### ## 1. Core Concepts You Need to Know

To build "Exam-Gen" successfully, you need a solid grasp of these five core concepts.

#### **A. Text Embeddings**

* **What it is**: The process of converting text (words, sentences, or paragraphs) into a list of numbers called a **vector**. This vector captures the semantic meaning of the text.
* **Analogy**: Think of it like a coordinate system for meaning. Words with similar meanings, like "king" and "queen," will have vectors that are numerically close to each other in this "meaning space."
* **Why it's needed**: It allows the computer to understand and compare text based on its meaning, not just keywords. This is the foundation for finding relevant information in your documents.

#### **B. Vector Databases**

* **What it is**: A special type of database designed to store and search through text embeddings efficiently.
* **Analogy**: It's like a library's card catalog, but instead of searching by author or title, you search by giving it an idea (another vector), and it finds the most similar ideas (vectors) on its shelves almost instantly.
* **Why it's needed**: When a teacher wants questions about "Thermodynamics," the vector database can instantly find all the text chunks in the textbook that are semantically related to that topic.

#### **C. Retrieval-Augmented Generation (RAG)**

* **What it is**: The modern architecture that makes your project possible. It's a two-step process to get accurate, context-aware answers from an LLM.
  1. **Retrieve (Find)**: First, you search your vector database to find the most relevant text chunks from the uploaded document.
  2. **Generate (Use)**: Then, you give those relevant chunks to a Large Language Model (LLM) as context and ask it to generate a question *only* based on that information.
* **Why it's needed**: This prevents the LLM from making up facts (hallucinating) and ensures every question is directly based on the source material provided by the teacher.

#### **D. Large Language Models (LLMs) & APIs**

* **What it is**: LLMs (like Google's Gemini or OpenAI's GPT) are the "brains" that will generate the questions. You interact with them through an **API** (Application Programming Interface), which is like a messenger that sends your instructions (your prompt) to the model and brings back its response.
* **Why it's needed**: The LLM does the creative work of phrasing the question, setting the difficulty, and writing the answer key based on the context you provide.

#### **E. Prompt Engineering**

* **What it is**: The skill of writing clear, specific, and structured instructions (prompts) for the LLM to get the exact output you want.
* **Analogy**: It's like being a good manager. If you give vague instructions, you get poor results. If you give precise, step-by-step instructions with examples, you get high-quality work.
* **Why it's needed**: A well-engineered prompt is the key to forcing the LLM to generate questions in the correct format (e.g., JSON), with the right difficulty, and without adding extra conversational text.

### ## 2. Recommended Technology Stack 💻

This stack is modern, beginner-friendly, and perfect for building a functional prototype within a few weeks. All tools are based in **Python**.

| **Category** | **Tool** | **Purpose** |
| --- | --- | --- |
| **Web Framework** | **Streamlit** | The fastest and easiest way to build a user interface for your app (file uploads, forms). |
| **AI/NLP Framework** | **LangChain** or **LlamaIndex** | Simplifies the entire RAG pipeline: loading documents, chunking, and managing LLM calls. |
| **Text Embedding Model** | **Sentence-Transformers** | A free, open-source library to generate high-quality text embeddings locally. |
| **Vector Database** | **ChromaDB** | A very simple, in-memory vector database that is perfect for a capstone project. |
| **Generative LLM** | **Google AI for Developers (Gemini API)** | Provides a powerful generative model with a generous free tier for development. |
| **PDF Handling** | **PyMuPDF** | A fast and reliable library for extracting text from uploaded PDF files. |

### ## 3. Formal Project Problem Statement

You can use this statement for your project proposal and final report.

**Project Title**: Exam-Gen: An AI-Powered Question Paper Generation System using Retrieval-Augmented Generation

1. Introduction

Educators in academic institutions spend a significant amount of time and manual effort in designing and creating examination papers. This process involves ensuring comprehensive syllabus coverage, maintaining a balance of question types (e.g., MCQ, Short Answer), varying difficulty levels, and preparing a corresponding marking scheme. This manual process is often tedious, repetitive, and prone to inconsistency.

2. Objective

The primary objective of this project is to develop an intelligent system, "Exam-Gen," that automates the creation of well-structured question papers and answer keys. The system will leverage modern Natural Language Processing (NLP) techniques to generate high-quality, relevant questions directly from source materials provided by the educator, thereby saving them valuable time and effort.

3. Scope and Features

The proposed system will have the following key features:

* **Material Ingestion**: Allow educators to upload source documents in various formats (e.g., PDF).
* **Dynamic Structure Configuration**: Enable users to define the structure of the exam paper dynamically, specifying the number of questions, marks per question, and desired difficulty level for different sections.
* **Context-Aware Question Generation**: Utilize a Retrieval-Augmented Generation (RAG) architecture to ensure all generated questions are factually grounded in the provided source material.
* **Varied Question Types**: Generate a mix of question formats as specified by the user.
* **Answer Key Generation**: Automatically produce a detailed answer key or marking scheme corresponding to the generated question paper.
* **Export Functionality**: Allow the final question paper and answer key to be downloaded as formatted PDF documents.

4. Methodology

The system will be built using a Python-based stack. The core methodology involves:

1. Extracting text from source documents and splitting it into manageable chunks.
2. Converting text chunks into semantic vector embeddings using a sentence-transformer model.
3. Storing and indexing these embeddings in a ChromaDB vector database.
4. Implementing a RAG pipeline that retrieves relevant context from the database based on user-defined topics.
5. Using a Large Language Model (e.g., Gemini) with carefully engineered prompts to generate questions and answers based on the retrieved context and user-defined constraints.
6. Developing a user-friendly web interface with Streamlit for interaction.

5. Expected Outcome

The final deliverable will be a functional web application that serves as a proof-of-concept for "Exam-Gen." It will successfully demonstrate the end-to-end pipeline of uploading a document, specifying exam parameters, and generating a complete, formatted question paper and answer key ready for use.