# Assignment: Build a Multi-Threaded Text Retrieval and Processing System with Web Data

## Objective:

Design and implement a lightweight, multi-threaded system in Python that simulates a simplified Retrieval-Augmented Generation (RAG) pipeline. The system should extract and process data from the Wikipedia page on Artificial Intelligence ("<https://en.wikipedia.org/wiki/Artificial_intelligence>"), create embeddings, retrieve relevant content based on a query, and process it efficiently using foundational programming concepts.

**You are free to use generative AI tools (e.g., ChatGPT, GitHub Copilot) to assist, but your submission must demonstrate a strong grasp of core programming principles.**

## Requirements:

1. **Input:**
   * Corpus: Extract text data from the Wikipedia page "<https://en.wikipedia.org/wiki/Artificial_intelligence>.
   * Sample query string (e.g., "What is the impact of AI?").
2. **Tasks:**
   * **Data Extraction and Cleaning (OOPS):**
     + Write a class-based module to scrape or load the Wikipedia page content (e.g., using requests and BeautifulSoup for scraping, or parsing the provided document).
     + Clean the extracted text by removing HTML tags, references (e.g., "[1]"), and irrelevant sections (e.g., footnotes, external links). Split the content into manageable chunks (e.g., paragraphs or sections).
   * **Embedding Creation (Multiprocessing):**
     + Use Python’s multiprocessing module to compute embeddings for each chunk in parallel. You can use a simple word-embedding approach (e.g., averaging word vectors from a library like gensim with pre-trained Word2Vec) or a lightweight sentence embedding method (e.g., sentence-transformers).
     + Store embeddings in a structured format (e.g., a dictionary or NumPy array).
   * **Document Retrieval (Threading):**
     + Implement a multi-threaded retrieval module using Python’s threading module to compute cosine similarity (or another metric) between the query embedding and chunk embeddings.
     + Retrieve the top 3 most relevant chunks based on similarity scores, parallelizing the computation across threads.
   * **Text Processing (Async Programming):**
     + Write an asynchronous function using asyncio to preprocess the retrieved chunks (e.g., tokenization, removing stopwords) concurrently.
     + Return the processed text for each chunk.
   * **Linux Integration:**
     + Include a Bash script to set up the environment (e.g., install dependencies via pip), run the program, and log the output to a file in a Linux-compatible format.
3. **Output:**
   * A ranked list of the top 3 chunks with their relevance scores, printed to the console and saved to a log file.
   * A brief explanation (in comments or a README) of how threading, async programming, multiprocessing, and embeddings were used, and if/how Gen AI tools were leveraged.
4. **Constraints:**
   * No high-level LLM APIs (e.g., Hugging Face Transformers, LangChain) for core retrieval/processing logic—focus on custom implementation. Gen AI tools may be used for code assistance or optimization only.
   * Code must follow OOPS principles (e.g., classes for extraction, embedding, retrieval, and processing).
   * Must be executable on a Linux environment.

**Deliverables:**

* Source code (Python + Bash script).
* A README with setup instructions, a short explanation of the approach, and a note on how (if at all) Gen AI tools were used.
* Example output based on the query "What is the impact of AI?"

**Evaluation Criteria:**

* Correctness and efficiency of multi-threading, async programming, and multiprocessing implementations.
* Proper use of OOPS concepts (e.g., encapsulation, modularity).
* Quality of data extraction, cleaning, and embedding creation.
* Accuracy of retrieval (relevance of returned chunks).
* Code readability, structure, and documentation.
* Ability to handle edge cases (e.g., failed web requests, empty chunks).
* Linux compatibility and script functionality.
* Effective and appropriate use of Gen AI tools (if used), with evidence of personal understanding and customization beyond tool-generated code.

**Note on Gen AI Tools:**  
You are free to use generative AI tools to assist with brainstorming, debugging, or optimizing your code. However, your submission must reflect a deep understanding of the underlying concepts (OOPS, threading, async, multiprocessing, embeddings) and not simply be a copy-paste of tool-generated output. Please document where and how you used these tools in your README.