**Information Indexing and Retrieval :  
Retrieval Augmented Generation on academic sources**

Authors :  
  
BELAYACHI Ibrahim  
NAIT BAHLOUL Rayane  
NTICHA Amen Allah  
OUZZANE Yassine  
TAIBI Hamza

[**I. Introduction 2**](#_iqf3s7hx1gx3)

[**II. How does it work? 2**](#_hye89w340opt)

[1. Document Indexing 2](#_rrhcquh2e2oa)

[2. Search and Answer Generation 2](#_64mg0gx9gvu4)

[**III. Data Collections 3**](#_mmy7dgdud3qp)

[1. Corpus Collection and Curation 3](#_hsq6aa3f2ysw)

[**IV. Document Cleaning and Chunking Process 4**](#_j39s0uxw0u6u)

[1. Document Extraction 4](#_af33sup8faun)

[2. Prompt-Based Text Cleaning 4](#_7urtvykaaks9)

[3. Cleaning Output Integration 4](#_naz4ds11xna0)

# Introduction

Predictive learning analytics has become a critical area of research in education, offering powerful tools to anticipate student outcomes and inform timely pedagogical decisions. By leveraging student data—such as activity logs, assessments, and engagement metrics—researchers have developed models capable of forecasting key academic results, most notably final grades. These models support data-informed practices aimed at improving performance, identifying at-risk students, and personalizing learning environments.

This project seeks to harness these research efforts by assembling a curated dataset of 136 scientific publications, all focused on predictive modeling using student data. The collected studies, sourced from Scopus through a structured search process, were selected based on clear criteria: each must include a predictive model built on student-related data with the objective of forecasting academic outcomes. This dataset represents a rich knowledge base of state-of-the-art predictive analytics in education.

Our overarching goal is to explore how large language models (LLMs)—such as ChatGPT—can interact with these predictive models. Specifically, we aim to assess whether an LLM can understand the structure and intent of these models and, more importantly, generate contextually relevant, evidence-based recommendations for educators, learners, and institutional stakeholders. This aligns with the broader vision of augmenting AI with curated academic knowledge to support actionable, research-grounded insights.

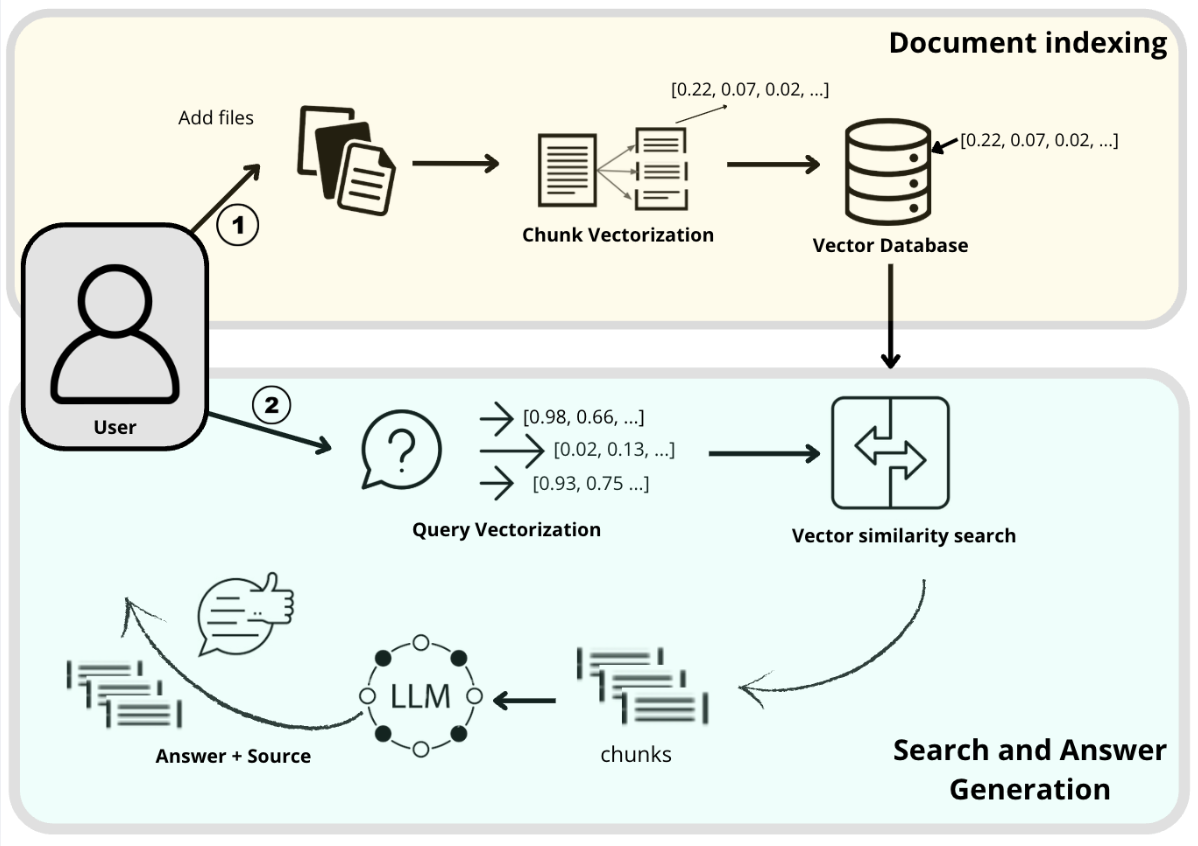
By bridging the gap between empirical research and practical application, this work positions predictive learning analytics not merely as a theoretical tool, but as a conversational, interpretable resource usable within educational ecosystems powered by generative AI.

