In [1]:
```
# System packages
!apt-get -y update >/dev/null
!apt-get -y install poppler-utils tesseract-ocr >/dev/null

# Python deps
!pip -q install "langchain>=0.2.0" "langchain-core>=0.2.0" langchain-
community \
                langchain-google-genai google-generativeai chromadb \
                "unstructured[all-docs]" camelot-py[cv] \
                google-cloud-vision pytesseract pillow lxml pydantic
tiktoken gradio
```

```
W: Skipping acquire of configured file 'main/source/Sources' as
repository 'https://r2u.stat.illinois.edu/ubuntu jammy InRelease'
does not seem to provide it (sources.list entry misspelt?)
[2K     [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m67.3/67.3 kB[0m [31m5.5 MB/s[0m eta [36m0:00:00[0m
[?25h  Installing build dependencies ... [?25l[?25hdone
  Getting requirements to build wheel ... [?25l[?25hdone
  Preparing metadata (pyproject.toml) ... [?25l[?25hdone
[2K     [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m981.5/981.5 kB[0m [31m30.3 MB/s[0m eta [36m0:00:00[0m
[?25h  Preparing metadata (setup.py) ... [?25l[?25hdone
[33mWARNING: camelot-py 1.0.9 does not provide the extra
'cv'[0m[33m
[2K     [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m67.7/67.7 kB[0m [31m4.9 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m2.5/2.5 MB[0m [31m52.7 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m63.6/63.6 kB[0m [31m5.3 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m475.3/475.3 kB[0m [31m33.6 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m21.4/21.4 MB[0m [31m91.6 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m529.1/529.1 kB[0m [31m36.1 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m278.2/278.2 kB[0m [31m22.3 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m2.0/2.0 MB[0m [31m81.4 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m1.0/1.0 MB[0m [31m50.5 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m103.3/103.3 kB[0m [31m8.7 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m18.1/18.1 MB[0m [31m76.9 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m17.4/17.4 MB[0m [31m94.7 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m72.5/72.5 kB[0m [31m5.8 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m132.4/132.4 kB[0m [31m10.6 MB/s[0m eta [36m0:00:00[0m
```

␛[2K   ␛[90m━━␛[0m
␛[32m66.4/66.4 kB␛[0m ␛[31m5.6 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m220.0/220.0 kB␛[0m ␛[31m17.7 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m5.6/5.6 MB␛[0m ␛[31m101.0 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m105.4/105.4 kB␛[0m ␛[31m8.9 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m71.6/71.6 kB␛[0m ␛[31m5.4 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m313.2/313.2 kB␛[0m ␛[31m25.4 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m3.0/3.0 MB␛[0m ␛[31m100.4 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m253.0/253.0 kB␛[0m ␛[31m18.6 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m472.8/472.8 kB␛[0m ␛[31m30.9 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m64.7/64.7 kB␛[0m ␛[31m5.2 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m47.9/47.9 kB␛[0m ␛[31m3.6 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m66.8/66.8 kB␛[0m ␛[31m5.3 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m112.5/112.5 kB␛[0m ␛[31m9.4 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m608.4/608.4 kB␛[0m ␛[31m40.9 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m48.7/48.7 kB␛[0m ␛[31m3.8 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m1.4/1.4 MB␛[0m ␛[31m60.8 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m2.6/2.6 MB␛[0m ␛[31m88.0 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m167.8/167.8 kB␛[0m ␛[31m13.2 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m3.2/3.2 MB␛[0m ␛[31m86.5 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m1.8/1.8 MB␛[0m ␛[31m78.1 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m207.8/207.8 kB␛[0m ␛[31m15.8 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m517.7/517.7 kB␛[0m ␛[31m33.6 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m50.9/50.9 kB␛[0m ␛[31m4.2 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m114.6/114.6 kB␛[0m ␛[31m9.9 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m128.4/128.4 kB␛[0m ␛[31m11.4 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m4.4/4.4 MB␛[0m ␛[31m94.6 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m456.8/456.8 kB␛[0m ␛[31m34.2 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m
␛[32m175.3/175.3 kB␛[0m ␛[31m14.5 MB/s␛[0m eta ␛[36m0:00:00␛[0m
␛[2K   ␛[90m━━␛[0m

```
[32m46.0/46.0 kB[0m [31m3.7 MB/s[0m eta [36m0:00:00[0m
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m
[32m86.8/86.8 kB[0m [31m6.7 MB/s[0m eta [36m0:00:00[0m
[?25h  Building wheel for pypika (pyproject.toml) ... [?25l[?
