Neccessary Libraries to install before staring code

In [2]:
```
import os
```

In [3]:
```
from pathlib import Path
```

In [4]:
```
import hashlib
```

In [5]:
```
from langchain.document_loaders import PyMuPDFLoader
```

In [6]:
```
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

In [7]:
```
import pytesseract
from PIL import Image
```

In [8]:
```
from langchain_core.documents import Document
```

importing Pdf's Folder into environment

In [10]:
```
RAW_PDF_DIR=Path(r"C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\Training Data")
```

creating chunks folder, if folder already exists it ignores, if not it creates a folder to store chunks

In [12]:
```
CHUNK_DIR=Path(r"C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\chunks")
CHUNK_DIR.mkdir(parents=True, exist_ok=True)
```

In [13]:
```
import json

HASH_CACHE_FILE = CHUNK_DIR / "file_hashes.json"

# Load existing cache (if any)
if HASH_CACHE_FILE.exists():
    with open(HASH_CACHE_FILE, "r") as f:
        hash_cache = json.load(f)
else:
    hash_cache = {}
```

In [14]:
```
# ⬇️ Load previously saved pdf chunks
CHUNK_FILE = CHUNK_DIR / "pdf_chunks.json"
if CHUNK_FILE.exists():
    with open(CHUNK_FILE, "r", encoding="utf-8") as f:
        json_data = json.load(f)
    all_chunks = [
        Document(page_content=d["page_content"], metadata=d["metadata"])
        for d in json_data
    ]
else:
    all_chunks = []

# ⬇️ Load previously saved image chunks
IMAGE_CHUNK_FILE = CHUNK_DIR / "image_chunks.json"
if IMAGE_CHUNK_FILE.exists():
    with open(IMAGE_CHUNK_FILE, "r", encoding="utf-8") as f:
        image_data = json.load(f)
    image_documents = [
        Document(page_content=d["page_content"], metadata=d["metadata"])
        for d in image_data
    ]
else:
    image_documents = []
```

In [15]:
```
print(RAW_PDF_DIR)
```

```
C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\Training Data
```

In [16]:
```
print(CHUNK_DIR)
```

```
C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\chunks
```

Defining Text Splitter Parameters

chunk size is charectors/words, seperators is spaces, paragraph endings, etx, overlap means relativity of this chunk to other chunks

Defining a function for Loading pdfs into model and cutting into chunks using text_splitter

In [20]:
```python
def ocr_image(image: Image.Image) -> str:
    # Run OCR on image and clean up result
    text = pytesseract.image_to_string(image)
    text = text.strip()
    if not text:
        return "[No OCR text found]"
    return text
```

calling the function and creating chunks from pdfs and printing number of chunks created

Summerizing Images

In [23]:
```python
# Image summary storage
IMAGE_DIR = Path(r"C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\image_chunks")
IMAGE_DIR.mkdir(parents=True, exist_ok=True)
```

In [24]:
```python
import fitz  # PyMuPDF
from PIL import Image
import io

def extract_images_from_pdf(pdf_path, output_folder):
    doc = fitz.open(pdf_path)
    image_info_list = []

    for i in range(len(doc)):
        page = doc[i]
        images = page.get_images(full=True)
        for img_index, img in enumerate(images):
            xref = img[0]
            base_image = doc.extract_image(xref)
            image_bytes = base_image["image"]
            image_ext = base_image["ext"]
            image = Image.open(io.BytesIO(image_bytes)).convert("RGB")

            # ✅ Filter 1: Ignore very small images
            if image.width < 100 or image.height < 100:
                continue

            # ✅ Filter 2: Ignore pure black or near-black images
            avg_pixel = sum(image.convert("L").getdata()) / (image.width * image.height)
```

```
            if avg_pixel < 10:
                continue

            # ✅ Filter 3: Weird aspect ratio
            aspect_ratio = image.width / image.height
            if aspect_ratio < 0.2 or aspect_ratio > 5:
                continue

            filename = f"{Path(pdf_path).stem}_page{i+1}_{img_index}.{image_ext}"
            filepath = output_folder / filename
            image.save(filepath)

            image_info_list.append({
                "file": filename,
                "page": i + 1,
                "source": Path(pdf_path).name,
                "hash": hashlib.md5(image.tobytes()).hexdigest(),  # consistent hash
                "image_obj": image  # ✅ store preloaded image
            })

    doc.close()
    return image_info_list
```

In [25]:

