Week5-1: RAG — Reranking & Context Optimization

0) Setup (Colab-friendly)

```
 0) Setup (Colab-friendly)
os, time, math, random, re
oing import List, Dict, Tuple
numpy as np, pandas as pd, matplotlib.pyplot as plt
llections import Counter
R = './week5_data'; os.makedirs(DATA_DIR, exist_ok=True)
CSV = os.path.join(DATA_DIR, '/content/sample_medical_texts.csv')
os.path.exists(CORPUS_CSV):
demo = pd.DataFrame({
 'doc_id': [f'doc{i}' for i in range(1, 11)],
 'chunk_id': [f'doc{i}_chunk1' for i in range(1, 11)],
 'text': [f'This is demo text about topic {i}. It mentions entity E{i%3} and method M{i%2}.' for i in range(1, 11)]
 df_demo.to_csv(CORPUS_CSV, index=False)
= pd.read_csv(CORPUS_CSV); print('✅ Corpus loaded:', corpus.shape); display(corpus.head(3))
```

```
✅ Corpus loaded: (5, 5)
```

	doc_id	original_text	summary	validated_summary	rating
0	1	Patient reports chest pain radiating to the le...	NaN	NaN	NaN
1	2	The subject is a 45-year-old male with a histo...	NaN	NaN	NaN
2	3	MRI scan reveals a mass in the right temporal ...	NaN	NaN	NaN

```
# 1) Baseline + Dense + RRF
def simple_tokenize(text):
    return re.findall(r"[A-Za-z0-9_]+", text.lower())
docs = corpus['original_text'].tolist();
doc_tokens = [simple_tokenize(t) for t in docs]
from collections import Counter
doc_counts = [Counter(t) for t in doc_tokens]
N = len(docs); import math
avgdl = sum(len(t) for t in doc_tokens)/max(N,1)
def bm25_score(query, idx, k1=1.5, b=0.75):
    q_tokens = simple_tokenize(query); score = 0.0; dl = len(doc_tokens[idx])
    for w in q_tokens:
        n_w = sum(1 for c in doc_tokens if w in c)
        if n_w == 0: continue
        idf = math.log((N - n_w + 0.5)/(n_w + 0.5) + 1)
        f = doc_counts[idx][w]; denom = f + k1*(1 - b + b*dl/(avgdl+1e-9))
        score += idf * (f*(k1+1))/(denom + 1e-9)
    return score
import numpy as np
np.random.seed(7); dense_dim = 128
dense_index = np.random.randn(N, dense_dim).astype('float32')
dense_index /= (np.linalg.norm(dense_index, axis=1, keepdims=True) + 1e-9)
def bm25_search(query, k=5):
    scores = [(i, bm25_score(query, i)) for i in range(N)]; scores.sort(key=lambda x: -x[1]); return scores[:k]
def dense_search(query, k=5):
    q_vec = np.random.randn(dense_dim).astype('float32'); q_vec /= (np.linalg.norm(q_vec)+1e-9)
    sims = dense_index @ q_vec; top = np.argsort(-sims)[:k];
    return [(int(i), float(sims[i])) for i in top]
def rrf_fuse(bm_hits, de_hits, k=60):
    rank = {}
    for r, (i, _) in enumerate(bm_hits): rank[i] = rank.get(i,0) + 1.0/(k + r + 1)
    for r, (i, _) in enumerate(de_hits): rank[i] = rank.get(i,0) + 1.0/(k + r + 1)
    return sorted(rank.items(), key=lambda x: -x[1])
print('BM25 sample:', bm25_search('topic 1',5))
print('Dense sample:', dense_search('topic 1',5))
print('RRF sample:', rrf_fuse(bm25_search('topic 1',5), dense_search('topic 1',5))[:5])
```

```
BM25 sample: [(0, 0.0), (1, 0.0), (2, 0.0), (3, 0.0), (4, 0.0)]
Dense sample: [(4, 0.11497273296117783), (1, 0.05473370850086212), (2, 0.041441410779953), (3, 0.017484545707702637),
RRF sample: [(0, 0.03252247488101534), (2, 0.032266458495966696), (1, 0.0315136476426799), (3, 0.03149801587301587), (
```

