Phase 1

In [1]:
```
!pip install faiss-cpu sentence-transformers
```

```
Collecting faiss-cpu
  Downloading faiss_cpu-1.13.2-cp310-abi3-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (7.6
kB)
Requirement already satisfied: sentence-transformers in
/usr/local/lib/python3.12/dist-packages (5.2.2)
Requirement already satisfied: numpy<3.0,>=1.25.0 in
/usr/local/lib/python3.12/dist-packages (from faiss-cpu)
(2.0.2)
Requirement already satisfied: packaging in
/usr/local/lib/python3.12/dist-packages (from faiss-cpu)
(26.0)
Requirement already satisfied: transformers<6.0.0,>=4.41.0 in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (5.0.0)
Requirement already satisfied: huggingface-hub>=0.20.0 in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (1.4.0)
Requirement already satisfied: torch>=1.11.0 in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (2.9.0+cu128)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (1.6.1)
Requirement already satisfied: scipy in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (1.16.3)
Requirement already satisfied: typing_extensions>=4.5.0 in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (4.15.0)
Requirement already satisfied: tqdm in
/usr/local/lib/python3.12/dist-packages (from sentence-
transformers) (4.67.3)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (3.20.3)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (2025.3.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (1.2.0)
Requirement already satisfied: httpx<1,>=0.23.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (0.28.1)
```

```
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (6.0.3)
Requirement already satisfied: shellingham in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (1.5.4)
Requirement already satisfied: typer-slim in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub>=0.20.0->sentence-transformers) (0.21.1)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (3.6.1)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-
cu12==12.8.93 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.11.0->sentence-transformers) (12.8.93)
Requirement already satisfied: nvidia-cuda-runtime-
cu12==12.8.90 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.11.0->sentence-transformers) (12.8.90)
Requirement already satisfied: nvidia-cuda-cupti-
cu12==12.8.90 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.11.0->sentence-transformers) (12.8.90)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.8.4.1
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (12.8.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.3.83
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (11.3.3.83)
Requirement already satisfied: nvidia-curand-cu12==10.3.9.90
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (10.3.9.90)
Requirement already satisfied: nvidia-cusolver-
cu12==11.7.3.90 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.11.0->sentence-transformers) (11.7.3.90)
Requirement already satisfied: nvidia-cusparse-
cu12==12.5.8.93 in /usr/local/lib/python3.12/dist-packages
(from torch>=1.11.0->sentence-transformers) (12.5.8.93)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (3.3.20)
```

```
Requirement already satisfied: nvidia-nvtx-cu12==12.8.90 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (12.8.90)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.8.93
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (12.8.93)
Requirement already satisfied: nvidia-cufile-cu12==1.13.1.3
in /usr/local/lib/python3.12/dist-packages (from
torch>=1.11.0->sentence-transformers) (1.13.1.3)
Requirement already satisfied: triton==3.5.0 in
/usr/local/lib/python3.12/dist-packages (from torch>=1.11.0-
>sentence-transformers) (3.5.0)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from
transformers<6.0.0,>=4.41.0->sentence-transformers)
(2025.11.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from
transformers<6.0.0,>=4.41.0->sentence-transformers) (0.22.2)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.12/dist-packages (from
transformers<6.0.0,>=4.41.0->sentence-transformers) (0.7.0)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn-
>sentence-transformers) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn-
>sentence-transformers) (3.6.0)
Requirement already satisfied: anyio in
/usr/local/lib/python3.12/dist-packages (from
httpx<1,>=0.23.0->huggingface-hub>=0.20.0->sentence-
transformers) (4.12.1)
Requirement already satisfied: certifi in
/usr/local/lib/python3.12/dist-packages (from
httpx<1,>=0.23.0->huggingface-hub>=0.20.0->sentence-
transformers) (2026.1.4)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.12/dist-packages (from
httpx<1,>=0.23.0->huggingface-hub>=0.20.0->sentence-
transformers) (1.0.9)
Requirement already satisfied: idna in
/usr/local/lib/python3.12/dist-packages (from
httpx<1,>=0.23.0->huggingface-hub>=0.20.0->sentence-
transformers) (3.11)
Requirement already satisfied: h11>=0.16 in
/usr/local/lib/python3.12/dist-packages (from httpcore==1.*-
>httpx<1,>=0.23.0->huggingface-hub>=0.20.0->sentence-
transformers) (0.16.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3-
>torch>=1.11.0->sentence-transformers) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2-
>torch>=1.11.0->sentence-transformers) (3.0.3)
Requirement already satisfied: click>=8.0.0 in
/usr/local/lib/python3.12/dist-packages (from typer-slim-
>huggingface-hub>=0.20.0->sentence-transformers) (8.3.1)
```

```
Downloading faiss_cpu-1.13.2-cp310-abi3-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (23.8 MB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m
 [32m23.8/23.8 MB [0m  [31m92.3 MB/s [0m eta  [36m0:00:00 [0m
 [?25hInstalling collected packages: faiss-cpu
Successfully installed faiss-cpu-1.13.2
```

In [23]:

```
import json
import faiss
import numpy as np
from sentence_transformers import SentenceTransformer

# ========================
# CONFIG
# ========================
DATA_PATH = "/content/rag_chunks_engineered_v2.jsonl"
INDEX_PATH = "faiss_index_v2.bin"
METADATA_PATH = "metadata_v2.json"

CHUNK_SIZE = 900
CHUNK_OVERLAP = 200

# ========================
# LOAD MODEL (MXBAI LARGE)
# ========================
e_model = SentenceTransformer("mixedbread-ai/mxbai-embed-large-
v1")

# ========================
# CHUNKING FUNCTION
# ========================
def chunk_text(text, size=900, overlap=200):
    chunks = []
    start = 0
    while start < len(text):
        end = min(start + size, len(text))
        chunks.append(text[start:end])
        start += size - overlap
    return chunks

documents = []
metadata = []

# ========================
# LOAD + CHUNK DATA
# ========================
with open(DATA_PATH, "r") as f:
    for line in f:
        obj = json.loads(line)

        if obj["doc_type"] != "table_of_cover":
            continue

        full_text = obj["text"]
        plan_name = obj.get("plan_name", "Unknown Plan")
        insurer = obj.get("insurer", "Unknown Insurer")
        doc_id = obj.get("doc_id")

        chunks = chunk_text(full_text, CHUNK_SIZE, CHUNK_OVERLAP)

        for i, chunk in enumerate(chunks):

            enriched_chunk = (
                f"[Insurer: {insurer}]\n"
```

```
                f"[Plan: {plan_name}]\n"
                f"[Chunk: {i}]\n\n"
                f"{chunk}"
            )

            documents.append(enriched_chunk)

            metadata.append({
                "doc_id": doc_id,
                "insurer": insurer,
                "plan_name": plan_name,
                "chunk_id": i,
                "chunk_text": chunk,
                "full_text": full_text
            })

print(f"Total chunks indexed: {len(documents)}")

# =========================
# CREATE EMBEDDINGS
# =========================
embeddings = e_model.encode(
    documents,
    batch_size=16,
    show_progress_bar=True,
    normalize_embeddings=True
)

embeddings = np.array(embeddings).astype("float32")

# =========================
# BUILD FAISS INDEX (Cosine)
# =========================
dimension = embeddings.shape[1]
index = faiss.IndexFlatIP(dimension)
index.add(embeddings)

faiss.write_index(index, INDEX_PATH)

with open(METADATA_PATH, "w") as f:
    json.dump(metadata, f)

print("🔥 FAISS v2 index built successfully.")

# ============================================================
# SEARCH FUNCTION (Parent-Plan Aggregation)
# ============================================================
def search(query, k=5):
    # MXBAI works better with query prefix
    prefixed_query = "Represent this sentence for searching
relevant passages: " + query

    query_embedding = e_model.encode(
        [prefixed_query],
        normalize_embeddings=True
    )
```

```
    query_embedding = np.array(query_embedding).astype("float32")

    scores, indices = index.search(query_embedding, k)

    results = []
    seen_plans = set()

    for idx in indices[0]:
        plan_name = metadata[idx]["plan_name"]

        if plan_name not in seen_plans:
            seen_plans.add(plan_name)

            results.append({
                "plan_name": plan_name,
                "insurer": metadata[idx]["insurer"],
                "matched_chunk": metadata[idx]["chunk_text"],
                "full_text": metadata[idx]["full_text"]
            })

    return results


# =========================
# TEST QUERY
# =========================
query = """
Coverage for cardiac treatment, frequent hospital visits,
consultant fees, medication support,
and minimal excess for inpatient admission.
"""

results = search(query)

for r in results:
    print("Plan:", r["plan_name"])
    print("Insurer:", r["insurer"])
    print("Matched Section Preview:")
    print(r["matched_chunk"][:500])
    print("=" * 80)
```

```
Loading weights:   0%|          | 0/391 [00:00<?, ?it/s]

Total chunks indexed: 212

Batches:   0%|          | 0/14 [00:00<?, ?it/s]

🔥 FAISS v2 index built successfully.
Plan: Plan C (Table of Cover)
Insurer: Level Health
Matched Section Preview:
or the bed that you are in while in hospital. It includes
cover for in-patient (overnight stay) treatments and day case
(admitted and discharged on the same day) services. Covered
at semi-private rate means that if you choose accommodation
in a private room, we will cover up to the rate of a semi-
```