```
import fitz  # you already have this

def extract_surrounding_text(pdf_path: Path, page_num: int, window: int = 1) -> str:
    """
    Grab text from page_num ± window pages (handles boundaries).
    """
    doc = fitz.open(pdf_path)
    total = len(doc)
    start = max(0, page_num - 1 - window)
    end   = min(total, page_num + window)

    parts = []
    for i in range(start, end):
        txt = doc[i].get_text().strip()
        if txt:
            parts.append(f"[Page {i+1}]\n{txt}")
    doc.close()
    return "\n\n".join(parts)
```

In [26]:
```
import torch
print(torch.cuda.is_available())  # Should return True
```

```
True
```

In [27]:
```
# Load BLIP-2 as a pipeline (auto-handles device placement)
from transformers import pipeline
from PIL import Image
captioner = pipeline("image-to-text", model="Salesforce/blip2-opt-2.7b", device_map="cpu")

# Function to summarize one image
def summarize_image_blip2(image: Image.Image):
    image = image.resize((512, 512))
    result = captioner(image)
    return result[0]['generated_text'].strip()
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `use_fast=True` w
ill be the default behavior in v4.52, even if the model was saved with a slow processor. This will result in minor di
fferences in outputs. You'll still be able to use a slow processor with `use_fast=False`.
Device set to use cpu
```

In [28]:
```
def load_and_chunk_pdf(pdf_path):
    loader = PyMuPDFLoader(str(pdf_path))
    pages = loader.load()
    chunks = text_splitter.split_documents(pages)
    return chunks
```

In [29]:
```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,      # or 1200 based on tuning
    chunk_overlap=100,
    separators=["\n\n", "\n", " ", ""]
)
```

In [30]:
```
import hashlib

def hash_file(file_path):
    with open(file_path, "rb") as f:
        return hashlib.md5(f.read()).hexdigest()
```

In [31]:
```python
from concurrent.futures import ThreadPoolExecutor
from tqdm import tqdm
from langchain_core.documents import Document  # Required for vector embedding
import pytesseract
import sys

# ✅ Helper: Hash image to detect duplicates
def hash_image(image_path):
    with open(image_path, "rb") as f:
        image_bytes = f.read()
    return hashlib.md5(image_bytes).hexdigest()

# ✅ Helper: Extract surrounding text from ±1 pages
def extract_surrounding_text(pdf_path: Path, page_num: int, window: int = 1) -> str:
    doc = fitz.open(pdf_path)
    total = len(doc)
    start = max(0, page_num - 1 - window)
    end = min(total, page_num + window)

    parts = []
    for i in range(start, end):
        txt = doc[i].get_text().strip()
        if txt:
            parts.append(f"[Page {i+1}]\n{txt}")
    doc.close()
    return "\n\n".join(parts)

# ✅ Helper: Run OCR on image
def ocr_image(image):
    text = pytesseract.image_to_string(image).strip()
    if not text:
        return "[No OCR text found]"
    return text

# ✅ Helper: summarize + build Document (with filters)
def summarize_and_build(img):
    image = img["image_obj"]
    pdf_path = RAW_PDF_DIR / img["source"]

    try:
        summary = summarize_image_blip2(image)
```

```
    except Exception as e:
        summary = f"[Error summarizing image: {e}]"

    summary_lower = summary.lower()
    meaningless_keywords = ["logo", "pattern", "design", "icon", "symbol", "background", "shape", "color"]
    if any(kw in summary_lower for kw in meaningless_keywords):
        return None

    grayscale = image.convert("L")
    pixels = list(grayscale.getdata())
    mean = sum(pixels) / len(pixels)
    stddev = (sum((p - mean) ** 2 for p in pixels) / len(pixels)) ** 0.5
    if stddev < 5:
        return None

    try:
        ocr_text = ocr_image(image)
    except Exception as e:
        ocr_text = f"[Error running OCR: {e}]"

    nearby_text = extract_surrounding_text(pdf_path, img["page"], window=1)

    combined_context = f"""[Image Summary]
{summary}

[Image OCR Text]
{ocr_text}

[Nearby Text]
{nearby_text}"""

    return Document(
        page_content=combined_context,
        metadata={
            "source": img["source"],
            "page": img["page"],
            "type": "image",
            "image_file": img["file"],
            "hash": img["hash"]
        }
    )
```

```
# ------------------------------------------------------------
# NEW incremental-processing loop (text + images + hashing)
# ------------------------------------------------------------
new_chunks = []
new_image_documents = []

for pdf_file in tqdm(list(RAW_PDF_DIR.glob("*.pdf")), desc="📄 Processing PDFs"):
    file_hash = hash_file(pdf_file)

    if hash_cache.get(pdf_file.name) == file_hash:
        print(f"⏭️ Skipping unchanged PDF: {pdf_file.name}")
        continue

    print(f"📄 Processing new PDF: {pdf_file.name}")

    # ---- 1. text chunks ----------------------------------------------
    chunks = load_and_chunk_pdf(pdf_file)
    for chunk in chunks:
        chunk.metadata["source"] = pdf_file.name
    new_chunks.extend(chunks)