25hdone
  Building wheel for langdetect (setup.py) ... [?25l[?25hdone
[31mERROR: pip's dependency resolver does not currently take into
account all the packages that are installed. This behaviour is the
source of the following dependency conflicts.
google-colab 1.0.0 requires requests==2.32.4, but you have
requests 2.32.5 which is incompatible.
opentelemetry-exporter-gcp-logging 1.11.0a0 requires
opentelemetry-sdk<1.39.0,>=1.35.0, but you have opentelemetry-sdk
1.39.0 which is incompatible.
google-adk 1.20.0 requires opentelemetry-api<=1.37.0,>=1.37.0, but
you have opentelemetry-api 1.39.0 which is incompatible.
google-adk 1.20.0 requires opentelemetry-sdk<=1.37.0,>=1.37.0, but
you have opentelemetry-sdk 1.39.0 which is incompatible.
opentelemetry-exporter-otlp-proto-http 1.37.0 requires
opentelemetry-exporter-otlp-proto-common==1.37.0, but you have
opentelemetry-exporter-otlp-proto-common 1.39.0 which is
incompatible.
opentelemetry-exporter-otlp-proto-http 1.37.0 requires
opentelemetry-proto==1.37.0, but you have opentelemetry-proto
1.39.0 which is incompatible.
opentelemetry-exporter-otlp-proto-http 1.37.0 requires
opentelemetry-sdk~=1.37.0, but you have opentelemetry-sdk 1.39.0
which is incompatible.[0m[31m
[0m
```

In [2]:

```
import os, io, uuid, time, base64, glob, shutil
from typing import List, Tuple

# OCR (local)
import pytesseract
pytesseract.pytesseract.tesseract_cmd = "/usr/bin/tesseract"  #
Colab/Linux

from PIL import Image

# File upload + PDF parsing
from google.colab import files
from unstructured.partition.pdf import partition_pdf

# LangChain + Gemini (AI Studio)
from langchain_google_genai import ChatGoogleGenerativeAI,
GoogleGenerativeAIEmbeddings
from langchain_core.messages import HumanMessage

# Native AI Studio SDK (we use LC for text tasks, but keep SDK handy)
import google.generativeai as genai

# Google Cloud Vision
from google.oauth2 import service_account
from google.cloud import vision

# Vector store
from langchain_community.vectorstores import Chroma
from langchain_core.documents import Document

# Prompt for secrets safely (no hardcoding)
import getpass

# ---- Gemini (AI Studio) ----
if not os.environ.get("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = getpass.getpass("Enter your Gemini AI
Studio API key: ").strip()
genai.configure(api_key=os.environ["GOOGLE_API_KEY"])
GEMINI_TEXT_MODEL = "gemini-pro-latest"  # you confirmed this works

# ---- Google Cloud Vision (service account *fields*) ----
# You can paste values when prompted, or set them as env vars
beforehand.