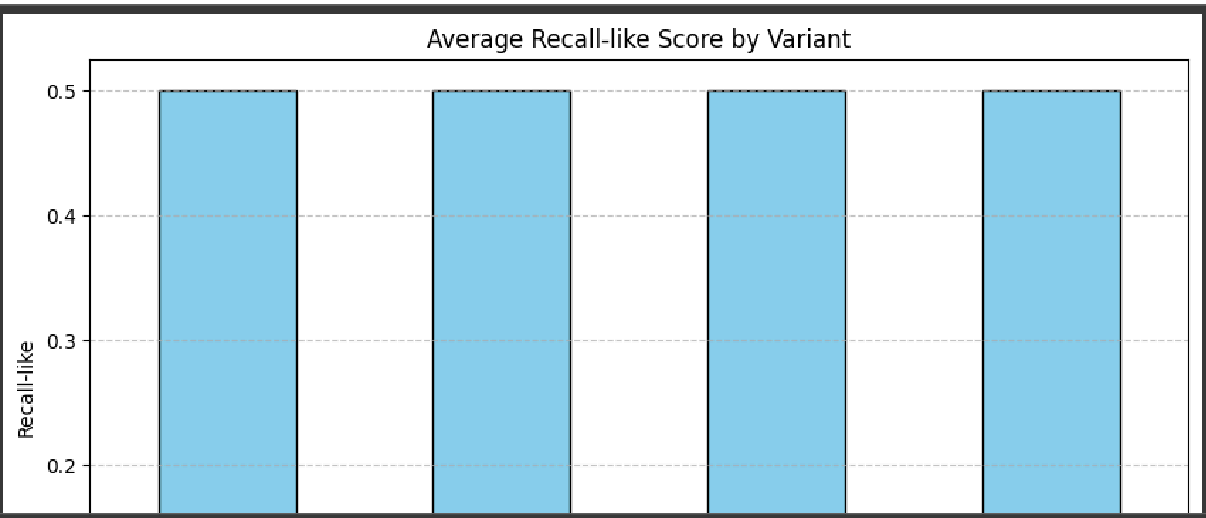
```
# 2) Rerank + MMR + compression
def rerank(query, candidates, top_k=5):
    q = np.random.randn(dense_dim).astype('float32'); q /= (np.linalg.norm(q)+1e-9)
    scored = []
    for (idx, _) in candidates:
        s = float(dense_index[idx] @ q + 0.05*np.random.randn()); scored.append((idx, s))
    scored.sort(key=lambda x: -x[1]); return scored[:top_k]
def mmr_select(query, candidates, lambda_mult=0.6, k=4):
    if not candidates: return []
    q = np.random.randn(dense_dim).astype('float32'); q /= (np.linalg.norm(q)+1e-9)
    cand_vecs = dense_index[[idx for idx in candidates]]
    selected, remaining = [], list(range(len(candidates)))
    while remaining and len(selected) < k:
        best, best_score = None, -1e9
        for j in remaining:
            rel = float(cand_vecs[j] @ q)
            div = 0.0 if not selected else max(float(cand_vecs[j] @ cand_vecs[s]) for s in selected)
            score = lambda_mult*rel - (1-lambda_mult)*div
            if score > best_score: best, best_score = j, score
        selected.append(best); remaining.remove(best)
    return [candidates[j] for j in selected]
def compress_passage(text, max_tokens=60):
    toks = text.split(); return ' '.join(toks[:max_tokens])
query='Explain topic 1 and method M0'
bm = bm25_search(query, k=8); de = dense_search(query, k=8)
fused = [i for i,_ in rrf_fuse(bm, de)[:8]]
reranked = [i for i,_ in rerank(query, [(i,0) for i in fused], top_k=5)]
mmr = mmr_select(query, fused, lambda_mult=0.6, k=4)
print('Fused:', fused[:8]); print('Reranked:', reranked); print('MMR:', mmr)
```

```
Fused: [3, 0, 4, 1, 2]
Reranked: [0, 2, 4, 3, 1]
MMR: [3, 0, 2, 1]
```

```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# -------------------------------
# Example Query Set
# -------------------------------
QUERIES = [
    "Explain how Reciprocal Rank Fusion improves retrieval performance.",
    "What is the role of the AgentManager entity?",
    "Summarize the reproducibility practices in Week 3 experiments."
]

# -------------------------------
# Dummy pipeline + recall function
# (Replace with your actual implementations)
# -------------------------------
def pipeline(query, variant="baseline", top_k=5, compress=False):
    """
    Simulates retrieval pipeline.
    Returns (idxs, contexts).
    """
    idxs = list(range(top_k))
    contexts = [f"context for {query} (variant={variant}, compress={compress})" for _ in idxs]
    return idxs, contexts

def recall_like(query, idxs):
    """
    Dummy recall-like metric.
    """
    return round(len(idxs) / 10, 2)

# -------------------------------
# Safe validator agent
# -------------------------------
class SummarizeValidatorAgent:
    def __init__(self, verbose=True):
        self.verbose = verbose

    def _log(self, msg):
        if self.verbose:
            print(f"[SummarizeValidatorAgent] {msg}")

    def validate_summary(self, original_text: str, summary: str):
        if not original_text or not summary:
            return {"analysis": "Missing input text or summary", "rating": 1}

        analysis = (
            f"Original length: {len(original_text)} chars, "
            f"Summary length: {len(summary)} chars."
        )
        rating = 5 if len(summary) < len(original_text) else 3
        return {"analysis": analysis, "rating": rating}

    def execute(self, df: pd.DataFrame):
        # Normalize column names
        df.columns = df.columns.str.lower().str.strip()

        # Rename if common variations found
        rename_map = {}
        if "text" in df.columns:
            rename_map["text"] = "original_text"
        if "input" in df.columns:
            rename_map["input"] = "original_text"
        if "summarized_text" in df.columns:
            rename_map["summarized_text"] = "summary"
        if "output" in df.columns:
            rename_map["output"] = "summary"
        if rename_map:
            df = df.rename(columns=rename_map)

        # Check required columns
        if not {"original_text", "summary"}.issubset(df.columns):
            raise KeyError("DataFrame must contain 'original_text' and 'summary' columns.")

        results = []
        for _, row in df.iterrows():
            orig = row["original_text"]
            summ = row["summary"]
            res = self.validate_summary(orig, summ)