    # ---- 2. image extraction & summarisation -------------------------
    images = extract_images_from_pdf(pdf_file, IMAGE_DIR)
    unique_images = []

    # prepare a set of existing hashes to avoid duplicates
    existing_hashes = {doc.metadata.get("hash")
                       for doc in image_documents + new_image_documents}

    for img in images:
        img_path = IMAGE_DIR / img["file"]
        try:
            img_hash = hash_image(img_path)
        except Exception:
            continue

        if img_hash in existing_hashes:
            img_path.unlink(missing_ok=True)
            continue

        img["hash"] = img_hash
        existing_hashes.add(img_hash)
```

```
        unique_images.append(img)

    with ThreadPoolExecutor(max_workers=4) as ex:
        futures = list(ex.map(summarize_and_build, unique_images))
        for doc in futures:
            if doc:
                new_image_documents.append(doc)

    # save hash so we never re-process this PDF
    hash_cache[pdf_file.name] = file_hash

# ------------------------------------------------------
# after loop finishes, we still have:
#   new_chunks, new_image_documents  -> only the fresh stuff
# ------------------------------------------------------
```

```
Processing PDFs: 100%|██████████| 27/27 [00:00<00:00, 272.09it/s]
```

```
⏭️ Skipping unchanged PDF: Descriptive Analytics - LinkedIn Post- Final.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Advanced.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Data Collection Nodes.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - History System.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Overview.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Security.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - System Management.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Advanced Analytics.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Advanced Options.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Android App.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Calculation Module.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Office Add-Ins.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Overview.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Smart Trend.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Views & Reder Options.pdf
⏭️ Skipping unchanged PDF: InSis Infoview Excel Addin Installation Manual_Rev3.pdf
⏭️ Skipping unchanged PDF: inSis Infoview Scheduler.pdf
⏭️ Skipping unchanged PDF: inSis Mobile App IOS.pdf
⏭️ Skipping unchanged PDF: inSis Platform - Fundamentals.pdf
⏭️ Skipping unchanged PDF: inSis Python Package.pdf
⏭️ Skipping unchanged PDF: inSis Suite -  Prosense.pdf
⏭️ Skipping unchanged PDF: inSis Suite - Dashboards.pdf
⏭️ Skipping unchanged PDF: inSis Suite - Personalised Home.pdf
⏭️ Skipping unchanged PDF: inSis Suite - PFD.pdf
⏭️ Skipping unchanged PDF: inSis Suite - Reports.pdf
⏭️ Skipping unchanged PDF: inSis Suite - SolutionSpace.pdf
⏭️ Skipping unchanged PDF: inSis Suite Installation Manual_v1.6.0.6_14Jul2023 (1).pdf
```

In [32]:

```python
# ✅ Append new chunks and save
all_chunks.extend(new_chunks)
image_documents.extend(new_image_documents)

# Save PDF chunks
with open(CHUNK_FILE, "w", encoding="utf-8") as f:
    json.dump([
        {"page_content": d.page_content, "metadata": d.metadata}
        for d in all_chunks
    ], f, ensure_ascii=False, indent=2)

# Save image chunks
with open(IMAGE_CHUNK_FILE, "w", encoding="utf-8") as f:
    json.dump([
```

```
        {"page_content": d.page_content, "metadata": d.metadata}
        for d in image_documents
    ], f, ensure_ascii=False, indent=2)

# Save hash cache
with open(HASH_CACHE_FILE, "w") as f:
    json.dump(hash_cache, f, indent=2)
```

In [33]:

```
all_chunks = []

for pdf_file in RAW_PDF_DIR.glob("*.pdf"):
    file_hash = hash_file(pdf_file)

    if hash_cache.get(pdf_file.name) == file_hash:
        print(f"⏭️ Skipping unchanged PDF: {pdf_file.name}")
        continue  # Skip reprocessing

    print(f"📄 Processing: {pdf_file.name}")
    chunks = load_and_chunk_pdf(pdf_file)
    for chunk in chunks:
        chunk.metadata["source"] = pdf_file.name
    all_chunks.extend(chunks)

