

```
!pip install jedi
```

```

Collecting jedi
  Downloading jedi-0.19.2-py2.py3-none-any.whl.metadata (22 kB)
    Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.12/dist-packages (from jedi) (0.8.5)
  Downloading jedi-0.19.2-py2.py3-none-any.whl (1.6 MB)
    _____ 1.6/1.6 MB 45.1 MB/s eta 0:00:00
Installing collected packages: jedi
Successfully installed jedi-0.19.2

```

```
!pip install -q torch transformers accelerate bitsandbytes langchain sentence-transformers faiss-cpu openpyxl pacmap dataset
```

```

_____ 647.5/647.5 kB 31.5 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
_____ 61.3/61.3 MB 9.1 MB/s eta 0:00:00
_____ 31.4/31.4 MB 27.0 MB/s eta 0:00:00
_____ 2.5/2.5 MB 56.7 MB/s eta 0:00:00
_____ 46.1/46.1 kB 3.8 MB/s eta 0:00:00
_____ 116.3/116.3 kB 9.8 MB/s eta 0:00:00
_____ 64.7/64.7 kB 6.4 MB/s eta 0:00:00
_____ 18.2/18.2 MB 41.8 MB/s eta 0:00:00
_____ 4.4/4.4 MB 45.5 MB/s eta 0:00:00
_____ 7.6/7.6 MB 22.5 MB/s eta 0:00:00
_____ 1.2/1.2 MB 41.6 MB/s eta 0:00:00
_____ 303.3/303.3 kB 18.2 MB/s eta 0:00:00
_____ 51.8/51.8 kB 4.1 MB/s eta 0:00:00
_____ 50.9/50.9 kB 4.8 MB/s eta 0:00:00
_____ 332.9/332.9 kB 24.9 MB/s eta 0:00:00
_____ 180.7/180.7 kB 12.0 MB/s eta 0:00:00
_____ 57.4/57.4 kB 4.7 MB/s eta 0:00:00
_____ 42.5/42.5 kB 3.3 MB/s eta 0:00:00
_____ 63.9/63.9 kB 5.5 MB/s eta 0:00:00
_____ 310.5/310.5 kB 23.8 MB/s eta 0:00:00
_____ 139.4/139.4 kB 8.9 MB/s eta 0:00:00
Building wheel for annoy (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour
google-colab 1.0.0 requires requests==2.32.4, but you have requests 2.32.5 which is incompatible.

```

```

from google.colab import drive
import time, glob, os
drive.mount('/content/drive', force_remount=True)

```

```

HF_CACHE = "/content/drive/MyDrive/hf-cache"
os.environ["HF_HOME"] = HF_CACHE
os.environ["HF_HUB_CACHE"] = HF_CACHE
os.environ["TRANSFORMERS_CACHE"] = os.path.join(HF_CACHE, "transformers")

```

```

INDEX_DIR = "/content/drive/MyDrive/rag-indexes"
RESULTS_DIR = "/content/drive/MyDrive/llm-evals"
os.makedirs(INDEX_DIR, exist_ok=True)
os.makedirs(RESULTS_DIR, exist_ok=True)

```

```

RUN_TAG = time.strftime("%Y%m%d-%H%M%S")
print("HF cache dir:", HF_CACHE)

```

```

Mounted at /content/drive
HF cache dir: /content/drive/MyDrive/hf-cache

```

```

os.makedirs(HF_CACHE, exist_ok=True)
os.makedirs(os.path.join(HF_CACHE, "transformers"), exist_ok=True)

```

```

# look for the Mistral repo in the cache
!find "$HF_CACHE" -maxdepth 2 -type d -name "models--mistralai--Mistral-7B-Instruct-v0.3" -print || true

```

```

/content/drive/MyDrive/hf-cache/.locks/models--mistralai--Mistral-7B-Instruct-v0.3
/content/drive/MyDrive/hf-cache/models--mistralai--Mistral-7B-Instruct-v0.3
/content/drive/MyDrive/hf-cache/transformers/models--mistralai--Mistral-7B-Instruct-v0.3