private room and you pay the rest. A full list of all approved hospitals can be found here. QuickClaims All benefits are per policy year unless otherwise stated. For full terms and conditions, read your

==========

Plan: Inspire (Table of Benefits)
Insurer: Laya Healthcare
Matched Section Preview:
ch hospital excess €50 excess for day-case & out-patient surgical & €150 excess for semi-private or private Day-case/Out-patient surgical Full cover Semi-private Full cover in Beacon Hospital & €500 shortfall per night in Blackrock Clinic & Mater Private Dublin Private €165 shortfall in Beacon Hospital & €500 shortfall per night in Blackrock Clinic & Mater Private Dublin Cover for specialist cardiac procedures in Hi-tech hospitals Full cover - no excess Cover for specialist cardiac procedures in

==========

Plan: BeneFit
Insurer: Irish Life Health
Matched Section Preview:
nly; Subject to €100 excess per claim Listed Cardiac Procedures - - Covered subject to €200 excess per claim Listed Special Procedures - - Covered (Beacon Only) subject to €200 excess (first two claims per policy) subject to €2,000 copayment on certain orthopaedic procedures Maternity Public hospital cover for maternity 3 nights accommodation Inpatient maternity consultant fees As per schedule of benefits for professional fees Postnatal Domestic Support 2 x 3 hour cleaning sessions Welcome Home

==========

Plan: Plan A (Table of Cover)
Insurer: Level Health
Matched Section Preview:
Table of Cover - Level Health Plan A Your Hospital Cover Public Hospitals Day case Full cover Semi-private room Full cover Private room Covered at semi-private rate Hospital Cover - Other Benefits Consultants' fees Full cover for participating consultants Psychiatric treatment Covered up to 100 days per calendar year (with any insurer) - Public hospitals only Drug, alcohol, gambling & substance abuse Covered up to 91 days every 5 years (with any insurer) - Public facilities only Convalescence ca

==========

Plan: Plan D (Table of Cover)
Insurer: Level Health
Matched Section Preview:
ist of all approved hospitals can be found here. QuickClaims All benefits are per policy year unless otherwise stated. For full terms and conditions, read your Policy Document. 1 Hospital Cover - Other Benefits Consultants' fees Full cover for participating consultants Psychiatric treatment Covered up to 100 days per calendar year (with any insurer), subject

to excess Drug, alcohol, gambling & substance abuse Covered up to 91 days every 5 years (with any insurer), subject to excess Convalescence