GOOGLE_PROJECT_ID   = os.environ.get("GOOGLE_PROJECT_ID")   or
input("GOOGLE_PROJECT_ID: ").strip()
GOOGLE_CLIENT_EMAIL = os.environ.get("GOOGLE_CLIENT_EMAIL") or
input("GOOGLE_CLIENT_EMAIL: ").strip()
GOOGLE_PRIVATE_KEY  = os.environ.get("GOOGLE_PRIVATE_KEY")  or
getpass.getpass("GOOGLE_PRIVATE_KEY (include BEGIN/END, newlines ok or
\\n): ")

# Normalize private key in case it was pasted with escaped \n
GOOGLE_PRIVATE_KEY = GOOGLE_PRIVATE_KEY.replace("\\n", "\n")

sa_info = {
    "type": "service_account",
    "project_id": GOOGLE_PROJECT_ID,
```

```
    "private_key_id": "ignored",
    "private_key": GOOGLE_PRIVATE_KEY,
    "client_email": GOOGLE_CLIENT_EMAIL,
    "client_id": "ignored",
    "token_uri": "https://oauth2.googleapis.com/token",
}
creds = service_account.Credentials.from_service_account_info(sa_info)
vision_client = vision.ImageAnnotatorClient(credentials=creds)

# Working dirs
INPUT_DIR  = "/content"
OUTPUT_DIR = "/content/output"
PERSIST_DIR = "/content/chroma_db"  # optional persistence (used by
Gradio too)
os.makedirs(OUTPUT_DIR, exist_ok=True)
os.makedirs(PERSIST_DIR, exist_ok=True)
```

```
Enter your Gemini AI Studio API key: ··········
GOOGLE_PROJECT_ID: smart-recipe-generator-474722
GOOGLE_CLIENT_EMAIL: recipe-vision-service@smart-recipe-generator-
474722.iam.gserviceaccount.com
GOOGLE_PRIVATE_KEY (include BEGIN/END, newlines ok or \n):
··········
```

In [3]:
```
uploaded = files.upload()
assert uploaded, "No file uploaded."
pdf_name = list(uploaded.keys())[0]
pdf_path = os.path.join(INPUT_DIR, pdf_name)
print("Using PDF:", pdf_path)

# Try Unstructured hi_res (better layout capture), then fallback
try:
    raw_pdf_elements = partition_pdf(
        filename=pdf_path,
        strategy="hi_res",
        infer_table_structure=True,
        extract_images_in_pdf=True,
        image_output_dir_path=OUTPUT_DIR,
        pdf_image_dpi=300,
        chunking_strategy="by_title",
        max_characters=4000,
        new_after_n_chars=3800,
        combine_text_under_n_chars=2000,
    )
except Exception as e:
    print("partition_pdf failed, fallback to default:", e)
    raw_pdf_elements = partition_pdf(
        filename=pdf_path,
        infer_table_structure=True,
        extract_images_in_pdf=True,
        image_output_dir_path=OUTPUT_DIR,
    )

text_elements, table_elements = [], []
for el in raw_pdf_elements:
    t = str(type(el))
    if "CompositeElement" in t:
        text_elements.append(el.text or "")
    elif "Table" in t:
        table_elements.append(el.text or "")

print(f"Unstructured → Text chunks: {len(text_elements)}, Tables:
{len(table_elements)}")

# Camelot fallback for tables that are "text + lines"
import camelot
try:
    cam_tables = camelot.read_pdf(pdf_path, pages="all",
flavor="stream")
    print("Camelot tables found:", cam_tables.n)
    camelot_table_texts = [t.df.to_csv(index=False) for t in cam_tables]
except Exception as e:
    print("Camelot failed:", e)
    camelot_table_texts = []

if not table_elements and camelot_table_texts:
    table_elements = camelot_table_texts

print(f"Final table count used: {len(table_elements)}")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test1.pdf to test1.pdf
Using PDF: /content/test1.pdf
Warning: No languages specified, defaulting to English.

yolox_l0.05.onnx:   0%|          | 0.00/217M [00:00<?, ?B/s]

preprocessor_config.json:   0%|          | 0.00/274 [00:00<?, ?
B/s]

  The `max_size` parameter is deprecated and will be removed in
  v4.26. Please specify in `size['longest_edge'] instead`.