    # ✅ Store new hash
    hash_cache[pdf_file.name] = file_hash
```

```
⏭️ Skipping unchanged PDF: Descriptive Analytics - LinkedIn Post- Final.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Advanced.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Data Collection Nodes.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - History System.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Overview.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - Security.pdf
⏭️ Skipping unchanged PDF: inSis Administrator - System Management.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Advanced Analytics.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Advanced Options.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Android App.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Calculation Module.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Office Add-Ins.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Overview.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Smart Trend.pdf
⏭️ Skipping unchanged PDF: inSis Data Analytics - Views & Reder Options.pdf
⏭️ Skipping unchanged PDF: InSis Infoview Excel Addin Installation Manual_Rev3.pdf
⏭️ Skipping unchanged PDF: inSis Infoview Scheduler.pdf
⏭️ Skipping unchanged PDF: inSis Mobile App IOS.pdf
⏭️ Skipping unchanged PDF: inSis Platform - Fundamentals.pdf
⏭️ Skipping unchanged PDF: inSis Python Package.pdf
⏭️ Skipping unchanged PDF: inSis Suite -  Prosense.pdf
⏭️ Skipping unchanged PDF: inSis Suite - Dashboards.pdf
⏭️ Skipping unchanged PDF: inSis Suite - Personalised Home.pdf
⏭️ Skipping unchanged PDF: inSis Suite - PFD.pdf
⏭️ Skipping unchanged PDF: inSis Suite - Reports.pdf
⏭️ Skipping unchanged PDF: inSis Suite - SolutionSpace.pdf
⏭️ Skipping unchanged PDF: inSis Suite Installation Manual_v1.6.0.6_14Jul2023 (1).pdf
```

converting chunks into json type and storing that in.json format file in chunks folder

In [35]:
```python
def save_hash_cache():
    with open(HASH_CACHE_FILE, "w") as f:
        json.dump(hash_cache, f, indent=2)

# After all chunking & image doc creation:
save_hash_cache()
```

In [36]:
```python
import json
json_data=[]
for doc in all_chunks:
    json_data.append({
```

```
        "page_content":doc.page_content,
        "metadata":doc.metadata
    })

chunk_file = CHUNK_DIR / "pdf_chunks.json"
with open(chunk_file, "w", encoding="utf-8") as f:
    json.dump(json_data, f, ensure_ascii=False, indent=2)

print(f"Saved {len(json_data)} chunks to: {chunk_file}")
```

```
Saved 0 chunks to: C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\chunks\pdf_chunks.js
on
```

now converting these chunks into vectors using embedding and storing in database

In [38]:
```
from langchain_community.vectorstores import FAISS
from langchain_community.document_loaders import TextLoader
from langchain_core.documents import Document
```

In [39]:
```
!ollama pull nomic-embed-text
```

```
pulling manifest ⠋ pulling manifest ⠙ pulling manifest ⠹ pulling manifest ⠸ pulling manifest ⠼ pulling manifest ⠴ p
ulling manifest ⠦ pulling manifest ⠧ pulling manifest ⠇ pulling manifest ⠏ pulling manifest ⠋ pulling manifest ⠙ pu
lling manifest ⠹ pulling manifest ⠸ pulling manifest ⠼ pulling manifest ⠴ pulling manifest ⠦ pulling manifest ⠧ pul
ling manifest ⠇ pulling manifest ⠏ pulling manifest ⠋ pulling manifest ⠙ pulling manifest ⠹ pulling manifest ⠸ pull
ing manifest ⠼ pulling manifest ⠴ pulling manifest ⠦ pulling manifest ⠧ pulling manifest ⠇ pulling manifest ⠏ pulli
ng manifest
pulling 970aa74c0a90: 100% ▕██████████████████▏ 274 MB
pulling c71d239df917: 100% ▕██████████████████▏  11 KB
pulling ce4a164fc046: 100% ▕██████████████████▏   17 B
pulling 31df23ea7daa: 100% ▕██████████████████▏  420 B
verifying sha256 digest
writing manifest
success
```

In [40]:
```
CHUNK_FILE=Path(r"C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\chunks\pdf_chunks.js
INDEX_DIR=Path(r"C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\faiss_index\ollama")
```

In [41]:
```
from langchain_ollama import OllamaEmbeddings
```

In [42]:
```
embedding_model=OllamaEmbeddings(model="nomic-embed-text")
```

In [43]:
```
def load_documents_from_json(json_path):
    with open(json_path, "r", encoding="utf-8") as f:
        raw_docs=json.load(f)
    return [Document(page_content=d["page_content"], metadata=d["metadata"]) for d in raw_docs]

documents = load_documents_from_json(CHUNK_FILE)
documents.extend(image_documents)
print(f" Loaded {len(documents)} documents")
```

```
Loaded 801 documents
```

*⬇️ **Load image summaries as documents***

In [45]:
```
import tiktoken
```

In [46]:
```
tokenizer=tiktoken.get_encoding("cl100k_base")
def num_tokens(text):
    return len(tokenizer.encode(text))
```

In [47]:
```
INDEX_DIR.mkdir(parents=True, exist_ok=True)
```

In [48]:
```
from langchain_community.vectorstores import FAISS

if (INDEX_DIR / "index.faiss").exists():
    print("📂 Loading existing FAISS index from disk...")
    vectorstore = FAISS.load_local(
        str(INDEX_DIR),
        embeddings=embedding_model,
        allow_dangerous_deserialization=True
    )
else:
    print("🆕 Creating new FAISS index...")
    vectorstore = None
```