==========

Phase 2

In [24]:

```
import json
import numpy as np
import faiss
from collections import defaultdict
from sentence_transformers import SentenceTransformer

# ----------------------------
# CONFIG
# ----------------------------

INDEX_PATH = "faiss_index_v2.bin"
METADATA_PATH = "metadata_v2.json"
TOP_K = 3

# ----------------------------
# LOAD MODEL + INDEX
# ----------------------------

print("Loading embedding model...")
model = SentenceTransformer("mixedbread-ai/mxbai-embed-large-v1")

print("Loading FAISS index...")
index = faiss.read_index(INDEX_PATH)

print("Loading metadata...")
with open(METADATA_PATH, "r") as f:
    metadata = json.load(f)

# ----------------------------
# SEARCH FUNCTION (Chunk → Plan Aggregation)
# ----------------------------

def search(query, k=20):  # retrieve more chunks first
    prefixed_query = "Represent this sentence for searching
relevant passages: " + query

    query_embedding = model.encode(
        [prefixed_query],
        normalize_embeddings=True
    )

    query_embedding = np.array(query_embedding).astype("float32")

    scores, indices = index.search(query_embedding, k)

    # Aggregate scores per plan
    plan_scores = defaultdict(list)

    for score, idx in zip(scores[0], indices[0]):
        plan_id = metadata[idx]["doc_id"]
```

```
        plan_scores[plan_id].append(score)

    # Average score per plan
    aggregated = []
    for plan_id, score_list in plan_scores.items():
        avg_score = sum(score_list) / len(score_list)
        aggregated.append((plan_id, avg_score))

    # Sort descending
    aggregated.sort(key=lambda x: x[1], reverse=True)

    # Return top unique plans
    return [plan_id for plan_id, _ in aggregated[:TOP_K]]

# ----------------------------
# AUTO DOC_ID RESOLUTION
# ----------------------------

def get_doc_id(plan_keyword):
    matches = [
        m["doc_id"]
        for m in metadata
        if plan_keyword.lower() in m["plan_name"].lower()
    ]
    if not matches:
        raise ValueError(f"No doc_id found for keyword:
{plan_keyword}")
    return matches[0]

# ----------------------------
# BUILD EVALUATION SET
# ----------------------------

evaluation_set = [
    {
        "query": "Does Horizon 4 cover inpatient consultant
fees?",
        "expected_doc_id": get_doc_id("Horizon 4")
    },
    {
        "query": "Which plan has a €300 excess for semi-private
room admission?",
        "expected_doc_id": get_doc_id("300")
    },
    {
        "query": "How many days of psychiatric treatment are
covered under Plan A?",
        "expected_doc_id": get_doc_id("Plan A")
    },
    {
        "query": "Are inpatient scans fully covered under Health
Plan 26.1?",
        "expected_doc_id": get_doc_id("26.1")
    }
]

# ----------------------------
```

```
# METRIC COMPUTATION
# ----------------------------

correct_top1 = 0
correct_topk = 0
reciprocal_ranks = []

print("\nRunning evaluation...\n")

for item in evaluation_set:
    query = item["query"]
    expected = item["expected_doc_id"]

    returned_ids = search(query)

    print("Query:", query)
    print("Expected:", expected)
    print("Returned:", returned_ids)
    print("-" * 60)

    if returned_ids[0] == expected:
        correct_top1 += 1

    if expected in returned_ids:
        correct_topk += 1
        rank = returned_ids.index(expected) + 1
        reciprocal_ranks.append(1.0 / rank)
    else:
        reciprocal_ranks.append(0.0)

# ----------------------------
# FINAL METRICS
# ----------------------------

total = len(evaluation_set)

top1_accuracy = correct_top1 / total
topk_recall = correct_topk / total
mrr = sum(reciprocal_ranks) / total

print("\n===== FINAL METRICS =====")
print(f"Top-1 Accuracy: {top1_accuracy:.3f}")
print(f"Top-{TOP_K} Recall: {topk_recall:.3f}")
print(f"MRR: {mrr:.3f}")
```

```
Loading embedding model...

Loading weights:   0%|          | 0/391 [00:00<?, ?it/s]

Loading FAISS index...
Loading metadata...

Running evaluation...

Query: Does Horizon 4 cover inpatient consultant fees?
Expected: irish_life_health_horizon_4_table_of_cover_2026-01-
```

```
01
Returned: ['irish_life_health_horizon_4_table_of_cover_2026-
01-01',
'level_health_plan_c_table_of_cover__table_of_cover_2025-06-
27',
'level_health_plan_b_with_300_excess_table_of_cover__table_of
_cover_2025-06-27']
------------------------------------------------------------
Query: Which plan has a €300 excess for semi-private room
admission?
Expected:
level_health_plan_b_with_300_excess_table_of_cover__table_of_
cover_2025-06-27
Returned:
['level_health_plan_b_with_300_excess_table_of_cover__table_o
f_cover_2025-06-27',
'irish_life_health_horizon_4_table_of_cover_2026-01-01',
'irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01']
------------------------------------------------------------
Query: How many days of psychiatric treatment are covered
under Plan A?
Expected:
level_health_plan_a_table_of_cover__table_of_cover_2025-06-27
Returned:
['level_health_plan_a_table_of_cover__table_of_cover_2025-06-
27',
'laya_healthcare_first_family_plan_table_of_benefits_table_of
_cover_2021-02-01',
'level_health_plan_b_with_300_excess_table_of_cover__table_of
_cover_2025-06-27']
------------------------------------------------------------
Query: Are inpatient scans fully covered under Health Plan
26.1?
Expected:
irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
Returned:
['vhi_company_plan_plus_level_1_table_of_cover_2024-10-01',
'irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01',
'level_health_plan_d_table_of_cover__table_of_cover_2025-06-
27']
------------------------------------------------------------

===== FINAL METRICS =====
Top-1 Accuracy: 0.750
Top-3 Recall: 1.000
MRR: 0.875
```

Phase 3

In [4]:
```
! pip install rank-bm25
```

```
Collecting rank-bm25
  Downloading rank_bm25-0.2.2-py3-none-any.whl.metadata (3.2
kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from rank-bm25)
(2.0.2)
Downloading rank_bm25-0.2.2-py3-none-any.whl (8.6 kB)
Installing collected packages: rank-bm25
Successfully installed rank-bm25-0.2.2
```

In [25]:

```
import json
import numpy as np
import re
from collections import defaultdict
from rank_bm25 import BM25Okapi

# ----------------------------
# CONFIG
# ----------------------------

METADATA_PATH = "metadata_v2.json"
TOP_K = 3

# ----------------------------
# LOAD CHUNKED METADATA
# ----------------------------

print("Loading chunked metadata...")
with open(METADATA_PATH, "r") as f:
    metadata = json.load(f)

print(f"Using {len(metadata)} chunks")

# ----------------------------
# TOKENIZATION
# ----------------------------

def tokenize(text):
    text = text.lower()
    text = re.sub(r"[^a-z0-9€]+", " ", text)
    return text.split()

corpus = [m["chunk_text"] for m in metadata]
tokenized_corpus = [tokenize(doc) for doc in corpus]

print("Building BM25 chunk-level index...")
bm25 = BM25Okapi(tokenized_corpus)

# ----------------------------
# SEARCH (Chunk → Plan Aggregation)
# ----------------------------

def search_bm25(query, k=20):
    tokenized_query = tokenize(query)
    scores = bm25.get_scores(tokenized_query)

    # Aggregate scores per plan
    plan_scores = defaultdict(list)

    for idx, score in enumerate(scores):
        plan_id = metadata[idx]["doc_id"]
        plan_scores[plan_id].append(score)

    aggregated = []
    for plan_id, score_list in plan_scores.items():
        avg_score = sum(score_list) / len(score_list)
        aggregated.append((plan_id, avg_score))
```

```
    aggregated.sort(key=lambda x: x[1], reverse=True)

    return [plan_id for plan_id, _ in aggregated[:TOP_K]]

# ----------------------------
# DOC ID RESOLUTION
# ----------------------------

def get_doc_id(plan_keyword):
    matches = [
        m["doc_id"]
        for m in metadata
        if plan_keyword.lower() in m["plan_name"].lower()
    ]
    if not matches:
        raise ValueError(f"No doc_id found for keyword:
{plan_keyword}")
    return matches[0]

# ----------------------------
# EVALUATION SET
# ----------------------------

evaluation_set = [
    {
        "query": "Does Horizon 4 cover inpatient consultant
fees?",
        "expected_doc_id": get_doc_id("Horizon 4")
    },
    {
        "query": "Which plan has a €300 excess for semi-private
room admission?",
        "expected_doc_id": get_doc_id("300")
    },
    {
        "query": "How many days of psychiatric treatment are
covered under Plan A?",
        "expected_doc_id": get_doc_id("Plan A")
    },
    {
        "query": "Are inpatient scans fully covered under Health
Plan 26.1?",
        "expected_doc_id": get_doc_id("26.1")
    }
]

# ----------------------------
# METRICS
# ----------------------------

correct_top1 = 0
correct_topk = 0
reciprocal_ranks = []

print("\nRunning BM25 v2 evaluation...\n")
```

```
for item in evaluation_set:
    query = item["query"]
    expected = item["expected_doc_id"]

    returned_ids = search_bm25(query)

    print("Query:", query)
    print("Returned:", returned_ids)
    print("-" * 60)

    if returned_ids[0] == expected:
        correct_top1 += 1

    if expected in returned_ids:
        correct_topk += 1
        rank = returned_ids.index(expected) + 1
        reciprocal_ranks.append(1.0 / rank)
    else:
        reciprocal_ranks.append(0.0)

total = len(evaluation_set)

print("\n===== BM25 v2 =====")
print("Top-1:", correct_top1 / total)
print("Top-3:", correct_topk / total)
print("MRR:", sum(reciprocal_ranks) / total)
```

```
Loading chunked metadata...
Using 212 chunks
Building BM25 chunk-level index...

Running BM25 v2 evaluation...

Query: Does Horizon 4 cover inpatient consultant fees?
Returned:
['irish_life_health_first_cover_table_of_cover_2026-01-01',
'irish_life_health_select_starter_table_of_cover_2026-01-01',
'irish_life_health_horizon_4_table_of_cover_2026-01-01']
------------------------------------------------------------
Query: Which plan has a €300 excess for semi-private room
admission?
Returned:
['level_health_plan_b_with_300_excess_table_of_cover__table_o
f_cover_2025-06-27',
'level_health_plan_d_table_of_cover__table_of_cover_2025-06-
27',
'level_health_plan_c_table_of_cover__table_of_cover_2025-06-
27']
------------------------------------------------------------
Query: How many days of psychiatric treatment are covered
under Plan A?
Returned:
['irish_life_health_first_cover_table_of_cover_2026-01-01',
'irish_life_health_select_starter_table_of_cover_2026-01-01',
'laya_healthcare_first_family_plan_table_of_benefits_table_of
_cover_2021-02-01']
------------------------------------------------------------
```

```
Query: Are inpatient scans fully covered under Health Plan
26.1?
Returned:
['irish_life_health_select_starter_table_of_cover_2026-01-
01', 'irish_life_health_first_cover_table_of_cover_2026-01-
01', 'irish_life_health_benefit_table_of_cover_2026-01-01']
------------------------------------------------------------

===== BM25 v2 =====
Top-1: 0.25
Top-3: 0.5
MRR: 0.3333333333333333
```

