Fase 2: Implementación de Componentes Principales: El Cerebro Operativo del Tutor

En esta fase, construiremos los módulos centrales del sistema RAG: el recuperador (que encuentra la información relevante) y el generador (que crea la respuesta del tutor).

2.1. Diseño del Módulo Recuperador

El recuperador es la "biblioteca" de nuestro tutor. Su eficiencia y precisión determinarán qué tan bien el sistema encuentra el conocimiento correcto para responder a las preguntas de los estudiantes.

2.1.1. Indexación:

Aquí transformamos nuestra base de conocimiento preparada (textos y KGs) en un formato que la máquina puede buscar eficientemente.

Segmentación de Documentos (Chunking):

- *Herramientas/Librerías:* Utilizar librerías de procesamiento de texto en Python como NLTK, SpaCy, o LangChain (para su módulo de text splitting) para dividir los documentos.
- *Estrategias por Asignatura:*
  - **Matemática:** Segmentación por concepto (ej., "Teorema de Pitágoras"), por tipo de problema (ej., "Ecuaciones Cuadráticas"), por explicación de un paso a paso de un ejercicio.
  - **Español-Literatura:** Segmentación por figura retórica, por análisis de una obra específica, por regla gramatical, por biografía de autor, por fragmentos clave de textos literarios.
  - **Historia:** Segmentación por evento (ej., "Guerra de los Diez Años"), por personaje histórico, por ley o decreto, por concepto historiográfico (ej., "clase social", "revolución").
- *Superposición (Overlap):* Implementar una pequeña superposición entre chunks adyacentes para asegurar que el contexto no se pierda en los límites de los fragmentos.
- *Metadatos de Chunk:* Asegurar que cada chunk se almacena con sus metadatos asociados (asignatura, tema, subtema, tipo de contenido, dificultad, fuente original) previamente definidos en la Fase 1.

Modelos de Embedding (Vectorización):

- *Selección del Modelo:*
  - Modelos de propósito general en español: Evaluar modelos de sentence-transformers que estén pre-entrenados para español (ej., paraphrase-multilingual-MiniLM-L12-v2, distiluse-base-

multilingual-cased-v2).
    - Fine-tuning con datos académicos (Opcional pero Recomendado): Considerar el fine-tuning de un modelo de embedding base utilizando un corpus pequeño de preguntas/respuestas o textos académicos cubanos para adaptarlo mejor a la terminología específica y mejorar la relevancia semántica para el dominio de los exámenes de ingreso.
- *Implementación:* Usar la librería sentence_transformers para generar los vectores de alta dimensión de cada chunk.

Base de Datos Vectorial:

- *Elección de la DB:* Seleccionar una base de datos vectorial adecuada para almacenar eficientemente los embeddings y permitir búsquedas de Vecinos Más Cercanos Aproximados (ANN). Opciones populares incluyen:
    - FAISS (Facebook AI Similarity Search): Si se gestiona localmente y el volumen de datos es manejable, es muy eficiente para búsqueda ANN.
    - Pinecone, Weaviate, Milvus, ChromaDB: Soluciones gestionadas o auto-hospedadas que ofrecen escalabilidad, filtrado de metadatos y otras características empresariales. Para un proyecto inicial, ChromaDB o FAISS pueden ser buenas opciones.
- *Ingesta:* Desarrollar scripts para cargar los chunks vectorizados y sus metadatos en la base de datos vectorial.

Crawler Automatizado - Fase Inicial (Implementación):

- *Framework de Crawling:* Utilizar librerías como Scrapy o BeautifulSoup (con requests) en Python para la recolección de datos de sitios web académicos o repositorios digitales accesibles.
- *Procesamiento de Documentos:* Integrar librerías para la lectura de diferentes formatos de documentos: PyPDF2 o pdfminer.six para PDFs, python-docx para Word, markdown para Markdown.
- *OCR Integración:* Si la fuente principal son libros físicos digitalizados, integrar el proceso de OCR (ej., con pytesseract si se usa Tesseract OCR) directamente en el pipeline de ingesta para transformar imágenes en texto.
- *Pipeline de Indexación:* El crawler orquestará la extracción, limpieza, segmentación, vectorización y almacenamiento de los datos en la base de datos vectorial.