```

```

from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
import torch

```

```

model_id = "mistralai/Mistral-7B-Instruct-v0.3"
bnb = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16 if torch.cuda.is_available() else torch.float16,
)

```

```
reader_tok = AutoTokenizer.from_pretrained(model_id, cache_dir=HF_CACHE)
```



```

{
  "role": "user",
  "content": (
    "Context (excerpts):\n{context}\n"
    "----\n"
    "Question:\n{question}\n"
    "----\n"
    "Notes for the assistant:\n"
    f"- EMIT_JSON = {'true' if EMIT_JSON else 'false'}.\n"
    "- Follow the exact numbering and wording of the Instructions present inside the Question.\n"
    "- If an instruction is purely conceptual, focus on justification tied to Context (no freewheeling).\n"
    "- If an instruction requires calculations, show formula → substitution → result with appropriate rounding.\n"
  )
}
]

base_prompt_chat = [
  {"role": "system", "content": (
    "You are a Stats Tutor. An undergraduate statistics assistant Solve the question using correct statistical reasoning. "
    "Follow the question's Instructions in order to answer properly. Use the scenario to frame the question within context "
    "Follow the task as an explanation for what to do for each question"
    "Include the Null hypothesis H0/ and alternative hypothesis Ha, test & df, assumptions, formula then substitution of val
    "conclusion, and explicitly connect links between confidence intervals and hypothesis testing where relevant."
  )},
  {"role": "user", "content": (
    "Question:\n{question}\n----\n"
    "Number your answer 1..N or a..z to match the Instructions."
  )}
]

RAG_PROMPT_TEMPLATE = reader_tok.apply_chat_template(
  prompt_chat, tokenize=False, add_generation_prompt=True
)

BASE_PROMPT_TEMPLATE = reader_tok.apply_chat_template(
  base_prompt_chat, tokenize=False, add_generation_prompt=True
)

from tqdm.notebook import tqdm
import pandas as pd
from typing import Optional, List, Tuple
from datasets import Dataset
import matplotlib.pyplot as plt

pd.set_option("display.max_colwidth", None) #for visualising retriever outputs


from google.colab import files
import os, uuid

def upload_pdfs_to(folder: str):
  os.makedirs(folder, exist_ok=True)
  uploaded = files.upload() # select multiple PDFs
  for name, data in uploaded.items():
    if not name.lower().endswith(".pdf"):
      continue
    dst = os.path.join(folder, os.path.basename(name))
    if os.path.exists(dst):
      root, ext = os.path.splitext(dst)
      dst = f"{root}_{uuid.uuid4().hex[:6]}{ext}"
    with open(dst, "wb") as f: f.write(data)
    print("Saved to:", folder)


upload_pdfs_to("/content/ci_pdfs")

upload_pdfs_to("/content/ht_pdfs")

```

 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 ActualCI_Openintro.pdf to ActualCI_Openintro.pdf
 Saving CI_introductorystatsitics.pdf to CI_introductorystatsitics.pdf
 Saving CI_IntroToStats.pdf to CI_IntroToStats.pdf
 Saving ci_openintro.pdf to ci_openintro.pdf
 Saving ci_practicalstatsitics.pdf to ci_practicalstatsitics.pdf
 Saving ci_statsThinking.pdf to ci_statsThinking.pdf
 Saving inferenceskills.pdf to inferenceskills.pdf
 Saving MIT_CisolutionsExam.pdf to MIT_CisolutionsExam.pdf
 Saving MIT_OCWThreeViewsLecture.pdf to MIT_OCWThreeViewsLecture.pdf
 Saved to: /content/ci_pdfs