Phase 4

In [26]:

```
import json
import numpy as np
import faiss
import re
from collections import defaultdict
from rank_bm25 import BM25Okapi
from sentence_transformers import SentenceTransformer

# ----------------------------
# CONFIG
# ----------------------------

INDEX_PATH = "faiss_index_v2.bin"
METADATA_PATH = "metadata_v2.json"
TOP_K = 3
ALPHA = 0.6
BETA = 0.4

# ----------------------------
# LOAD DATA
# ----------------------------

print("Loading MXBAI model...")
model = SentenceTransformer("mixedbread-ai/mxbai-embed-large-v1")

print("Loading FAISS index...")
index = faiss.read_index(INDEX_PATH)

print("Loading metadata...")
with open(METADATA_PATH, "r") as f:
    metadata = json.load(f)

print(f"Using {len(metadata)} chunks")

# ----------------------------
# TOKENIZATION
# ----------------------------

def tokenize(text):
    text = text.lower()
    text = re.sub(r"[^a-z0-9€]+", " ", text)
    return text.split()

corpus = [m["chunk_text"] for m in metadata]
tokenized_corpus = [tokenize(doc) for doc in corpus]

bm25 = BM25Okapi(tokenized_corpus)

# ----------------------------
# NORMALIZATION
```

```
# ----------------------------

def normalize(scores_dict):
    scores = np.array(list(scores_dict.values()))
    min_s = scores.min()
    max_s = scores.max()

    normalized = {}
    for k, v in scores_dict.items():
        if max_s - min_s == 0:
            normalized[k] = 0.0
        else:
            normalized[k] = (v - min_s) / (max_s - min_s)
    return normalized

# ----------------------------
# HYBRID SEARCH
# ----------------------------

def hybrid_search(query, k=TOP_K):

    # -------- DENSE --------
    prefixed_query = "Represent this sentence for searching
relevant passages: " + query

    query_embedding = model.encode(
        [prefixed_query],
        normalize_embeddings=True
    )

    query_embedding = np.array(query_embedding).astype("float32")

    dense_scores, dense_indices = index.search(query_embedding,
50)

    dense_plan_scores = defaultdict(list)

    for score, idx in zip(dense_scores[0], dense_indices[0]):
        plan_id = metadata[idx]["doc_id"]
        dense_plan_scores[plan_id].append(score)

    dense_plan_scores = {
        plan: sum(scores)/len(scores)
        for plan, scores in dense_plan_scores.items()
    }

    # -------- BM25 --------
    tokenized_query = tokenize(query)
    bm25_scores_all = bm25.get_scores(tokenized_query)

    bm25_plan_scores = defaultdict(list)

    for idx, score in enumerate(bm25_scores_all):
        plan_id = metadata[idx]["doc_id"]
        bm25_plan_scores[plan_id].append(score)

    bm25_plan_scores = {
```

```
        plan: sum(scores)/len(scores)
        for plan, scores in bm25_plan_scores.items()
    }

    # -------- NORMALIZE --------
    dense_norm = normalize(dense_plan_scores)
    bm25_norm = normalize(bm25_plan_scores)

    # -------- FUSION --------
    final_scores = {}

    all_plans = set(dense_norm.keys()) | set(bm25_norm.keys())

    for plan in all_plans:
        d = dense_norm.get(plan, 0)
        b = bm25_norm.get(plan, 0)
        final_scores[plan] = ALPHA * d + BETA * b

    ranked = sorted(final_scores.items(), key=lambda x: x[1],
reverse=True)

    return [plan_id for plan_id, _ in ranked[:k]]

# ----------------------------
# DOC ID RESOLUTION
# ----------------------------

def get_doc_id(plan_keyword):
    matches = [
        m["doc_id"]
        for m in metadata
        if plan_keyword.lower() in m["plan_name"].lower()
    ]
    if not matches:
        raise ValueError(f"No doc_id found for keyword:
{plan_keyword}")
    return matches[0]

# ----------------------------
# EVALUATION SET
# ----------------------------

evaluation_set = [
    {
        "query": "Does Horizon 4 cover inpatient consultant
fees?",
        "expected_doc_id": get_doc_id("Horizon 4")
    },
    {
        "query": "Which plan has a €300 excess for semi-private
room admission?",
        "expected_doc_id": get_doc_id("300")
    },
    {
        "query": "How many days of psychiatric treatment are
covered under Plan A?",
        "expected_doc_id": get_doc_id("Plan A")
```

```
    },
    {
        "query": "Are inpatient scans fully covered under Health
Plan 26.1?",
        "expected_doc_id": get_doc_id("26.1")
    }
]

# ----------------------------
# METRICS
# ----------------------------

correct_top1 = 0
correct_topk = 0
reciprocal_ranks = []

print("\nRunning Hybrid v2 evaluation...\n")

for item in evaluation_set:
    query = item["query"]
    expected = item["expected_doc_id"]

    returned_ids = hybrid_search(query)

    print("Query:", query)
    print("Expected:", expected)
    print("Returned:", returned_ids)
    print("-" * 60)

    if returned_ids[0] == expected:
        correct_top1 += 1

    if expected in returned_ids:
        correct_topk += 1
        rank = returned_ids.index(expected) + 1
        reciprocal_ranks.append(1.0 / rank)
    else:
        reciprocal_ranks.append(0.0)

total = len(evaluation_set)

print("\n===== HYBRID v2 RESULTS =====")
print("Top-1:", correct_top1 / total)
print("Top-3:", correct_topk / total)
print("MRR:", sum(reciprocal_ranks) / total)
```

```
Loading MXBAI model...

Loading weights:   0%|          | 0/391 [00:00<?, ?it/s]

Loading FAISS index...
Loading metadata...
Using 212 chunks

Running Hybrid v2 evaluation...
```

```
Query: Does Horizon 4 cover inpatient consultant fees?
Expected: irish_life_health_horizon_4_table_of_cover_2026-01-
01
Returned: ['irish_life_health_horizon_4_table_of_cover_2026-
01-01', 'irish_life_health_first_cover_table_of_cover_2026-
01-01', 'irish_life_health_benefit_table_of_cover_2026-01-
01']
------------------------------------------------------------
Query: Which plan has a €300 excess for semi-private room
admission?
Expected:
level_health_plan_b_with_300_excess_table_of_cover__table_of_
cover_2025-06-27
Returned:
['level_health_plan_c_table_of_cover__table_of_cover_2025-06-
27',
'level_health_plan_b_with_300_excess_table_of_cover__table_of
_cover_2025-06-27',
'level_health_plan_b_with_150_excess_table_of_cover__table_of
_cover_2025-06-27']
------------------------------------------------------------
Query: How many days of psychiatric treatment are covered
under Plan A?
Expected:
level_health_plan_a_table_of_cover__table_of_cover_2025-06-27
Returned:
['irish_life_health_select_starter_table_of_cover_2026-01-
01', 'irish_life_health_first_cover_table_of_cover_2026-01-
01', 'vhi_health_access_table_of_cover_2023-12-31']
------------------------------------------------------------
Query: Are inpatient scans fully covered under Health Plan
26.1?
Expected:
irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
Returned:
['irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01', 'vhi_company_plan_plus_level_1_table_of_cover_2024-10-
01', 'irish_life_health_select_starter_table_of_cover_2026-
01-01']
------------------------------------------------------------

===== HYBRID v2 RESULTS =====
Top-1: 0.5
Top-3: 0.75
MRR: 0.625
```

