
3. KG relations

**Contribution breakdown**

1. Per document contribution %
2. Unused documents explained

**Final answer**

1. The retrieved ans with a full verification breakdown for the same.




# Research References

1. blog.langchian.com
2. docs.langchain.com
3. https://arxiv.org/abs/2312.10997
4. https://arxiv.org/pdf/2510.18633
5. https://arxiv.org/abs/2201.11903
6.