 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 HT_introductorystats1.pdf to HT_introductorystats1.pdf
 Saving HT_introductorystats2.pdf to HT_introductorystats2.pdf
 Saving HT_introstatistics.pdf to HT_introstatistics.pdf
 Saving HT_MITintro.pdf to HT_MITintro.pdf
 Saving HT_MITlecture.pdf to HT_MITlecture.pdf
 Saving HT_OpenIntro.pdf to HT_OpenIntro.pdf
 Saving HT_practicalstats.pdf to HT_practicalstats.pdf

```
from pathlib import Path
from langchain_community.document_loaders import PyPDFLoader
from transformers import AutoTokenizer
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain_community.vectorstores.utils import DistanceStrategy
```

```
EMBEDDING_MODEL_NAME = "thenlper/gte-small"
device = "cuda" if torch.cuda.is_available() else "cpu"
```


```
def load_folder_as_docs(folder: str):
    docs = []
    for pdf in Path(folder).glob("*.pdf"):
        for page in PyPDFLoader(str(pdf)).load():
            page.metadata = {"source": str(pdf), **page.metadata}
            docs.append(page)
    print(f"{folder}: {len(docs)} pages")
    return docs
```

```
def split_documents(kb, chunk_size_tokens=384, overlap_ratio=0.1, tokeniser_name=EMBEDDING_MODEL_NAME):
    tok = AutoTokenizer.from_pretrained(tokeniser_name)
    splitter = RecursiveCharacterTextSplitter.from_huggingface_tokenizer(
        tok, chunk_size=chunk_size_tokens, chunk_overlap=int(chunk_size_tokens*overlap_ratio),
        add_start_index=True, strip_whitespace=True
    )
    docs = splitter.split_documents(kb)
    seen, uniq = set(), []
    for d in docs:
        if d.page_content not in seen:
            seen.add(d.page_content); uniq.append(d)
    return uniq
```

```
def make_index(docs):
    emb = HuggingFaceEmbeddings(
        model_name=EMBEDDING_MODEL_NAME,
        model_kwargs={"device": device},
        encode_kwargs={"normalise_embeddings": True}
    )
    return FAISS.from_documents(docs, emb, distance_strategy=DistanceStrategy.COSINE)
```

```
ci_raw = load_folder_as_docs("/content/ci_pdfs")
ci_docs = split_documents(ci_raw, chunk_size_tokens=384)
ci_db = make_index(ci_docs)
CI_INDEX_DIR = f"{INDEX_DIR}/ci_faiss"
ci_db.save_local(CI_INDEX_DIR); print("Saved CI index ->", CI_INDEX_DIR)
```

```
ht_raw = load_folder_as_docs("/content/ht_pdfs")
ht_docs = split_documents(ht_raw, chunk_size_tokens=384)
ht_db = make_index(ht_docs)
HT_INDEX_DIR = f"{INDEX_DIR}/ht_faiss"
ht_db.save_local(HT_INDEX_DIR); print("Saved HT index ->", HT_INDEX_DIR)
```