```
📂 Loading existing FAISS index from disk...
```

In [49]:
```
BATCH_TOKEN_LIMIT = 150000
current_batch = []
current_tokens = 0

for doc in tqdm(documents, desc="Embedding Documents"):
```

```
    tokens = num_tokens(doc.page_content)
    if current_tokens + tokens > BATCH_TOKEN_LIMIT:
        texts = [d.page_content for d in current_batch]
        metadatas = [d.metadata for d in current_batch]

        if vectorstore is None:
            vectorstore = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
        else:
            new_store = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
            vectorstore.merge_from(new_store)

        current_batch = []
        current_tokens = 0

    current_batch.append(doc)
    current_tokens += tokens

# ⬇️ Final leftover batch
if current_batch:
    texts = [d.page_content for d in current_batch]
    metadatas = [d.metadata for d in current_batch]
    if vectorstore is None:
        vectorstore = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
    else:
        new_store = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
        vectorstore.merge_from(new_store)
```

```
Embedding Documents: 100%|██████████| 801/801 [01:07<00:00, 11.82it/s]
```

to inspect the vector embeddings stored in FAISS index

In [51]:
```
from langchain.chains import StuffDocumentsChain, LLMChain, RetrievalQA
from langchain_ollama import ChatOllama
from langchain.prompts import ChatPromptTemplate, PromptTemplate
```

In [52]:
```
INDEX_DIR=Path(r"C:\Users\subha\Desktop\LLM Projects\LocalPDFChatBot-IU\FAISS index and chunks\faiss_index\ollama")
```

In [53]:
```
embedding_model = OllamaEmbeddings(model="nomic-embed-text")
```

In [54]:
```
from langchain_community.vectorstores import FAISS
```

```
# 1. Define index path
index_path = INDEX_DIR / "index.faiss"

# 2. Try loading if exists
if index_path.exists():
    print("📂 Loading existing FAISS index...")
    vectorstore = FAISS.load_local(
        str(INDEX_DIR),
        embeddings=embedding_model,
        allow_dangerous_deserialization=True
    )
else:
    print("🆕 Creating new FAISS index...")

    # Prepare your `documents` list first (e.g. from chunks and image_documents)
    texts = [doc.page_content for doc in documents]
    metadatas = [doc.metadata for doc in documents]

    # Now create the vectorstore
    vectorstore = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)

    # Save it
    vectorstore.save_local(str(INDEX_DIR))
    print(f"✅ FAISS index saved to: {INDEX_DIR}")
```

```
📂 Loading existing FAISS index...
```

In [55]:
```
retriever = vectorstore.as_retriever(search_kwargs={"k": 10})  # higher k for reranking

# 🔁 Load reranker model once
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"

reranker_model = AutoModelForSequenceClassification.from_pretrained("BAAI/bge-reranker-base").to(device)
reranker_tokenizer = AutoTokenizer.from_pretrained("BAAI/bge-reranker-base")
```

In [56]:
```
def rerank_bge(query, docs, top_n=2):
    pairs = [(query, doc.page_content) for doc in docs]
    inputs = reranker_tokenizer(
        [q for q, d in pairs],
```

```
        [d for q, d in pairs],
        return_tensors="pt",
        padding=True,
        truncation=True,
        max_length=512
    ).to(device)

    with torch.no_grad():
        scores = reranker_model(**inputs).logits.squeeze(-1)

    ranked = sorted(zip(scores.tolist(), docs), key=lambda x: x[0], reverse=True)
    return [doc for _, doc in ranked[:top_n]]
```

In [57]:

```
from langchain_community.vectorstores import FAISS

# Load or create FAISS index
if (INDEX_DIR / "index.faiss").exists():
    print("📂 Loading existing FAISS index...")
    vectorstore = FAISS.load_local(
        str(INDEX_DIR),
        embeddings=embedding_model,
        allow_dangerous_deserialization=True
    )
else:
    print("🆕 Creating new FAISS index...")
    vectorstore = None

# Embed only new documents
documents_to_embed = new_chunks + new_image_documents

if documents_to_embed:
    print(f"🧠 Embedding {len(documents_to_embed)} new documents...")
    BATCH_TOKEN_LIMIT = 150000
    current_batch = []
    current_tokens = 0

    for doc in tqdm(documents_to_embed, desc="Embedding Documents"):
        tokens = num_tokens(doc.page_content)
        if current_tokens + tokens > BATCH_TOKEN_LIMIT:
            texts = [d.page_content for d in current_batch]
            metadatas = [d.metadata for d in current_batch]
```