config.json: 0.00B [00:00, ?B/s]

model.safetensors:   0%|          | 0.00/115M [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/46.8M [00:00<?, ?B/s]

Unstructured → Text chunks: 38, Tables: 0
Camelot failed: unsupported format string passed to
list.__format__
Final table count used: 0
```

In [4]:

```
# Render each page to PNG (works for vector PDFs too)
!mkdir -p "{OUTPUT_DIR}"
!pdftoppm -png -r 200 "{pdf_path}" "{OUTPUT_DIR}/page"

page_pngs = sorted(glob.glob(os.path.join(OUTPUT_DIR, "page-*.png")))
print("Rendered page images:", len(page_pngs))

def read_image_bytes(path: str) -> bytes:
    with open(path, "rb") as f:
        return f.read()

image_bytes_list = [read_image_bytes(p) for p in page_pngs]
```

```
Rendered page images: 19
```

In [5]:

```
# ---- Cloud Vision → labels + OCR ----
def vision_labels_and_ocr(image_bytes: bytes, max_labels=10):
    if not image_bytes:
        return [], ""
    img = vision.Image(content=image_bytes)

    labels = []
    try:
        lr = vision_client.label_detection(image=img)
        if lr.label_annotations:
            labels = [l.description for l in
lr.label_annotations[:max_labels]]
    except Exception as e:
        print("Vision label_detection error:", e)

    ocr_text = ""
    try:
        dr = vision_client.document_text_detection(image=img)
        if getattr(dr, "full_text_annotation", None):
            ocr_text = dr.full_text_annotation.text or ""
    except Exception as e:
        print("Vision document_text_detection error:", e)

    return labels, (ocr_text or "").strip()

# ---- Tesseract OCR (fallback or augmentation) ----
def tesseract_ocr(image_bytes: bytes, lang="eng"):
    if not image_bytes:
        return ""
    try:
        im = Image.open(io.BytesIO(image_bytes)).convert("RGB")
        text = pytesseract.image_to_string(im, lang=lang)
        return (text or "").strip()
    except Exception as e:
        print("Tesseract OCR error:", e)
        return ""

# ---- Gemini (text) for summaries ----
gemini_text = ChatGoogleGenerativeAI(model=GEMINI_TEXT_MODEL,
max_output_tokens=512)

def summarize_text_chunk(txt: str) -> str:
    if not txt or not str(txt).strip():
        return ""
    prompt = f"Summarize the following in 3–5 concise bullet
points:\n\n{txt}\n\nSummary:"
    return (gemini_text.invoke([HumanMessage(content=prompt)]).content
or "").strip()

def summarize_table_chunk(tbl: str) -> str:
    if not tbl or not str(tbl).strip():
        return ""
    prompt = f"Summarize the table in 3–5 concise bullet points. Focus
on trends and key figures:\n\n{tbl}\n\nSummary:"
    return (gemini_text.invoke([HumanMessage(content=prompt)]).content
or "").strip()
```

```
def summarize_image_bytes(img_bytes: bytes, augment_with_tesseract=True)
-> str:
    """Vision labels+OCR -> (optionally augment with Tesseract) ->
Gemini bullet summary."""