Phase 5

In [27]:

```
import json
import re
import numpy as np
import faiss
from collections import defaultdict
from sentence_transformers import SentenceTransformer

# ============================================================
# CONFIG
# ============================================================

INDEX_PATH = "faiss_index_v2.bin"
METADATA_PATH = "metadata_v2.json"

BETA = 0.6           # Risk weight
FINAL_TOP_K = 3

# ============================================================
# LOAD MODEL + INDEX
# ============================================================

print("Loading MXBAI model...")
model = SentenceTransformer("mixedbread-ai/mxbai-embed-large-v1")

print("Loading FAISS index...")
index = faiss.read_index(INDEX_PATH)

print("Loading metadata...")
with open(METADATA_PATH, "r") as f:
    metadata = json.load(f)

plan_ids = list(set(m["doc_id"] for m in metadata))
print(f"Loaded {len(plan_ids)} unique plans")

# ============================================================
# DENSE RELEVANCE (Chunk → Plan)
# ============================================================

def compute_dense_scores(query):

    prefixed_query = "Represent this sentence for searching
relevant passages: " + query

    q_emb = model.encode([prefixed_query],
normalize_embeddings=True)
    q_emb = np.array(q_emb).astype("float32")

    dense_scores, dense_indices = index.search(q_emb, 50)

    dense_plan_scores = defaultdict(list)
```

```
    for score, idx in zip(dense_scores[0], dense_indices[0]):
        plan_id = metadata[idx]["doc_id"]
        dense_plan_scores[plan_id].append(score)

    dense_plan_scores = {
        plan: np.mean(scores)
        for plan, scores in dense_plan_scores.items()
    }

    # Normalize 0–1
    if dense_plan_scores:
        vals = np.array(list(dense_plan_scores.values()))
        min_v, max_v = vals.min(), vals.max()

        dense_plan_scores = {
            k: (v - min_v) / (max_v - min_v + 1e-8)
            for k, v in dense_plan_scores.items()
        }

    return dense_plan_scores

# ========================================================
# DENSE RETRIEVAL HELPER (Plan-Specific)
# ========================================================

def retrieve_plan_chunks(plan_id, query, top_n=5):

    prefixed_query = "Represent this sentence for searching
relevant passages: " + query

    q_emb = model.encode([prefixed_query],
normalize_embeddings=True)
    q_emb = np.array(q_emb).astype("float32")

    scores, indices = index.search(q_emb, 50)

    results = []
    for score, idx in zip(scores[0], indices[0]):
        if metadata[idx]["doc_id"] == plan_id:
            results.append((score, metadata[idx]["chunk_text"]))
            if len(results) >= top_n:
                break

    return results

# ========================================================
# DISEASE RULES
# ========================================================

DISEASE_RULES = {
    "heart_disease": {
        "keywords": ["cardiac", "angioplasty", "stent"],
        "weight": 8,
        "requires_high_tech": True
    },
    "diabetes": {
```

```
        "keywords": ["diabetes", "insulin"],
        "weight": 5
    },
    "cancer": {
        "keywords": ["oncology", "chemotherapy", "radiotherapy"],
        "weight": 10,
        "requires_high_tech": True
    },
    "psychiatric_disorder": {
        "keywords": ["psychiatric", "mental health"],
        "weight": 7,
        "requires_psych_days": True
    },
    "neurological_disorder": {
        "keywords": ["neurology", "mri", "ct scan"],
        "weight": 8,
        "requires_high_tech": True
    },
    "pregnancy": {
        "keywords": ["maternity", "obstetric"],
        "weight": 8
    }
}

# ======================================================
# STRUCTURED FEATURE EXTRACTION
# ======================================================

def merge_plan_text(plan_id):
    return " ".join(
        m["chunk_text"].lower()
        for m in metadata
        if m["doc_id"] == plan_id
    )

def extract_features(text):

    features = {}

    features["full_inpatient"] = (
        "inpatient consultant fees" in text and "fully covered"
in text
    )

    excess_match = re.search(r"€\s?(\d+)\s+excess", text)
    features["excess"] = int(excess_match.group(1)) if
excess_match else 0

    copay_match = re.search(r"€\s?(\d+)[^\n]{0,50}cardiac", text)
    features["cardiac_copay"] = int(copay_match.group(1)) if
copay_match else 0

    psych_days = re.findall(r"(\d+)\s+days", text)
    features["psychiatric_days"] = max([int(x) for x in
psych_days], default=0)

    features["high_tech_available"] = (
```

```
        "high-tech hospital" in text and "not covered" not in
text
    )

    return features

# ======================================================
# DENSE-GUIDED DISEASE SCORING
# ======================================================

def dense_disease_score(plan_id, condition):

    rules = DISEASE_RULES.get(condition)
    if not rules:
        return 0

    query = f"Coverage details for {condition.replace('_',' ')}
including hospital and treatment"

    chunks = retrieve_plan_chunks(plan_id, query)

    score = 0

    for _, chunk in chunks:
        chunk = chunk.lower()

        for kw in rules["keywords"]:
            if kw in chunk:
                score += rules["weight"]

        if rules.get("requires_high_tech"):
            if "high-tech" in chunk and "not covered" not in
chunk:
                score += 6
            elif "not covered" in chunk:
                score -= 8

        if rules.get("requires_psych_days"):
            days = re.findall(r"(\d+)\s+days", chunk)
            if days:
                score += min(int(max(days)), 150) / 20

    return score

# ======================================================
# FINAL RISK SCORING
# ======================================================

def compute_risk_score(plan_id, profile):

    text = merge_plan_text(plan_id)
    f = extract_features(text)

    score = 0
    breakdown = {}

    inpatient = 8 if f["full_inpatient"] else -8
```

```
    score += inpatient
    breakdown["inpatient"] = inpatient

    excess_penalty = f["excess"] / 60
    score -= excess_penalty
    breakdown["excess_penalty"] = -round(excess_penalty, 2)

    cardiac_penalty = 0
    if "heart_disease" in profile["chronic_conditions"]:
        cardiac_penalty = f["cardiac_copay"] / 40
        score -= cardiac_penalty
    breakdown["cardiac_penalty"] = -round(cardiac_penalty, 2)

    disease_total = 0
    for condition in profile["chronic_conditions"]:
        disease_total += dense_disease_score(plan_id, condition)