2.1.2. Mecanismo de Recuperación:

Este componente toma la consulta del estudiante y la utiliza para buscar los chunks más relevantes.

Tipos de Recuperación:

- *Recuperación Densa (Predominante):* La consulta del estudiante se vectoriza usando el mismo modelo de embedding que se usó para los chunks. Luego, se realiza una búsqueda de similitud de coseno (o producto punto) en la base de datos vectorial para encontrar los k chunks más cercanos.

- *Recuperación Híbrida (BM25 + Densas):* Opcionalmente, combinar los resultados de una búsqueda de palabras clave (ej., usando BM25 con pyserini o Whoosh) con la búsqueda densa. Se pueden usar técnicas de fusión como RRF (Reciprocal Rank Fusion) para combinar los rankings.
- *Recuperación Basada en Grafo de Conocimiento (KG):*
    - Consulta SPARQL/Cypher: Para consultas que requieran razonamiento relacional (ej., "autores del Modernismo cubano"), se diseñarán templates de consultas a la base de datos de grafos (ej., Cypher para Neo4j, SPARQL para Blazegraph/RDF) para extraer subgrafos o relaciones específicas.
    - Integración de Resultados: Los resultados del KG (ej., una lista de autores y sus obras) pueden ser insertados en el prompt del LLM junto con los chunks recuperados de la base vectorial.

Implementación de Búsqueda Eficiente (ANN): La base de datos vectorial se encarga de la búsqueda ANN. Es fundamental configurar los parámetros de búsqueda (ej., n_probe en FAISS) para equilibrar velocidad y precisión.

Filtrado por Metadatos: Permitir filtrar la búsqueda de chunks basada en metadatos. Si un estudiante especifica "Matemática" o "Ejercicios de Historia", la búsqueda vectorial solo se realizará sobre los chunks con esos metadatos.

2.1.3. Procesamiento/Optimización de Consultas:

Mejorar la pregunta del estudiante antes de que sea procesada por el recuperador.

Análisis de Intención y Entidades (NLP Avanzado):

- *Librerías:* Utilizar SpaCy, NLTK, o un modelo pre-entrenado de Hugging Face para NER y clasificación de intención.
- *Detección de Intención:* Clasificar la pregunta del estudiante en categorías como definición, explicación, ejercicio, corrección, solución, ejemplo, biografía, análisis.
- *Extracción de Entidades:* Identificar automáticamente los conceptos clave, temas, personajes, fechas, asignaturas mencionados en la consulta del estudiante (ej., "Teorema de Pitágoras", "Modernismo", "Guerra de Independencia").
- *Asignación de Asignatura:* Si la consulta no especifica la asignatura, inferirla automáticamente a partir de las palabras clave o entidades detectadas.

Reescritura y Expansión de Consultas:

- *Reescritura:* Si la consulta es ambigua o demasiado corta, el sistema puede reescribirla para hacerla más específica (ej., "función" -> "función matemática lineal").
- *Expansión:* Añadir sinónimos o términos relacionados a la consulta original para ampliar el alcance de la recuperación, usando un tesauro o embeddings para encontrar términos semánticamente cercanos.

- *Descomposición de Consultas Complejas:* Para preguntas como "¿Explícame el teorema de Pitágoras y dame un ejercicio?", el sistema puede usar reglas heurísticas o un LLM más pequeño para descomponerla en dos sub-consultas y procesarlas secuencialmente.

Integración con KG para Procesamiento de Consultas: Si la consulta del estudiante contiene entidades que están en el KG, podemos usar el KG para expandir la consulta con relaciones relevantes antes de la vectorización para el recuperador.

2.1.4. Selección del Modelo de Embedding:

- *Confirmar Modelo:* Basados en pruebas iniciales y la disponibilidad de modelos pre-entrenados en español para contextos académicos, confirmaremos el modelo de embedding final.
- *Estrategia de Fine-tuning (si aplica):* Definir el dataset y el proceso para el fine-tuning del modelo de embedding, incluyendo métricas para evaluar su mejora.

2.2. Diseño del Módulo Generador

El generador es la "voz" de nuestro tutor. Toma la información recuperada y la transforma en una respuesta didáctica y comprensible.