```
            if vectorstore is None:
                vectorstore = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
            else:
                new_store = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
                vectorstore.merge_from(new_store)

            current_batch = []
            current_tokens = 0

        current_batch.append(doc)
        current_tokens += tokens

    # Final leftover batch
    if current_batch:
        texts = [d.page_content for d in current_batch]
        metadatas = [d.metadata for d in current_batch]
        if vectorstore is None:
            vectorstore = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
        else:
            new_store = FAISS.from_texts(texts, embedding_model, metadatas=metadatas)
            vectorstore.merge_from(new_store)

    # ✅ Save updated index
    vectorstore.save_local(str(INDEX_DIR))
    print(f"✅ Saved updated FAISS index to: {INDEX_DIR}")
```

```
📂 Loading existing FAISS index...
```

Setting Promt

In [59]:

```
prompt = PromptTemplate(
    input_variables=["context", "question"],
    template="""
You are a document question-answering assistant.
Use only the parts of the context that directly answer the user's question.
Ignore any unrelated sections like examples, task instructions, or summaries.

If no useful answer is found, respond: "I couldn’t find an answer based on the provided documents."

--- Context ---
```

```
{context}
----------------

Question: {question}
Answer:
""".strip()
)
```

In [60]:
```
#setting llm
llm=ChatOllama(model="phi3",temperature=1)

#wrap llm with prompt
llm_chain=LLMChain(llm=llm,prompt=prompt)

#wrap it into stuffdocumentchain
stuff_chain=StuffDocumentsChain(
    llm_chain=llm_chain,
    document_variable_name="context"   #matches variable used in promt
)

#passing it to retrival QA

qa_chain=RetrievalQA(
    retriever=retriever,
    combine_documents_chain=stuff_chain,
    return_source_documents=True
)
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_5284\3162385832.py:5: LangChainDeprecationWarning: The class `LLMChain` w
as deprecated in LangChain 0.1.17 and will be removed in 1.0. Use :meth:`~RunnableSequence, e.g., `prompt | llm`` ins
tead.
  llm_chain=LLMChain(llm=llm,prompt=prompt)
C:\Users\subha\AppData\Local\Temp\ipykernel_5284\3162385832.py:8: LangChainDeprecationWarning: This class is deprecat
ed. Use the `create_stuff_documents_chain` constructor instead. See migration guide here: https://python.langchain.co
m/docs/versions/migrating_chains/stuff_docs_chain/
  stuff_chain=StuffDocumentsChain(
C:\Users\subha\AppData\Local\Temp\ipykernel_5284\3162385832.py:15: LangChainDeprecationWarning: This class is depreca
ted. Use the `create_retrieval_chain` constructor instead. See migration guide here: https://python.langchain.com/doc
s/versions/migrating_chains/retrieval_qa/
  qa_chain=RetrievalQA(
```

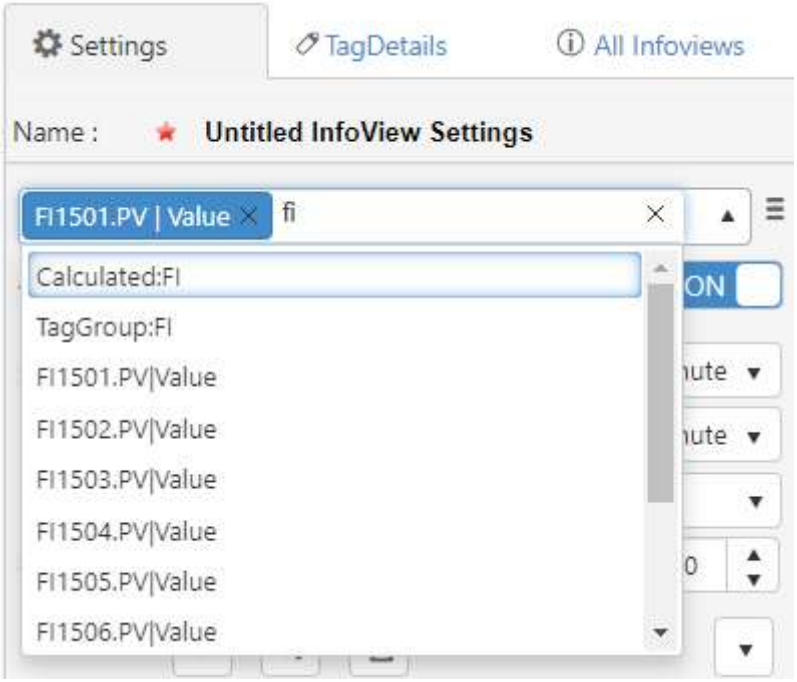
now taking the Query

In [62]:
```python
from IPython.display import display, Image as IPyImage
import torch

def ask_questions(qa_chain, retriever, reranker_model, reranker_tokenizer, image_dir, device="cpu"):
    while True:
        query = input("\n🔎 Ask your question (or type 'exit' to quit):\n").strip()
        if query.lower() in ["exit", "quit"]:
            print("👋 Exiting the chatbot.")
            break