    score += disease_total
    breakdown["dense_disease_score"] = round(disease_total, 2)

    return score, breakdown

# ============================================================
# MULTI-SCENARIO EVALUATION
# ============================================================

SCENARIOS = {
    "Cardiac + Diabetes": {
        "profile": {
            "age": 55,
            "chronic_conditions": ["heart_disease", "diabetes"]
        },
        "query": """
        Cardiac procedures, insulin support,
        frequent inpatient admission,
        consultant coverage, high-tech hospital.
        """
    },

    "Cancer + Neurological": {
        "profile": {
            "age": 60,
            "chronic_conditions": ["cancer",
"neurological_disorder"]
        },
        "query": """
        Oncology treatment, chemotherapy,
        MRI scans, neurological admission,
        high-tech hospital access.
        """
    },

    "Psychiatric + Diabetes": {
        "profile": {
            "age": 40,
            "chronic_conditions": ["psychiatric_disorder",
"diabetes"]
```

```
        },
        "query": """
        Psychiatric admission days,
        mental health coverage,
        diabetes management.
        """
    },

    "Pregnancy Case": {
        "profile": {
            "age": 30,
            "chronic_conditions": ["pregnancy"]
        },
        "query": """
        Maternity cover,
        obstetric services,
        hospital delivery,
        minimal excess.
        """
    }
}

print("\n====================================")
print("AUTOMATED MULTI-SCENARIO EVALUATION (DENSE ONLY)")
print("====================================")

for scenario_name, scenario_data in SCENARIOS.items():

    print(f"\n============================")
    print(f"Scenario: {scenario_name}")
    print("============================")

    patient_profile = scenario_data["profile"]
    query = scenario_data["query"]

    dense_scores = compute_dense_scores(query)

    results = []

    for plan_id in plan_ids:

        risk_score, breakdown = compute_risk_score(plan_id,
patient_profile)

        final_score = (BETA * risk_score) +
dense_scores.get(plan_id, 0)

        results.append((plan_id, final_score, risk_score,
                        dense_scores.get(plan_id, 0), breakdown))

    results.sort(key=lambda x: x[1], reverse=True)

    for rank, (doc_id, final_score, risk_score,
               dense_score, breakdown) in
enumerate(results[:FINAL_TOP_K], 1):

        print(f"\n{rank}. {doc_id}")
```

```
        print(f"   Final Score: {round(final_score,2)}")
        print(f"   Risk Score: {round(risk_score,2)}")
        print(f"   Dense Score: {round(dense_score,3)}")

        print("   Breakdown:")
        for k, v in breakdown.items():
            print(f"     {k}: {v}")

    margin = results[0][1] - results[1][1]

    print("\nConfidence Margin:", round(margin, 3))
```

```
Loading MXBAI model...

Loading weights:   0%|          | 0/391 [00:00<?, ?it/s]

Loading FAISS index...
Loading metadata...
Loaded 16 unique plans

====================================
AUTOMATED MULTI-SCENARIO EVALUATION (DENSE ONLY)
====================================

==============================
Scenario: Cardiac + Diabetes
==============================

1. irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01
   Final Score: 22.469999313354492
   Risk Score: 36.73
   Dense Score: 0.4300000071525574
   Breakdown:
     inpatient: 8
     excess_penalty: -1.25
     cardiac_penalty: -0.03
     dense_disease_score: 30

2. level_health_plan_c_table_of_cover__table_of_cover_2025-
06-27
   Final Score: 15.180000305175781
   Risk Score: 24.38
   Dense Score: 0.5569999814033508
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: -4.38
     dense_disease_score: 38

3. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 14.869999885559082
   Risk Score: 24.17
   Dense Score: 0.375
   Breakdown:
     inpatient: 8
```

```
     excess_penalty: -3.33
     cardiac_penalty: -2.5
     dense_disease_score: 22

Confidence Margin: 7.283

==============================
Scenario: Cancer + Neurological
==============================

1. vhi_company_plan_plus_level_1_table_of_cover_2024-10-01
   Final Score: 31.450000762939453
   Risk Score: 50.75
   Dense Score: 1.0
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: 0
     dense_disease_score: 60

2.
laya_healthcare_first_family_plan_table_of_benefits_table_of_
cover_2021-02-01
   Final Score: 30.940000534057617
   Risk Score: 50.0
   Dense Score: 0.9350000023841858
   Breakdown:
     inpatient: -8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 58

3. vhi_health_access_table_of_cover_2023-12-31
   Final Score: 20.0
   Risk Score: 32.0
   Dense Score: 0.7960000038146973
   Breakdown:
     inpatient: -8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 40

Confidence Margin: 0.515

==============================
Scenario: Psychiatric + Diabetes
==============================

1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Final Score: 23.670000076293945
   Risk Score: 38.55
   Dense Score: 0.5389999747276306
   Breakdown:
     inpatient: 8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 30.55
```

```
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01
   Final Score: 15.420000076293945
   Risk Score: 25.55
   Dense Score: 0.0949999988079071
   Breakdown:
     inpatient: 8
     excess_penalty: -1.25
     cardiac_penalty: 0
     dense_disease_score: 18.8

3. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 14.119999885559082
   Risk Score: 23.47
   Dense Score: 0.04500000178813934
   Breakdown:
     inpatient: 8
     excess_penalty: -3.33
     cardiac_penalty: 0
     dense_disease_score: 18.8

Confidence Margin: 8.244

==============================
Scenario: Pregnancy Case
==============================

1. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 17.700000762939453
   Risk Score: 28.67
   Dense Score: 0.4959999918937683
   Breakdown:
     inpatient: 8
     excess_penalty: -3.33
     cardiac_penalty: 0
     dense_disease_score: 24

2. irish_life_health_first_cover_table_of_cover_2026-01-01
   Final Score: 14.609999656677246
   Risk Score: 24.0
   Dense Score: 0.20600000023841858
   Breakdown:
     inpatient: 8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 16

3. level_health_plan_c_table_of_cover__table_of_cover_2025-
06-27
   Final Score: 14.119999885559082
   Risk Score: 22.75
   Dense Score: 0.46799999475479126
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: 0
```

```
     dense_disease_score: 32

Confidence Margin: 3.09
```