2.2.1. Seleccionar el Modelo de Lenguaje (LLM):

Criterios de Selección:

- *Rendimiento en Español:* Debe tener una alta capacidad para entender y generar texto fluido y preciso en español.
- *Capacidad de Razonamiento:* Importante para Matemática e Historia, donde se requiere inferencia y explicación de procesos.
- *Tamaño y Costo:* Balancear el tamaño del modelo (que influye en la calidad) con los recursos de cómputo disponibles (GPU, RAM) y los costos de inferencia si se usa una API.
- *Disponibilidad y Licencia:* Considerar modelos de código abierto (ej., Llama 3, Mistral, Gemma, adaptaciones de BERT/T5 para generación) o APIs comerciales (OpenAI GPT, Gemini, Claude) si se ajustan al presupuesto y la estrategia de despliegue. Para Cuba, la disponibilidad y acceso a APIs internacionales puede ser una consideración importante, lo que podría inclinar la balanza hacia modelos on-premise o de código abierto.

Estrategia de Fine-tuning (Opcional pero Altamente Recomendado):

- *LoRA (Low-rank Adaptation):* Utilizar técnicas de fine-tuning eficiente de parámetros como LoRA para adaptar el LLM al tono pedagógico, al estilo de explicación (ej., "paso a paso" para

matemáticas), y a los formatos específicos de respuesta (ej., cómo presentar una solución de ejercicio, cómo analizar un fragmento literario).

- *Dataset para Fine-tuning:* Crear un dataset de pares de (pregunta + contexto recuperado, respuesta ideal del tutor) basándose en los materiales académicos cubanos y respuestas de expertos humanos.

2.2.2. Implementar el Proceso de Generación:

Esta es la orquestación de la entrada al LLM y la post-procesamiento de su salida.

Construcción del Prompt (Ingeniería de Prompts): El prompt que se envía al LLM es crítico. Deberá incluir:

- *Instrucciones Claras:* Rol del LLM (tutor inteligente, experto en la asignatura X), tono (pedagógico, paciente), formato de salida (paso a paso, conciso, con ejemplos).
- *Consulta Original del Estudiante:* La pregunta tal cual la formuló.
- *Contexto Recuperado:* Los chunks más relevantes obtenidos del módulo recuperador (y, si aplica, la información del KG).
- *Instrucciones de Fusión:* Indicar al LLM cómo usar el contexto. Ej., "Usa solo la información proporcionada en el contexto para responder", "Si la respuesta no está en el contexto, indica que no tienes esa información".
- *Few-shot Examples (Opcional):* Incluir algunos pares de (consulta + contexto, respuesta ideal) para guiar al LLM en el formato y estilo deseado.

Manejo del Context Window del LLM: Asegurar que la longitud combinada del prompt y el contexto recuperado no exceda el límite de tokens del LLM. Se pueden usar técnicas de compresión de contexto o selección inteligente de chunks para manejar esto.

Post-procesamiento de la Respuesta Generada:

- *Formato:* Dar formato a la respuesta (Markdown para negritas, listas, bloques de código para ecuaciones).
- *Verificación Básica:* Implementar verificaciones de sanidad, como asegurar que no haya alucinaciones obvias (aunque el RAG reduce esto, no lo elimina por completo).
- *Manejo de Respuestas "No Sé":* Si el LLM indica que no tiene información relevante, o si la confianza en la recuperación es baja, el tutor debe poder decir "No tengo información suficiente sobre esto" en lugar de inventar.
- *Manejo de Errores de Generación:* Implementar reintentos o mensajes de error amigables si el LLM falla en generar una respuesta coherente.

Interacción Específica por Asignatura:

- *Matemática:* Integrar librerías para renderizar ecuaciones (ej., MathJax si es un front-end web) o para verificar la validez de expresiones matemáticas.
- *Español-Literatura:* Enfocarse en la gramática, ortografía y el estilo en la generación de respuestas.
- *Historia:* Priorizar la precisión de fechas y nombres, y la conexión causa-efecto.

Esta fase es donde la mayor parte del "código" se escribirá, conectando las bases de datos, los modelos de NLP, los embeddings y los LLMs para crear un sistema funcional.