        # Step 1: Retrieve top-k candidates
        retrieved_docs = retriever.get_relevant_documents(query)

        # Step 2: Rerank using bge-reranker
        pairs = [(query, doc.page_content) for doc in retrieved_docs]
        inputs = reranker_tokenizer(
            [q for q, d in pairs],
            [d for q, d in pairs],
            return_tensors="pt",
            padding=True,
            truncation=True,
            max_length=512
        ).to(device)

        with torch.no_grad():
            scores = reranker_model(**inputs).logits.squeeze(-1)

        ranked = sorted(zip(scores.tolist(), retrieved_docs), key=lambda x: x[0], reverse=True)
        top_docs = [doc for _, doc in ranked[:2]]

        # Step 3: Run QA chain manually
        response = qa_chain.combine_documents_chain.run(
            input_documents=top_docs,
            question=query
        )
        answer = response.strip()
```

```
        # Step 4: Output
        if "i don't know" in answer.lower() or "not able to answer" in answer.lower():
            print("\n❗ Sorry, I couldn't find this information in the documents.")
        else:
            print("\n✅ Answer:\n", answer)

            print("\n📚 Sources:")
            for doc in top_docs:
                source = doc.metadata.get("source", "Unknown")
                page = doc.metadata.get("page", "N/A")
                print(f"- {source} (Page {page})")

                # 🌄 Display image if it's part of the retrieved docs
                if doc.metadata.get("type") == "image":
                    image_file = doc.metadata.get("image_file")
                    if image_file:
                        image_path = image_dir / image_file
                        if image_path.exists():
                            print(f"\n🖼️ Related Image from Page {page} of {source}:")
                            display(IPyImage(filename=str(image_path)))
```

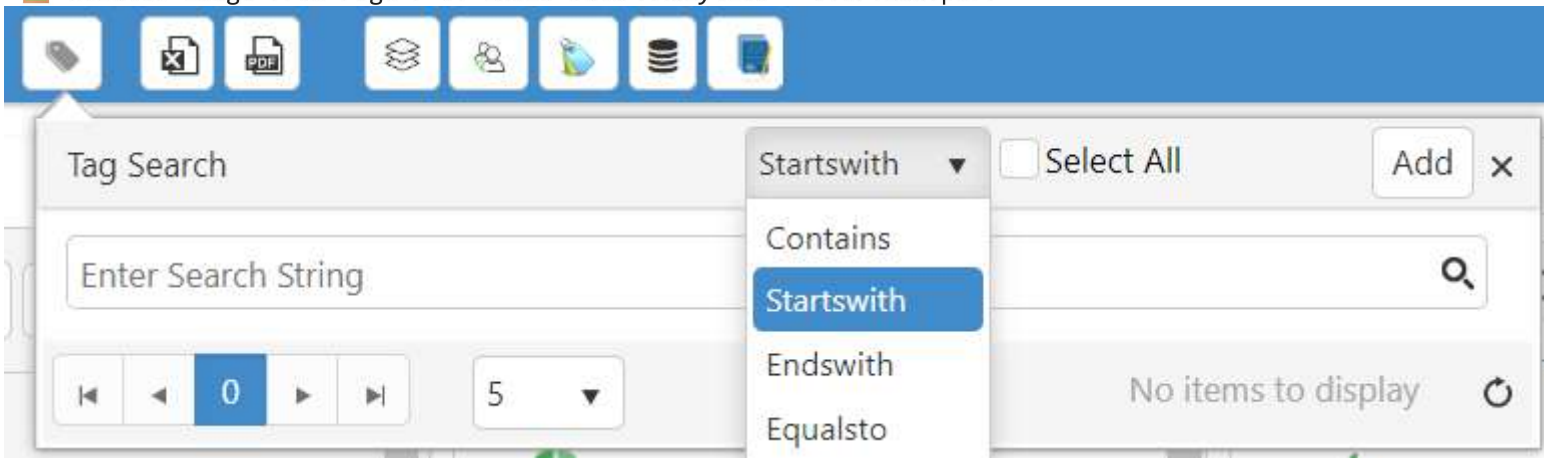
In [63]:
```
ask_questions(qa_chain, retriever, reranker_model, reranker_tokenizer, IMAGE_DIR, device=device)
```