In [28]:

```
# ========================================================
# ADVANCED MULTI-SCENARIO EVALUATION
# ========================================================

SCENARIOS = {

    "Cardiac + Diabetes": {
        "profile": {
            "age": 55,
            "chronic_conditions": ["heart_disease", "diabetes"]
        },
        "query": """
        Cardiac procedures, insulin support,
        frequent inpatient admission,
        consultant coverage, high-tech hospital.
        """
    },

    "Cancer + Neurological": {
        "profile": {
            "age": 60,
            "chronic_conditions": ["cancer",
"neurological_disorder"]
        },
        "query": """
        Oncology treatment, chemotherapy,
        MRI scans, neurological admission,
        high-tech hospital access.
        """
    },

    "Psychiatric + Diabetes": {
        "profile": {
            "age": 40,
            "chronic_conditions": ["psychiatric_disorder",
"diabetes"]
        },
        "query": """
        Psychiatric admission days,
        mental health coverage,
        diabetes management.
        """
    },

    "Pregnancy Case": {
        "profile": {
            "age": 30,
            "chronic_conditions": ["pregnancy"]
        },
        "query": """
        Maternity cover,
        obstetric services,
        hospital delivery,
        minimal excess.
        """
    },
```

```
    "High-Risk Elderly Multi-Morbidity": {
        "profile": {
            "age": 72,
            "chronic_conditions": ["heart_disease", "cancer",
"neurological_disorder"]
        },
        "query": """
        Cardiac procedures, chemotherapy,
        MRI scans, frequent hospital admissions,
        high-tech hospital access.
        """
    },

    "Long-Term Psychiatric Intensive": {
        "profile": {
            "age": 42,
            "chronic_conditions": ["psychiatric_disorder"]
        },
        "query": """
        Extended psychiatric inpatient treatment,
        high number of covered days,
        strong mental health support.
        """
    }
}

print("\n====================================")
print("ADVANCED MULTI-SCENARIO EVALUATION (DENSE ONLY)")
print("====================================")

plan_win_counter = defaultdict(int)
plan_rank_sum = defaultdict(int)
scenario_margins = []

for scenario_name, scenario_data in SCENARIOS.items():

    print("\n==============================")
    print(f"Scenario: {scenario_name}")
    print("==============================")

    patient_profile = scenario_data["profile"]
    query = scenario_data["query"]

    dense_scores = compute_dense_scores(query)

    results = []

    for plan_id in plan_ids:

        risk_score, breakdown = compute_risk_score(plan_id,
patient_profile)

        final_score = (BETA * risk_score) +
dense_scores.get(plan_id, 0)

        results.append((plan_id, final_score, risk_score,
                        dense_scores.get(plan_id, 0), breakdown))
```

```
    results.sort(key=lambda x: x[1], reverse=True)

    # Track winner
    winner = results[0][0]
    plan_win_counter[winner] += 1

    # Track average ranks
    for rank, (doc_id, *_ ) in enumerate(results, 1):
        plan_rank_sum[doc_id] += rank

    # Print top results
    for rank, (doc_id, final_score, risk_score,
               dense_score, breakdown) in
enumerate(results[:FINAL_TOP_K], 1):

        print(f"\n{rank}. {doc_id}")
        print(f"   Final Score: {round(final_score,2)}")
        print(f"   Risk Score: {round(risk_score,2)}")
        print(f"   Dense Score: {round(dense_score,3)}")

        print("   Breakdown:")
        for k, v in breakdown.items():
            print(f"     {k}: {v}")

    margin = results[0][1] - results[1][1]
    scenario_margins.append(margin)

    print("\nConfidence Margin:", round(margin, 3))

    if margin < 1:
        print("Interpretation: Close competition")
    elif margin < 3:
        print("Interpretation: Moderate separation")
    else:
        print("Interpretation: Strong winner")

# ======================================================
# GLOBAL ANALYTICS
# ======================================================

print("\n==================================")
print("GLOBAL ANALYTICS")
print("==================================")

print("\nPlan Win Frequency:")
for plan, count in sorted(plan_win_counter.items(), key=lambda x:
x[1], reverse=True):
    print(f"{plan}: {count} wins")

print("\nAverage Rank Per Plan:")
num_scenarios = len(SCENARIOS)

avg_ranks = {
    plan: plan_rank_sum[plan] / num_scenarios
    for plan in plan_ids
}
```

```
for plan, avg_rank in sorted(avg_ranks.items(), key=lambda x:
x[1]):
    print(f"{plan}: Avg Rank = {round(avg_rank,2)}")

print("\nAverage Confidence Margin Across Scenarios:",
      round(np.mean(scenario_margins), 3))

print("Margin Std Dev:",
      round(np.std(scenario_margins), 3))
```

```
==================================
ADVANCED MULTI-SCENARIO EVALUATION (DENSE ONLY)
==================================

==============================
Scenario: Cardiac + Diabetes
==============================

1. irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01
   Final Score: 22.469999313354492
   Risk Score: 36.73
   Dense Score: 0.4300000071525574
   Breakdown:
     inpatient: 8
     excess_penalty: -1.25
     cardiac_penalty: -0.03
     dense_disease_score: 30

2. level_health_plan_c_table_of_cover__table_of_cover_2025-
06-27
   Final Score: 15.180000305175781
   Risk Score: 24.38
   Dense Score: 0.5569999814033508
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: -4.38
     dense_disease_score: 38

3. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 14.869999885559082
   Risk Score: 24.17
   Dense Score: 0.375
   Breakdown:
     inpatient: 8
     excess_penalty: -3.33
     cardiac_penalty: -2.5
     dense_disease_score: 22

Confidence Margin: 7.283
Interpretation: Strong winner

==============================
Scenario: Cancer + Neurological
```

```
==============================

1. vhi_company_plan_plus_level_1_table_of_cover_2024-10-01
   Final Score: 31.450000762939453
   Risk Score: 50.75
   Dense Score: 1.0
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: 0
     dense_disease_score: 60

2.
laya_healthcare_first_family_plan_table_of_benefits_table_of_
cover_2021-02-01
   Final Score: 30.940000534057617
   Risk Score: 50.0
   Dense Score: 0.9350000023841858
   Breakdown:
     inpatient: -8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 58

3. vhi_health_access_table_of_cover_2023-12-31
   Final Score: 20.0
   Risk Score: 32.0
   Dense Score: 0.7960000038146973
   Breakdown:
     inpatient: -8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 40

Confidence Margin: 0.515
Interpretation: Close competition

==============================
Scenario: Psychiatric + Diabetes
==============================

1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Final Score: 23.670000076293945
   Risk Score: 38.55
   Dense Score: 0.5389999747276306
   Breakdown:
     inpatient: 8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 30.55

2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01
   Final Score: 15.420000076293945
   Risk Score: 25.55
   Dense Score: 0.0949999988079071
   Breakdown:
```

```
     inpatient: 8
     excess_penalty: -1.25
     cardiac_penalty: 0
     dense_disease_score: 18.8

3. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 14.119999885559082
   Risk Score: 23.47
   Dense Score: 0.04500000178813934
   Breakdown:
     inpatient: 8
     excess_penalty: -3.33
     cardiac_penalty: 0
     dense_disease_score: 18.8

Confidence Margin: 8.244
Interpretation: Strong winner

==============================
Scenario: Pregnancy Case
==============================

1. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 17.700000762939453
   Risk Score: 28.67
   Dense Score: 0.4959999918937683
   Breakdown:
     inpatient: 8
     excess_penalty: -3.33
     cardiac_penalty: 0
     dense_disease_score: 24

2. irish_life_health_first_cover_table_of_cover_2026-01-01
   Final Score: 14.609999656677246
   Risk Score: 24.0
   Dense Score: 0.20600000023841858
   Breakdown:
     inpatient: 8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 16

3. level_health_plan_c_table_of_cover__table_of_cover_2025-
06-27
   Final Score: 14.119999885559082
   Risk Score: 22.75
   Dense Score: 0.46799999475479126
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: 0
     dense_disease_score: 32

Confidence Margin: 3.09
Interpretation: Strong winner

==============================
```

```
Scenario: High-Risk Elderly Multi-Morbidity
==============================

1. vhi_company_plan_plus_level_1_table_of_cover_2024-10-01
   Final Score: 36.029998779296875
   Risk Score: 58.75
   Dense Score: 0.781000018119812
   Breakdown:
     inpatient: -8
     excess_penalty: -1.25
     cardiac_penalty: -0.0
     dense_disease_score: 68

2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01
   Final Score: 35.68000030517578
   Risk Score: 58.73
   Dense Score: 0.4449999928474426
   Breakdown:
     inpatient: 8
     excess_penalty: -1.25
     cardiac_penalty: -0.03
     dense_disease_score: 52

3.
laya_healthcare_first_family_plan_table_of_benefits_table_of_
cover_2021-02-01
   Final Score: 33.15999984741211
   Risk Score: 54.25
   Dense Score: 0.6150000095367432
   Breakdown:
     inpatient: -8
     excess_penalty: -0.0
     cardiac_penalty: -3.75
     dense_disease_score: 66

Confidence Margin: 0.351
Interpretation: Close competition

==============================
Scenario: Long-Term Psychiatric Intensive
==============================

1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Final Score: 23.920000076293945
   Risk Score: 38.55
   Dense Score: 0.7900000214576721
   Breakdown:
     inpatient: 8
     excess_penalty: -0.0
     cardiac_penalty: 0
     dense_disease_score: 30.55

2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-
01
   Final Score: 15.680000305175781
   Risk Score: 25.55
```

```
   Dense Score: 0.3540000021457672
   Breakdown:
     inpatient: 8
     excess_penalty: -1.25
     cardiac_penalty: 0
     dense_disease_score: 18.8

3. irish_life_health_benefit_table_of_cover_2026-01-01
   Final Score: 14.430000305175781
   Risk Score: 23.47
   Dense Score: 0.35100001096725464
   Breakdown:
     inpatient: 8
     excess_penalty: -3.33
     cardiac_penalty: 0
     dense_disease_score: 18.8

Confidence Margin: 8.236
Interpretation: Strong winner

====================================
GLOBAL ANALYTICS
====================================

Plan Win Frequency:
vhi_company_plan_plus_level_1_table_of_cover_2024-10-01: 2
wins
irish_life_health_first_cover_table_of_cover_2026-01-01: 2
wins
irish_life_health_health_plan_26.1_table_of_cover_2026-01-01:
1 wins
irish_life_health_benefit_table_of_cover_2026-01-01: 1 wins

Average Rank Per Plan:
irish_life_health_health_plan_26.1_table_of_cover_2026-01-01:
Avg Rank = 2.5
irish_life_health_benefit_table_of_cover_2026-01-01: Avg Rank
= 3.83
laya_healthcare_first_family_plan_table_of_benefits_table_of_
cover_2021-02-01: Avg Rank = 5.5
level_health_plan_c_table_of_cover__table_of_cover_2025-06-
27: Avg Rank = 5.83
vhi_company_plan_plus_level_1_table_of_cover_2024-10-01: Avg
Rank = 6.67
laya_healthcare_pmi_plan_b_table_of_benefits_table_of_cover_2
021-10-01: Avg Rank = 7.17
irish_life_health_first_cover_table_of_cover_2026-01-01: Avg
Rank = 7.33
level_health_plan_d_table_of_cover__table_of_cover_2025-06-
27: Avg Rank = 7.5
vhi_health_access_table_of_cover_2023-12-31: Avg Rank = 8.67
irish_life_health_horizon_4_table_of_cover_2026-01-01: Avg
Rank = 8.67
level_health_plan_a_table_of_cover__table_of_cover_2025-06-
27: Avg Rank = 10.83
level_health_plan_b_with_300_excess_table_of_cover__table_of_
cover_2025-06-27: Avg Rank = 12.0
```