 /content/ci_pdfs: 253 pages
 Saved CI index -> /content/drive/MyDrive/rag-indexes/ci_faiss
 /content/ht_pdfs: 204 pages
 Saved HT index -> /content/drive/MyDrive/rag-indexes/ht_faiss

```

!pip -q install pacmap

import numpy as np, pandas as pd
from pacmap import PaCMAP
import matplotlib.pyplot as plt

def faiss_to_arrays(vdb):
    index = vdb.index
    n = index.ntotal
    xb = np.zeros((n, index.d), dtype="float32")
    index.reconstruct_n(0, n, xb)
    id_map = vdb.index_to_docstore_id
    metas, texts = [], []
    for i in range(n):
        doc_id = id_map[i]
        doc = vdb.docstore.search(doc_id)
        metas.append(doc.metadata if doc else {})
        texts.append(doc.page_content if doc else "")
    return xb, metas, texts

X_ci, meta_ci, _ = faiss_to_arrays(ci_db)
X_ht, meta_ht, _ = faiss_to_arrays(ht_db)

df_ci = pd.DataFrame(meta_ci); df_ci["__label__"] = "CI"
df_ht = pd.DataFrame(meta_ht); df_ht["__label__"] = "HT"
X_all = np.vstack([X_ci, X_ht])
df_all = pd.concat([df_ci, df_ht], ignore_index=True)

def subsample(X, df, max_n=4000, seed=0):
    if len(X) <= max_n: return X, df
    rng = np.random.default_rng(seed)
    idx = rng.choice(len(X), size=max_n, replace=False)
    return X[idx], df.iloc[idx].reset_index(drop=True)

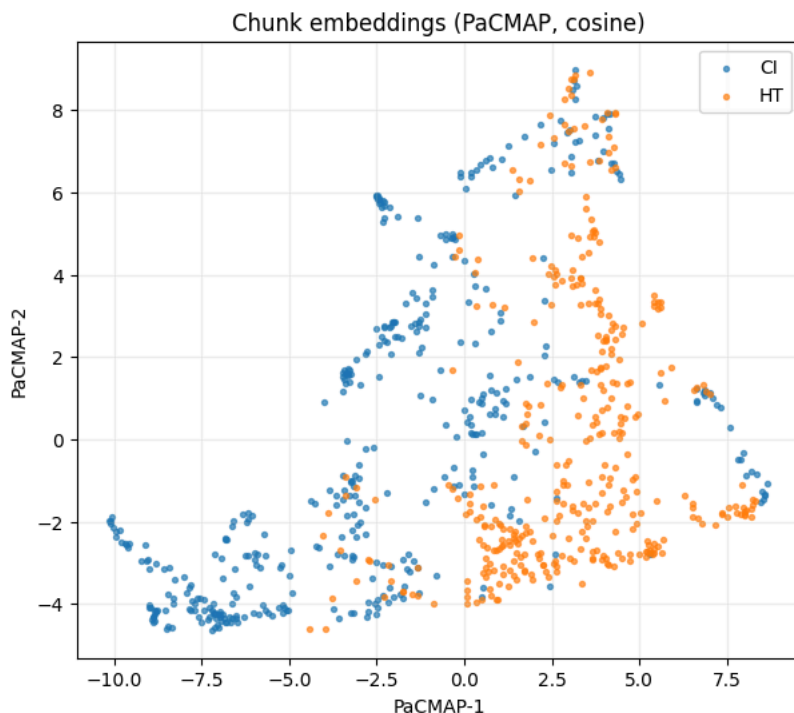
X_small, df_small = subsample(X_all, df_all, max_n=4000)

mapper = PaCMAP(n_components=2, random_state=0)
Z = mapper.fit_transform(X_small)
df_small["x"], df_small["y"] = Z[:,0], Z[:,1]

plt.figure(figsize=(7,6))
for name, sub in df_small.groupby("__label__"):
    plt.scatter(sub["x"], sub["y"], s=8, alpha=0.65, label=name)
plt.legend()
plt.title("Chunk embeddings (PaCMAP, cosine)")
plt.xlabel("PaCMAP-1"); plt.ylabel("PaCMAP-2"); plt.grid(True, alpha=0.2)
plt.show()