```
C:\Users\subha\AppData\Local\Temp\ipykernel_5284\1225680841.py:12: LangChainDeprecationWarning: The method `BaseRetriever.get_relevant_documents` was deprecated in langchain-core 0.1.46 and will be removed in 1.0. Use :meth:`~invoke` instead.
  retrieved_docs = retriever.get_relevant_documents(query)
C:\Users\subha\AppData\Local\Temp\ipykernel_5284\1225680841.py:32: LangChainDeprecationWarning: The method `Chain.run` was deprecated in langchain 0.1.0 and will be removed in 1.0. Use :meth:`~invoke` instead.
  response = qa_chain.combine_documents_chain.run(
✅ Answer:
 Quickly finding relevant tags from thousands of options instantly by typing part or full name/description into a search box. It provides immediate information such as Tag Name, Description & Units and current value with the last half an hour trend. Users can also apply quick search criteria to filter results further.

📚 Sources:
- inSis Data Analytics - Overview.pdf (Page 7)

🖼️ Related Image from Page 7 of inSis Data Analytics - Overview.pdf:
```



- inSis Data Analytics - Overview.pdf (Page 7)

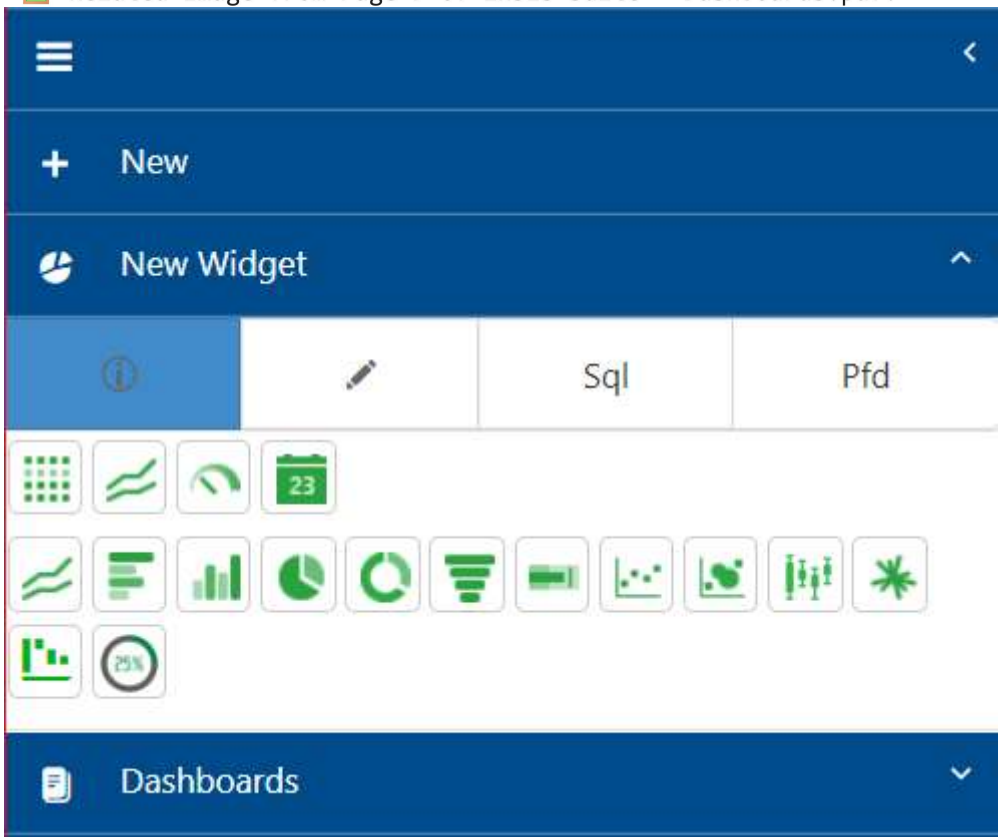
Related Image from Page 7 of inSis Data Analytics - Overview.pdf:



✅ Answer:
The provided context does not contain a direct answer about what an Analytics Widget provides; therefore, I cannot give you that information based on this document alone. It only mentions the existence and options of such widgets without detailing their specific functions or outputs regarding process operations insights for quick decision-making.

📚 Sources:
- inSis Suite - Dashboards.pdf (Page 9)
- inSis Suite - Dashboards.pdf (Page 9)

🖼️ Related Image from Page 9 of inSis Suite - Dashboards.pdf:



👋 Exiting the chatbot.

In [114…
```
ask_questions(qa_chain, retriever, reranker_model, reranker_tokenizer, IMAGE_DIR, device=device)
```

✅ Answer:
- Sensor data

- Real-time, historical data and publish (set manual data)

- Calculated Tags, KPIs & aggregates data

- Digital Logbooks data

- Reports data, etc.

📚 Sources:
- inSis Python Package.pdf (Page 2)
- inSis Python Package.pdf (Page 4)
👋 Exiting the chatbot.