

lab_answer

April 3, 2026

1 Lab 09 — Advanced RAG (ANSWERS)

Upgrade your RAG pipeline with hybrid search, reranking, and evaluation.

Exercises: 1. Hybrid search with BM25 + reciprocal rank fusion 2. Cross-encoder reranking 3. LLM-as-judge faithfulness evaluation

```
[ ]: # Setup
import os
import json
import numpy as np
from pathlib import Path
from dotenv import load_dotenv
import anthropic
import chromadb
from sentence_transformers import SentenceTransformer, CrossEncoder
from rank_bm25 import BM25Okapi

load_dotenv()
client = anthropic.Anthropic()
MODEL = "claude-sonnet-4-20250514"
embed_model = SentenceTransformer("all-MiniLM-L6-v2")
chroma_client = chromadb.Client()
print("Setup complete.")
```

```
[ ]: # Load documents (reuse from Lesson 08)
docs_dir = Path("../lesson08-rag-basics/demo/sample_docs")
doc_files = sorted(docs_dir.glob("*.txt"))
documents = [f.read_text(encoding="utf-8") for f in doc_files]
doc_names = [f.stem for f in doc_files]
print(f"Loaded {len(documents)} documents")

# Create ChromaDB collection
embeddings = embed_model.encode(documents).tolist()
collection = chroma_client.create_collection("python_docs")
collection.add(
    ids=doc_names,
    documents=documents,
```

```

        embeddings=embeddings,
    )
    print(f"ChromaDB collection: {collection.count()} docs")

```

1.1 Exercise 1 — Hybrid Search with BM25

```

[ ]: # Create BM25 index
tokenized_corpus = [doc.lower().split() for doc in documents]
bm25 = BM25Okapi(tokenized_corpus)

def bm25_search(query: str, k: int = 5) -> list[int]:
    """Return top-k document indices by BM25 score."""
    scores = bm25.get_scores(query.lower().split())
    top_indices = scores.argsort()[::-1][:k]
    return top_indices.tolist()

def dense_search(query: str, collection, k: int = 5) -> list[int]:
    """Return top-k document indices by dense embedding similarity."""
    query_emb = embed_model.encode([query]).tolist()
    results = collection.query(query_embeddings=query_emb, n_results=k)
    # Map ChromaDB IDs back to indices
    return [doc_names.index(doc_id) for doc_id in results["ids"][0]]

def reciprocal_rank_fusion(ranked_lists: list[list[int]], k: int = 60) -> list[int]:
    """Merge multiple ranked lists using RRF. Returns fused ranking."""
    scores = {}
    for ranked_list in ranked_lists:
        for rank, doc_idx in enumerate(ranked_list):
            if doc_idx not in scores:
                scores[doc_idx] = 0.0
            scores[doc_idx] += 1.0 / (k + rank)
    # Sort by fused score descending
    return sorted(scores.keys(), key=lambda x: scores[x], reverse=True)

def hybrid_search(query: str, k: int = 5) -> list[int]:
    """Combine BM25 and dense search via reciprocal rank fusion."""
    bm25_results = bm25_search(query, k=k)
    dense_results = dense_search(query, collection, k=k)
    fused = reciprocal_rank_fusion([bm25_results, dense_results])
    return fused[:k]

```

```
[ ]: # Test Exercise 1
print("Keyword query ('lru_cache decorator'):")
for idx in hybrid_search("lru_cache decorator", k=3):
    print(f"  {doc_names[idx]}")

print("\nSemantic query ('how to measure code performance'):")
for idx in hybrid_search("how to measure code performance", k=3):
    print(f"  {doc_names[idx]}")
```

1.2 Exercise 2 — Cross-Encoder Reranking

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

def rerank(query: str, doc_indices: list[int], top_k: int = 3) -> list[int]:
    """Rerank candidate documents using cross-encoder. Return top_k indices."""
    if not doc_indices:
        return []
    # Build (query, document) pairs for the cross-encoder
    pairs = [(query, documents[idx]) for idx in doc_indices]
    scores = reranker.predict(pairs)
    # Sort indices by cross-encoder score descending
    scored = sorted(zip(doc_indices, scores), key=lambda x: x[1], reverse=True)
    return [idx for idx, _ in scored[:top_k]]

def advanced_search(query: str, retrieve_k: int = 10, final_k: int = 3) -> list[int]:
    """Hybrid retrieval + cross-encoder reranking."""
    candidates = hybrid_search(query, k=retrieve_k)
    return rerank(query, candidates, top_k=final_k)
```

```
[ ]: # Test Exercise 2
query = "How do I create a custom dictionary with default values?"
print("Advanced search results:")
for idx in advanced_search(query, retrieve_k=10, final_k=3):
    print(f"  {doc_names[idx]}: {documents[idx][:60]}...")
```

1.3 Exercise 3 — LLM-as-Judge Evaluation

```
[ ]: # 5 test questions
test_questions = [
    "How do I read a JSON configuration file in Python?",
    "What is a defaultdict and when should I use it?",
    "How can I cache expensive function results?",
    "How do I list all Python files in a directory recursively?",
    "What are the different logging levels in Python?",
]

def rag_answer(question: str, doc_indices: list[int]) -> tuple[str, str]:
    """Generate answer from retrieved docs. Returns (answer, context)."""
    context = "\n\n---\n\n".join(documents[i] for i in doc_indices)
    prompt = f"""Use the following context to answer the question.
    If the context does not contain enough information, say so.

    Context:
    {context}

    Question: {question}"""
    response = client.messages.create(
        model=MODEL, max_tokens=512,
        messages=[{"role": "user", "content": prompt}],
    )
    return response.content[0].text, context

def judge_faithfulness(question: str, context: str, answer: str) -> int:
    """Ask Claude to rate faithfulness 1-5."""
    judge_prompt = f"""Rate the faithfulness of this answer on a scale of 1-5.
    1 = completely unfaithful (hallucinated or contradicts context)
    5 = completely faithful (every claim is supported by context)

    Context: {context[:2000]}
    Question: {question}
    Answer: {answer}

    Respond with ONLY a single integer (1-5)."""
    response = client.messages.create(
        model=MODEL, max_tokens=10,
        messages=[{"role": "user", "content": judge_prompt}],
    )
    try:
        return int(response.content[0].text.strip()[0])
    except (ValueError, IndexError):
```

```

    return 3 # fallback

# Run evaluation
print(f"{'Question':<55} {'Naive':>6} {'Advanced':>9}")
print("-" * 72)

for q in test_questions:
    # Naive: dense search only
    naive_indices = dense_search(q, collection, k=3)
    naive_ans, naive_ctx = rag_answer(q, naive_indices)
    naive_score = judge_faithfulness(q, naive_ctx, naive_ans)

    # Advanced: hybrid + rerank
    adv_indices = advanced_search(q, retrieve_k=10, final_k=3)
    adv_ans, adv_ctx = rag_answer(q, adv_indices)
    adv_score = judge_faithfulness(q, adv_ctx, adv_ans)

    print(f"{q[:53]:<55} {naive_score:>5}/5 {adv_score:>7}/5")

```