**Use Case**: Develop an AI-Powered Knowledge Base Assistant (PoC) utilizing organization Internal Document(s)

**Problem Statement**: At Daythree, a wide range of valuable information is stored across various internal documents and policies. Accessing the right information quickly becomes increasingly challenging when the volume grows. To improve efficiency and empower internal agents with faster, precise and accurate access to relevant knowledge, you are tasked with building an AI solution that leverages these documents to deliver intelligent, context-aware responses through a proof-of-concept (PoC) assistant.

**Data provided**: 4 PDF documents

1. celcomdigi-eratkanikatan-sahur-moreh-pass.pdf
2. celcomdigi\_raya\_video\_internet\_pass.pdf
3. celcomdigi\_samsung\_galaxy\_s25\_series\_launch.pdf
4. celcomdigi\_port-in-rebate-offer.pdf

**Area of assessment**:

1. Data Processing & Ingestion: How would the data being processed and ingested for consumption?
2. Retrieval Pipeline: How can the processed data be retrieved efficiently? What might be the suggested approach?
3. Prompt Engineering & LLM Utilization: What is the prompting method & type of model utilized? Are they efficient/modular/scalable?
4. Evaluation & Feedback Loop: How can we evaluate this AI solution that we created? Any example metrics to be looked at?
5. Presentation and/or UI/UX: Presentation on findings for solution built, utilization of simple and effective UI presentation can be a bonus. Example of presentation but not limited to below:
   1. Walk through your thought process for solution creation, design choices and/or architecture if any. Ability to explain reasoning, trade-offs, and limitations
   2. Justify your selection of tools, libraries, models, and UI approach.
   3. Showcase on how the solution can solve the business problem ie. multi-document scenarios, ambiguous questions and/or other areas of relevant interest

**Submission**: Submission of relevant documents (Python Notebook/Github Link) through email and any other form of presentation that might be helpful (eg. PPT slide if necessary) – Zip your files (if with attachments) and rename them to Daythree\_AI\_UseCase\_{Name}.zip

**Answer:**

**Question 1**

* **HTTP Ingest Endpoint**
  + POST /api/ingest accepts one or more PDFs.
  + Files are streamed to ingestion.ingest\_files().
* **Text Extraction**
  + Primary: LangChain’s PyPDFLoader (uses unstructured under the hood).
  + Fallback: EasyOCR when text loader fails (scanned pages).
* **Chunking Strategy**
  + Pages split into ~1,000-character chunks via CharacterTextSplitter.
  + Balances retrieval granularity against prompt size limits.
* **Embedding & Storage**
  + Chunks embedded with Ollama’s granite-embedding:278m.
  + Vectors upserted into a persistent ChromaDB store on disk.
* **Trade-offs**
  + **Disk persistence** ensures index survives restarts;
  + **In-memory mode** is faster but lost on crash.

**Question 2**

**2. Retrieval Pipeline**

* **Retriever Initialization**
  + retriever.get\_retriever() loads Chroma with OllamaEmbeddings.
  + Configured as a .as\_retriever(k=4) for top-4 hits.
* **Semantic Search**
  + Queries are embedded with the same embedding model.
  + Chroma performs a fast k-NN search (sub-10 ms per query).
* **RAG Chain**
  + Wrapped in LangChain’s RetrievalQA.from\_chain\_type(chain\_type="stuff").
  + Injects retrieved chunks + user prompt into the LLM.
* **Efficiency Recommendations**
  + Use approximate nearest-neighbor indexes (HNSW, Faiss) for millions of vectors.
  + Tune k based on precision/recall balance.
* **Question 3  
  Model Selection**
  + Frontend dropdown keys (llama3, gemma, deepseek-r1) mapped in MODEL\_ALIASES.
  + Swap or add models by updating the alias map—no chain changes required.
* **System Prompt**
  + Enforces no hallucination, conciseness, friendly tone, and context-only answers.
  + Polite clarifying questions when the user’s query is ambiguous.
* **QA Template**
  + **```**
  + **<SYSTEM\_PROMPT>**
  + **Context:**
  + **{context}**
  + **User Question:**
  + **{question}**
  + **Assistant Answer:**
  + **```**

**Question 4**

**Evaluation & Feedback Loop (Future Work)**To fully measure and iterate on our system we propose:

1. Retrieval Recall@k – compare the top-k retrieved chunks against a small, hand-labeled test set to quantify semantic recall.
2. LLM Latency – wrap each qa\_chain.run() call in a simple timer to record end-to-end response time.
3. Answer Accuracy – add a thumb-up/thumbs-down or star rating widget in the UI and POST ratings to a new /api/rate endpoint, storing each rating alongside the question/answer.
4. Satisfaction Score – similarly capture a 1–5 satisfaction rating after each response in the chat interface.

**Current Status**: The PoC already stores all questions and answers in memory (session\_logs) but doesn’t yet compute any of these metrics. Instrumenting timers around the LLM call, adding a small evaluation suite for Recall@k, and extending the frontend/back-end to collect user ratings would complete the loop.