```

⚠ WARNING:pacmap.pacmap:Warning: random state is set to 0.



```

from transformers import AutoTokenizer
tok_len = AutoTokenizer.from_pretrained("thenlper/gte-small")

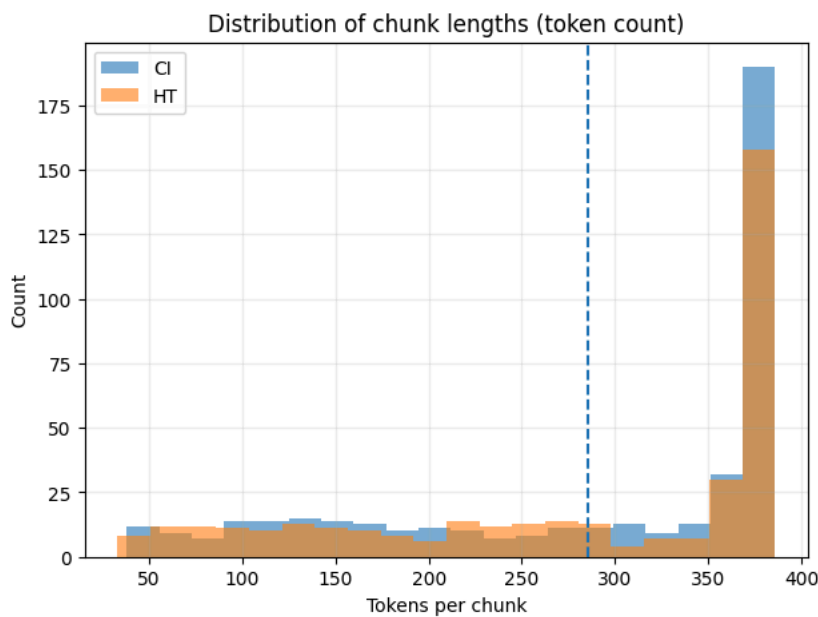
```

```
def chunk_token_lengths(vdb, sample_max=None):
    texts = []
    for _id, doc in vdb.docstore._dict.items():
        if doc and doc.page_content:
            texts.append(doc.page_content)
            if sample_max and len(texts) >= sample_max:
                break
    return [len(tok_len(t).input_ids) for t in texts]

lens_ci = chunk_token_lengths(ci_db)
lens_ht = chunk_token_lengths(ht_db)

import numpy as np, matplotlib.pyplot as plt
plt.figure(figsize=(7,5))
plt.hist(lens_ci, bins=20, alpha=0.6, label="CI")
plt.hist(lens_ht, bins=20, alpha=0.6, label="HT")
plt.axvline(np.mean(lens_ci + lens_ht), linestyle="--")
plt.title("Distribution of chunk lengths (token count)")
plt.xlabel("Tokens per chunk"); plt.ylabel("Count"); plt.legend(); plt.grid(True, alpha=0.25)
plt.show()

print(f"CI: mean={np.mean(lens_ci):.1f}, median={np.median(lens_ci):.1f}, n={len(lens_ci)}")
print(f"HT: mean={np.mean(lens_ht):.1f}, median={np.median(lens_ht):.1f}, n={len(lens_ht)}")
```



CI: mean=288.7, median=359.0, n=423
 HT: mean=281.9, median=356.0, n=373

```
import re, random, json, pandas as pd

EVAL_JSONL = "/content/stats_eval_questions_only.jsonl copy"

KS = (4, 6, 8, 10)
SAMPLE_N = 25

PATTERNS = [
    re.compile(r"\bconfidence\s+interval\b", re.I),
    re.compile(r"\breject\b.*\b(h0|h[1-9]?|h[0-9]|null)\b", re.I),
    re.compile(r"\bfail\s+to\s+reject\b", re.I),
    re.compile(r"\bhypothesis(es)\b", re.I),
    re.compile(r"\bp[-\s]?value\b", re.I),
]

def load_jsonl_local(path):
    rows=[]
    with open(path, "r", encoding="utf-8") as f:
        for ln in f:
            ln=ln.strip()
            if ln: rows.append(json.loads(ln))
    return rows

def get_query_local(row):
    for k in ("question","query","prompt","text"):
        if k in row and row[k]:
            return str(row[k])
    raise ValueError("No question-like field found in row")
```

```

def signal(txt):
    t = (txt or "")
    return any(p.search(t) for p in PATTERNS)


def probe_search(vdb, query, K):
    docs = vdb.similarity_search(query, k=K)
    if not docs: return 0, 0
    top1 = (docs[0].page_content or "")
    anytxt = " ".join((d.page_content or "") for d in docs)
    return int(signal(top1)), int(signal(anytxt))

rows_eval = load_jsonl_local(EVAL_JSONL)
sample = random.sample(rows_eval, min(SAMPLE_N, len(rows_eval)))

def ksweep(vdb, label):
    out=[]
    for K in KS:
        stats = [probe(vdb, get_query_local(r), K) for r in sample]
        df = pd.DataFrame(stats, columns=["top1_signal", "any_signal"]).mean()
        out.append({"corpus": label, "K": K,
                    "top1_signal": df["top1_signal"], "any_signal": df["any_signal"]})
    return pd.DataFrame(out)

dfK = pd.concat([ksweep(ci_db, "CI"), ksweep(ht_db, "HT")], ignore_index=True)
display(dfK)