# How does it work?

## Document Indexing

The RAG process begins with the user uploading local files to the interface (Step 1 in the figure 1). These documents are broken into smaller, manageable pieces called chunks. Each chunk undergoes vectorization, where its content is transformed into a numerical representation (vectors) that captures the semantic meaning of the text. These vectors are stored in a vector database, a specialized storage optimized for quick similarity searches. This step creates the foundation for the RAG system, as it allows the 19system to efficiently compare a user’s query with pre-processed document chunks during the retrieval process.

## Search and Answer Generation

When the user submits a query (Step 2 in the figure 1), the question is first converted into a query vector using the same vectorization method applied to the document chunks. The RAG system then performs a vector similarity search, comparing the query vector with the chunk vectors stored in the database. The chunks are ranked based on relevance scores (ranging from 0 to 1), calculated by measuring the distance between the query vector and the document vectors. The most relevant chunks are sent to the LLM (e.g., Llama 3.2 via Ollama) along with the query. The LLM processes this context to generate a detailed and enriched response. Crucially, the retrieved chunks are included as sources, providing traceability for the user. This ensures that the generated answers are grounded in reliable and verifiable data

**Figure 1**: RAG process

# III. Data Collections

## 1. Corpus Collection and Curation

A total of **136 scientific articles** were collected from the **Scopus** database. These articles were selected based on predefined inclusion criteria:

* The use of **student-related data** (e.g., demographic, behavioral, academic)
* The development of **predictive models** aimed at forecasting educational outcomes such as academic performance, engagement, dropout risk, etc.
* The presence of the keyword *"predict"*\* in the title, abstract, or keywords
* Published as peer-reviewed **journal or conference papers**

This corpus represents a curated body of knowledge at the intersection of machine learning and learning analytics in higher education.

# IV. Document Cleaning and Chunking Process

To prepare the dataset for use in a Retrieval-Augmented Generation (RAG) system, we implemented a systematic cleaning and chunking procedure applied to all PDF documents collected during the literature review phase.

## 1. Document Extraction

Each selected academic article in PDF format was processed individually using the PyPDFLoader module from the LangChain framework. This tool enables efficient extraction of textual content from PDF files, which was then structured into a tabular format. Each page of the document was treated as an independent unit of content, stored with its associated source filename. The resulting raw dataset was saved as a CSV file to facilitate further processing.

## 2. Prompt-Based Text Cleaning

The extracted raw text often contained noise resulting from misinterpreted visual elements such as figures, tables, graphs, and page headers/footers. To address this, we developed a custom prompt-driven cleaning strategy, leveraging the Gemini 2.0 Flash model via Google Generative AI APIs.

The cleaning prompt was specifically designed to:

Remove:

* Garbled output from figures, images, tables, and equations
* Page numbers, headers, footers, and academic citations

Preserve:

* Narrative text explaining visual content
* Section headings and structurally relevant text

Each text chunk (page content) was processed iteratively, with up to 10 retry attempts implemented in case of API errors or quota limitations. Exponential backoff delays were included to ensure stability and continuity of processing in the event of server or rate-limit issues.

## 3. Cleaning Output Integration

The cleaned output for each chunk was saved and matched back to its original page content. A final dataframe was generated containing:

* The source file name
* The original raw text (page-level)
* The cleaned version of the text

This cleaned dataset was exported as a new CSV file and formed the basis for downstream vectorization and indexing in the RAG pipeline.