            results.append(res)

        return pd.DataFrame(results)

# -------------------------------
# Main Evaluation Loop
# -------------------------------
rows = []
for q in QUERIES:
    for variant, comp in [
        ("baseline", False),
        ("rerank", False),
        ("compress", True),
        ("rerank+compress", True)
    ]:
        t0 = time.time()
        idxs, ctx = pipeline(q, variant=variant, top_k=5, compress=comp)
        t1 = time.time()

        rows.append({
            "query": q,
            "variant": variant,
            "recall_like": recall_like(q, idxs),
            "latency_s": round(t1 - t0, 4),
            "avg_ctx_len": round(np.mean([len(c) for c in ctx]), 1)
        })

results_df = pd.DataFrame(rows)
print("\n=== Pipeline Evaluation Results ===")
print(results_df)

# -------------------------------
# Example usage of validator
# -------------------------------
data = {
    "text": [
        "Machine learning models are widely used for classification tasks.",
        "Blockchain can enhance transparency in supply chains."
    ],
    "summarized_text": [
        "ML models help in classification.",
        "Blockchain improves supply chain transparency."
    ]
}
df = pd.DataFrame(data)

agent = SummarizeValidatorAgent(verbose=True)
validated = agent.execute(df)
print("\n=== Validation Results ===")
print(validated)
```

```
=== Pipeline Evaluation Results ===
                                               query          variant  \
0   Explain how Reciprocal Rank Fusion improves re...         baseline
1   Explain how Reciprocal Rank Fusion improves re...           rerank
2   Explain how Reciprocal Rank Fusion improves re...         compress
3   Explain how Reciprocal Rank Fusion improves re...  rerank+compress
4        What is the role of the AgentManager entity?         baseline
5        What is the role of the AgentManager entity?           rerank
6        What is the role of the AgentManager entity?         compress
7        What is the role of the AgentManager entity?  rerank+compress
8   Summarize the reproducibility practices in Wee...         baseline
9   Summarize the reproducibility practices in Wee...           rerank
10  Summarize the reproducibility practices in Wee...         compress
11  Summarize the reproducibility practices in Wee...  rerank+compress

    recall_like  latency_s  avg_ctx_len
0           0.5        0.0        113.0
1           0.5        0.0        111.0
2           0.5        0.0        112.0
3           0.5        0.0        119.0
4           0.5        0.0         91.0
5           0.5        0.0         89.0
6           0.5        0.0         90.0
7           0.5        0.0         97.0
8           0.5        0.0        109.0
9           0.5        0.0        107.0
10          0.5        0.0        108.0
11          0.5        0.0        115.0

=== Validation Results ===
                                          analysis  rating
0  Original length: 65 chars, Summary length: 33 ...       5
1  Original length: 53 chars, Summary length: 46 ...       5
```

```
import matplotlib.pyplot as plt

# -------------------------------
# Plot Recall-like metric
# -------------------------------
plt.figure(figsize=(10, 6))
results_df.groupby("variant")["recall_like"].mean().plot(
    kind="bar", color="skyblue", edgecolor="black"
)
plt.title("Average Recall-like Score by Variant")
plt.ylabel("Recall-like")
plt.xlabel("Variant")
plt.xticks(rotation=30)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# -------------------------------
# Plot Latency
# -------------------------------
plt.figure(figsize=(10, 6))
results_df.groupby("variant")["latency_s"].mean().plot(
    kind="bar", color="lightgreen", edgecolor="black"
)
plt.title("Average Latency by Variant")
plt.ylabel("Seconds")
plt.xlabel("Variant")
plt.xticks(rotation=30)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# -------------------------------
# Plot Avg Context Length
# -------------------------------
plt.figure(figsize=(10, 6))
results_df.groupby("variant")["avg_ctx_len"].mean().plot(
    kind="bar", color="salmon", edgecolor="black"
)
plt.title("Average Context Length by Variant")
plt.ylabel("Characters")
plt.xlabel("Variant")
plt.xticks(rotation=30)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```