```



| | corpus | K | top1_signal | any_signal |
|---|--------|----|-------------|------------|
| 0 | CI | 4 | 0.76 | 1.0 |
| 1 | CI | 6 | 0.76 | 1.0 |
| 2 | CI | 8 | 0.76 | 1.0 |
| 3 | CI | 10 | 0.76 | 1.0 |
| 4 | HT | 4 | 0.92 | 1.0 |
| 5 | HT | 6 | 0.92 | 1.0 |
| 6 | HT | 8 | 0.92 | 1.0 |
| 7 | HT | 10 | 0.92 | 1.0 |

```

def load_index(index_dir: str):
    emb = HuggingFaceEmbeddings(
        model_name=EMBEDDING_MODEL_NAME,
        model_kwargs={"device": device},
        encode_kwargs={"normalise_embeddings": True}
    )
    return FAISS.load_local(index_dir, emb, allow_dangerous_deserialization=True)

CI_INDEX_DIR = f"{INDEX_DIR}/ci_faiss"
HT_INDEX_DIR = f"{INDEX_DIR}/ht_faiss"

EMIT_JSON = True
USE_RERANKER = False
TOP_K = 8
K_INITIAL = 30

def build_context(retrieved_docs):
    parts = []
    for i, d in enumerate(retrieved_docs):
        parts.append(f"Document {i} (source={d.metadata.get('source')}, page={d.metadata.get('page')}):\n{d.page_content}")
    return "\n\n".join(parts)

#cross-encoder reranker is OPTINONAL but it improves precision at K
#i can turn on with USE_RERANKER = True
from typing import List
def similarity_search(vdb, query: str, k: int) -> List:
    return vdb.similarity_search(query, k=k)

#enable reranking by running this cell
!pip install -q sentence-transformers
from sentence_transformers import CrossEncoder
reranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")

from typing import List

def similarity_search_with_rerank(vdb, query: str, k_initial: int = 30, k_final: int = 8) -> List:

```

```

prelim = vdb.similarity_search(query, k=k_initial)
if not prelim:
    return []
pairs = [(query, (d.page_content or "")[:1200]) for d in prelim]
scores = reranker.predict(pairs)
ranked = [
    doc for doc, _ in sorted(
        zip(prelim, scores), key=lambda t: t[1], reverse=True
    )[:k_final]
]
return ranked

```



```

config.json: 100% 794/794 [00:00<00:00, 54.1kB/s]

model.safetensors: 100% 90.9M/90.9M [00:04<00:00, 22.8MB/s]

tokenizer_config.json: 1.33k/? [00:00<00:00, 117kB/s]

vocab.txt: 232k/? [00:00<00:00, 9.39MB/s]

tokenizer.json: 711k/? [00:00<00:00, 29.0MB/s]

special_tokens_map.json: 100% 132/132 [00:00<00:00, 13.9kB/s]

README.md: 3.66k/? [00:00<00:00, 213kB/s]