    labels, ocr_text = vision_labels_and_ocr(img_bytes)
    if augment_with_tesseract:
        t_ocr = tesseract_ocr(img_bytes)
        # Merge OCRs (prefer Vision, append tesseract if additional text
found)
        if t_ocr and t_ocr not in ocr_text:
            ocr_text = (ocr_text + "\n" + t_ocr).strip() if ocr_text
else t_ocr

    labels_str = ", ".join(labels) if labels else "None"
    snippet = ocr_text[:1200] + ("..." if len(ocr_text) > 1200 else "")
    prompt = (
        "You are given outputs from image analysis of a PDF page. "
        "Write 3–5 concise bullet points describing the page. "
        "Include salient visual elements and meaningful text cues.\n\n"
        f"Labels: {labels_str}\n\n"
        f"OCR (snippet):\n{snippet}\n\n"
        "Bulleted description:"
    )
    return (gemini_text.invoke([HumanMessage(content=prompt)]).content
or "").strip()
```

In [7]:

```
MAX_ITEMS = 10  # set to an int (e.g., 10) to test faster; None =
process all

def maybe_slice(lst):
    return lst if MAX_ITEMS is None else lst[:MAX_ITEMS]

text_summaries  = [summarize_text_chunk(t) for t in
maybe_slice(text_elements) if str(t).strip()]
table_summaries = [summarize_table_chunk(t) for t in
maybe_slice(table_elements) if str(t).strip()]
image_summaries = [summarize_image_bytes(b) for b in
maybe_slice(image_bytes_list)]

print("Summaries → text:", len(text_summaries),
      "| tables:", len(table_summaries),
      "| images:", len(image_summaries))
```

```
Summaries → text: 10 | tables: 0 | images: 10
```

In [8]:
```
# Simple in-memory store for originals
class SimpleMemoryStore:
    def __init__(self): self._data = {}
    def mset(self, pairs):
        for k, v in pairs: self._data[k] = v
    def mget(self, keys):
        return [self._data.get(k) for k in keys]

store = SimpleMemoryStore()
ID_KEY = "doc_id"

# Embeddings (Gemini AI Studio)
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")

# Use persistence so Gradio chat can reload index across callbacks
vectorstore = Chroma(collection_name="summaries",
                     embedding_function=embeddings,
                     persist_directory=PERSIST_DIR)

def add_documents_to_index(summaries: List[str], originals: List[str]):
    pairs = [(str(s).strip(), str(o)) for s, o in zip(summaries,
originals) if s and str(s).strip()]
    if not pairs:
        print("Nothing to index (empty inputs).")
        return []
    s_list = [s for s, _ in pairs]
    o_list = [o for _, o in pairs]
    n = min(len(s_list), len(o_list))
    ids = [str(uuid.uuid4()) for _ in range(n)]
    docs = [Document(page_content=s_list[i], metadata={ID_KEY: ids[i]})
for i in range(n)]
    vectorstore.add_documents(docs)
    store.mset(list(zip(ids, o_list[:n])))
    vectorstore.persist()
    return ids

def retrieve_originals(query: str, k: int = 4):
    hits = vectorstore.similarity_search_with_score(query, k=k)
    out = []
    for summary_doc, score in hits:
        did = summary_doc.metadata.get(ID_KEY)
        original = store.mget([did])[0] if did else None
        if original:
            out.append((Document(page_content=original, metadata=
{ID_KEY: did}),
                        score,
                        summary_doc.page_content))
    return out
```

```
/tmp/ipython-input-1364733514.py:16:
LangChainDeprecationWarning: The class `Chroma` was deprecated
in LangChain 0.2.9 and will be removed in 1.0. An updated
version of the class exists in the `langchain-chroma package
and should be used instead. To use it run `pip install -U
`langchain-chroma` and import as `from `langchain_chroma import
```

```
Chroma``.
  vectorstore = Chroma(collection_name="summaries",
```

In [9]:

```
# Text
n_text = len(text_summaries)
add_documents_to_index(text_summaries, text_elements[:n_text])

# Tables
n_tab = len(table_summaries)
if n_tab:
    add_documents_to_index(table_summaries, table_elements[:n_tab])
else:
    print("No tables to index — skipping.")

# Images (use their summaries as originals for now)
n_img = len(image_summaries)
if n_img:
    add_documents_to_index(image_summaries, image_summaries)
else:
    print("No image summaries to index — skipping.")
```

```
/tmp/ipython-input-1364733514.py:32:
LangChainDeprecationWarning: Since Chroma 0.4.x the manual
persistence method is no longer supported as docs are
automatically persisted.
  vectorstore.persist()


No tables to index — skipping.