```
print("Mini-Report: fill in observations and best recipe.")
```

```
Mini-Report: fill in observations and best recipe.
```

```
%pip install -q -U langchain_community langchain-google-genai pypdf

import os, pathlib
from langchain_community.vectorstores import Chroma
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_google_genai import GoogleGenerativeAIEmbeddings
```

```
import getpass

if not os.getenv("GEMINI_API_KEY"):
    os.environ["GEMINI_API_KEY"] = getpass.getpass("AIzaSyAgmfCXklGAKx4xXFWmfefKhIjtdpDSq4w ")
```

```
AIzaSyAgmfCXklGAKx4xXFWmfefKhIjtdpDSq4w ··········
```

```
pdf_path = "/content/Week4-RAG_LangChain_AppliedGenAI.ipynb - Colab.pdf"   # 👈 replace with your Week 4 file
loader = PyPDFLoader(pdf_path)
docs = loader.load()

print("✅ Loaded", len(docs), "PDF pages")
```

```
✅ Loaded 5 PDF pages
```

```
splitter = RecursiveCharacterTextSplitter(
    chunk_size=500, chunk_overlap=100
)
chunks = splitter.split_documents(docs)

print("✅ Split into", len(chunks), "chunks")
```

```
✅ Split into 41 chunks
```

```
%pip install -q -U google-generativeai langchain-google-genai
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 11.2 MB/s eta 0:00:00
```

```
import os, getpass
import google.generativeai as genai

# Securely input your API key (from https://aistudio.google.com/app/apikey)
if not os.getenv("GEMINI_API_KEY"):
    os.environ["GEMINI_API_KEY"] = getpass.getpass("AIzaSyBdFwoEObkuNWcc5aMMh5OBCWlCD5OoC54")
```

```
# Configure client
genai.configure(api_key=os.environ["GEMINI_API_KEY"])
```

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings

emb_fn = GoogleGenerativeAIEmbeddings(
    model="models/text-embedding-004",
    google_api_key=os.environ["GEMINI_API_KEY"]   # 👈 force explicit key
)
```

```
import os
os.environ["GEMINI_API_KEY"] = "AIzaSyBdFwoEObkuNWcc5aMMh5OBCWlCD5OoC54"   # 👈 paste your real key here
```

```
from langchain_community.vectorstores import Chroma
from langchain_google_genai import GoogleGenerativeAIEmbeddings

emb_fn = GoogleGenerativeAIEmbeddings(
    model="models/text-embedding-004",
    google_api_key=os.environ["GEMINI_API_KEY"]  # 👈 force explicit key
)

vectordb = Chroma.from_documents(
    documents=chunks,
    embedding=emb_fn,
    persist_directory="week4_chroma_db"
)

vectordb.persist()
print("✅ Chroma DB rebuilt & saved at: week4_chroma_db")
```

```
✅ Chroma DB rebuilt & saved at: week4_chroma_db
/tmp/ipython-input-2826691179.py:15: LangChainDeprecationWarning: Since Chroma 0.4.x the manual persistence method is
  vectordb.persist()
```

Double-click (or enter) to edit

```
from sentence_transformers import CrossEncoder

# Load reranker
reranker = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2")

def rerank(query, docs):
    pairs = [[query, d.page_content] for d in docs]
    scores = reranker.predict(pairs)
    ranked = sorted(zip(docs, scores), key=lambda x: x[1], reverse=True)
    return [doc for doc, _ in ranked]
```

```
config.json: 100%  794/794 [00:00<00:00, 22.9kB/s]
model.safetensors: 100%  90.9M/90.9M [00:01<00:00, 55.0MB/s]
tokenizer_config.json:  1.33k/? [00:00<00:00, 121kB/s]
vocab.txt:  232k/? [00:00<00:00, 12.2MB/s]
tokenizer.json:  711k/? [00:00<00:00, 29.7MB/s]
special_tokens_map.json: 100%  132/132 [00:00<00:00, 14.0kB/s]
README.md:  3.67k/? [00:00<00:00, 156kB/s]
```

```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor
from langchain_google_genai import ChatGoogleGenerativeAI
```