```

```

import re, json

def parse_json_last_line(text: str):
    lines = [ln for ln in text.strip().splitlines() if ln.strip()]
    if not lines: return None
    last = lines[-1]
    try:
        return json.loads(last)
    except Exception:
        m = re.search(r'(\{.*\})\s*$', text, flags=re.DOTALL)
        if m:
            try: return json.loads(m.group(1))
            except Exception: return None
    return None

def strip_final_json(text: str) -> str:
    lines = text.rstrip().splitlines()
    if not lines: return text
    try:
        json.loads(lines[-1])
        return "\n".join(lines[:-1])
    except Exception:
        return text

def answer_experiment(question: str, *, vector_db, use_rag: bool, prompt_template, top_k: int = 8):
    if use_rag and vector_db is not None:
        if USE_RERANKER and 'similarity_search_with_rerank' in globals():
            retrieved = similarity_search_with_rerank(vector_db, question, k_initial=K_INITIAL, k_final=top_k)
        else:
            retrieved = similarity_search(vector_db, question, k=top_k)
        context = build_context(retrieved)
    else:
        retrieved, context = [], ""

    prompt = (prompt_template
        .replace("{context}", context)
        .replace("{question}", question))
    out = reader(prompt)[0]["generated_text"]
    parsed = parse_json_last_line(out) if EMIT_JSON else None

    prov = [{"source": d.metadata.get("source"), "page": d.metadata.get("page")} for d in retrieved]
    return {"question": question, "answer_text": out, "answer_json": parsed, "retrieved": prov}

def load_jsonl(path: str) -> list[dict]:
    rows = []
    with open(path, "r", encoding="utf-8") as f:
        for ln in f:
            ln = ln.strip()
            if ln:
                rows.append(json.loads(ln))
    return rows

def get_question_field(row: dict) -> str:

```



```

for k in ("question","query","prompt","q"):
    if k in row and row[k]: return str(row[k])
raise ValueError("No question-like key found.")

def get_id_field(row: dict):
    for k in ("question_id","id","qid","uid"):
        if k in row: return row[k]
    return None

def append_jsonl(path: str, row: dict):
    with open(path, "a", encoding="utf-8") as f:
        f.write(json.dumps(row, ensure_ascii=True) + "\n")

def select_prompt(key: str):
    return RAG_PROMPT_TEMPLATE if key == "rag" else BASE_PROMPT_TEMPLATE

experiments = [
    {"name": "ci_rag","index_dir": CI_INDEX_DIR, "use_rag": True, "prompt": "rag", "top_k": 8},
    {"name": "ht_rag", "index_dir": HT_INDEX_DIR, "use_rag": True, "prompt": "rag", "top_k": 8},
    {"name": "base_no_rag", "index_dir": None, "use_rag": False, "prompt": "base", "top_k": 0},
]

def run_all_experiments(eval_jsonl_path: str):
    data = load_jsonl(eval_jsonl_path)
    for exp in experiments:
        vdb = load_index(exp["index_dir"]) if exp["index_dir"] else None
        tpl = select_prompt(exp["prompt"])
        out_path = f"{RESULTS_DIR}/{exp['name']}_{RUN_TAG}.jsonl"
        open(out_path, "w", encoding="utf-8").close()

        for row in data:
            q = get_question_field(row)
            qid = get_id_field(row)
            res = answer_experiment(q, vector_db=vdb, use_rag=exp["use_rag"], prompt_template=tpl, top_k=exp["top_k"])
            append_jsonl(out_path, {
                "id": qid, "experiment": exp["name"],
                "question": q,
                "answer_text": res["answer_text"],
                "answer_json": res["answer_json"],
                "retrieved": res["retrieved"]
            })
        print(f"[{exp['name']}] saved -> {out_path}")

EVAL_JSONL = "/content/stats_eval_questions_only.jsonl"
run_all_experiments(EVAL_JSONL)

!ls -lh "/content/drive/MyDrive/llm-evals"

from google.colab import files
BUNDLE = "/content/rag_results_bundle.zip"
!zip -j "$BUNDLE" /content/drive/MyDrive/llm-evals/*_{RUN_TAG}.jsonl 2>/dev/null
files.download(BUNDLE)

```

10/12

11/12

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.