```

In [10]:
```
from langchain_core.runnables import RunnablePassthrough, RunnableLambda
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

retriever = vectorstore.as_retriever(search_kwargs={"k": 4})

def format_docs(docs):
    return "\n\n".join(d.page_content for d in docs)

template = """Answer the question based only on the following context.
If the answer is not in the context, say you don't know.

Context:
{context}

Question: {question}
"""
prompt = ChatPromptTemplate.from_template(template)

rag_model = ChatGoogleGenerativeAI(model=GEMINI_TEXT_MODEL,
temperature=0)

rag_chain = (
    {
        "context": retriever | RunnableLambda(format_docs),
        "question": RunnablePassthrough(),
    }
    | prompt
    | rag_model
    | StrOutputParser()
)

# Quick test
print(rag_chain.invoke("What do the images generally contain?"))
```

```
I don't know.
```

In [11]:
```
q = "Key findings from the document"
res = retrieve_originals(q, k=3)
if not res:
    print("No relevant documents found.")
else:
    for i, (doc, score, summary) in enumerate(res, 1):
        print(f"\nResult {i} | Score: {score:.4f}")
        print("Summary:", summary[:300])
        print("Original (first 300):", doc.page_content[:300])
```

```
Result 1 | Score: 0.6607
Summary: Here is a summary of the provided text in four concise
bullet points:

*   Arsenic contamination in drinking water is a major global
health crisis, impacting approximately 150 million people and
creating an urgent need for effective purification methods.
*   Various nanoparticles, such as iron oxide
Original (first 300): Table 2. BET Surface Areas of Different
Magnetic Nanoparticles

magnetic nanoparticles BET surface area (m?/g) MnFe,O, 180.0
MgFe,O, 70.3 ZnFe,O, 79.6 CuFe,0, 93.8 NiFe,O, 101.2 CoFe,0, 58.1

of alumina have low cost, high surface area, and good thermal
stability.°°** These nanoparticles have been u

Result 2 | Score: 0.6761
Summary: Here is a summary of the provided text in four concise
bullet points:

*   Water quality is deteriorating globally due to population
growth, industrialization, and agricultural activities, making
water pollution a critical issue with widespread impact.
*   Pollutants fall into three main categories:
Original (first 300): 1. INTRODUCTION

Water is the most important and essential component on the earth
for vital activities of living beings. Unfortunately, water
quality of our water resources is deteriorating continuously due
to geometrical growth of population, industrialization,
civilization, domestic, and agricultu

Result 3 | Score: 0.7069
Summary: Here are three bullet points describing the PDF page:

*   This page is an excerpt from a scientific journal article,
specifically a review in *Chemical Reviews*.
*   The main feature is "Table 4," which details the use of
different nanoparticles (e.g., zero-valent iron, titanium dioxide)
as adsorbe
Original (first 300): Here are three bullet points describing the
PDF page:

*   This page is an excerpt from a scientific journal article,
```

```
specifically a review in *Chemical Reviews*.
*   The main feature is "Table 4," which details the use of
different nanoparticles (e.g., zero-valent iron, titanium dioxide)
as adsorbe
```

In [12]:
```
import gradio as gr

# Re-open embeddings & vectorstore inside Gradio callbacks (using same
persist dir)
def get_vectorstore():
    return Chroma(collection_name="summaries",

embedding_function=GoogleGenerativeAIEmbeddings(model="models/embedding-
001"),
                  persist_directory=PERSIST_DIR)

def encode_image(path: str) -> str:
    with open(path, "rb") as f:
        return base64.b64encode(f.read()).decode("utf-8")

def process_pdf(file_obj):
    """Parse PDF, render pages, build summaries, index into Chroma.
Returns a markdown report."""
    # Clean output dir for this run
    if os.path.exists(OUTPUT_DIR):
        shutil.rmtree(OUTPUT_DIR)
    os.makedirs(OUTPUT_DIR, exist_ok=True)

    # 1) Parse PDF
    raw = partition_pdf(
        filename=file_obj.name,
        strategy="hi_res",
        infer_table_structure=True,
        extract_images_in_pdf=True,
        image_output_dir_path=OUTPUT_DIR,
        pdf_image_dpi=300,
        chunking_strategy="by_title",
        max_characters=4000,
        new_after_n_chars=3800,
        combine_text_under_n_chars=2000,
    )

    texts, tables = [], []
    for el in raw:
        tt = str(type(el))
        if "CompositeElement" in tt:
            texts.append(el.text or "")
        elif "Table" in tt:
            tables.append(el.text or "")

    # 2) Render pages to PNG
    os.system(f'pdftoppm -png -r 200 "{file_obj.name}" "
{OUTPUT_DIR}/page"')
    pages = sorted(glob.glob(os.path.join(OUTPUT_DIR, "page-*.png")))

    # 3) Summarize
    text_sums  = [summarize_text_chunk(t) for t in texts if
str(t).strip()]
    table_sums = [summarize_table_chunk(t) for t in tables if
str(t).strip()]
    img_sums   = []
    for p in pages:
```

```
        with open(p, "rb") as f:
            img_sums.append(summarize_image_bytes(f.read()))

    # 4) Index into Chroma
    vs = get_vectorstore()
    # Align and add
    if text_sums:
        vs.add_documents([Document(page_content=s) for s in text_sums])
        vs.persist()
    if table_sums:
        vs.add_documents([Document(page_content=s) for s in table_sums])
        vs.persist()
    if img_sums:
        vs.add_documents([Document(page_content=s) for s in img_sums])
        vs.persist()

    md = (
        "## 📘 Text Summaries\n\n" + ("\n\n".join(text_sums[:10]) or
"_None_") +
        "\n\n## 📊 Table Summaries\n\n" + ("\n\n".join(table_sums[:10])
or "_None_") +
        "\n\n## 🖼️ Image Descriptions (Vision + Gemini)\n\n" +
("\n\n".join(img_sums[:10]) or "_None_")
    )
    return md

def rag_answer(question):
    """Answer using the persisted Chroma index."""
    vs = get_vectorstore()
    retr = vs.as_retriever(search_kwargs={"k": 4})
    def format_docs(docs):
        return "\n\n".join(d.page_content for d in docs)
    prompt = ("Answer the question based only on the following context.
"
              "If the answer is not in the context, say you don't
know.\n\n"
              "Context:\n{context}\n\nQuestion: {question}")
    prompt_tpl = ChatPromptTemplate.from_template(prompt)
    model = ChatGoogleGenerativeAI(model=GEMINI_TEXT_MODEL,
temperature=0)
    chain = (
        {
            "context": retr | RunnableLambda(format_docs),
            "question": RunnablePassthrough(),
        } | prompt_tpl | model | StrOutputParser()
    )
    try:
        return chain.invoke(question)
    except Exception as e:
        return f"RAG error: {e}"

with gr.Blocks() as demo:
    gr.Markdown("# 📄 PDF Summarizer — Vision (labels+OCR) + Tesseract
+ Gemini + Chroma RAG")

    with gr.Row():
        pdf_input = gr.File(label="Upload a PDF")
```

```
        process_btn = gr.Button("Process & Index")

    output_md = gr.Markdown(value="Upload a PDF and click **Process &
Index**.")
    with gr.Row():
        q = gr.Textbox(label="Ask a question about this PDF")
        ask_btn = gr.Button("Ask")
    a = gr.Textbox(label="Answer", lines=6)

    process_btn.click(fn=process_pdf, inputs=pdf_input,
outputs=output_md)
    ask_btn.click(fn=rag_answer, inputs=q, outputs=a)

demo.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set
debug=True in launch()
* Running on public URL: https://f4c2b342dedc705599.gradio.live

This share link expires in 1 week. For free permanent hosting and
GPU upgrades, run `gradio deploy` from the terminal in the working
directory to deploy to Hugging Face Spaces
(https://huggingface.co/spaces)
```