laya_healthcare_prime_plan_table_of_benefits_table_of_cover_2019-08-01: Avg Rank = 12.17
irish_life_health_select_starter_table_of_cover_2026-01-01: Avg Rank = 12.17
level_health_plan_b_with_150_excess_table_of_cover__table_of_cover_2025-06-27: Avg Rank = 12.17
laya_healthcare_inspire_table_of_benefits__table_of_cover_2025-08-01: Avg Rank = 13.0

Average Confidence Margin Across Scenarios: 4.62
Margin Std Dev: 3.433

Phase 6

In [30]:

```
import json
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# ======================================================
# CONFIG
# ======================================================

MODEL_NAME = "Qwen/Qwen2.5-3B-Instruct"
TOP_K_PLANS = 3
MAX_NEW_TOKENS = 180

# ======================================================
# STEP 1 — PREPARE RANKED RESULTS
# ======================================================

dense_scores = compute_dense_scores(query)

ranked_results = []

for plan_id in plan_ids:
    risk_score, breakdown = compute_risk_score(plan_id,
patient_profile)
    final_score = (BETA * risk_score) + dense_scores.get(plan_id,
0)

    ranked_results.append({
        "doc_id": plan_id,
        "final_score": final_score,
        "risk_score": risk_score,
        "dense_score": dense_scores.get(plan_id, 0),
        "breakdown": breakdown
    })

ranked_results.sort(key=lambda x: x["final_score"], reverse=True)

top_plans = ranked_results[:TOP_K_PLANS]

# ======================================================
# STEP 2 — BUILD STRUCTURED + DENSE EVIDENCE
# ======================================================

def build_dense_evidence(plan_id, profile):

    text = merge_plan_text(plan_id)
    features = extract_features(text)

    evidence = {
        "full_inpatient": features["full_inpatient"],
        "excess": features["excess"],
```

```
        "cardiac_copay": features["cardiac_copay"],
        "high_tech_available": features["high_tech_available"],
        "psychiatric_days": features["psychiatric_days"]
    }

    # Dense chunk evidence per condition
    retrieved_chunks = {}

    for condition in profile["chronic_conditions"]:
        condition_query = f"Coverage details for
{condition.replace('_',' ')}"
        chunks = retrieve_plan_chunks(plan_id, condition_query,
top_n=2)

        retrieved_chunks[condition] = [
            chunk_text[:200] for _, chunk_text in chunks
        ]

    evidence["retrieved_chunks"] = retrieved_chunks

    return evidence

# ======================================================
# STEP 3 — LOAD LLM (SAFE FP16 VERSION)
# ======================================================

print("\nLoading Explanation LLM...")

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

llm_model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    torch_dtype=torch.float16,
    device_map="auto"
)

llm_model.eval()

print("LLM loaded.")

# ======================================================
# STEP 4 — EXPLANATION GENERATOR
# ======================================================

def generate_explanation(rank_data, next_plan=None):

    evidence = build_dense_evidence(rank_data["doc_id"],
patient_profile)

    comparison_text = ""
    if next_plan:
        score_diff = round(
            rank_data["final_score"] - next_plan["final_score"],
2
```

```
        )

        comparison_text = f"""
Comparison With Next Ranked Plan:
Next Plan ID: {next_plan['doc_id']}
Score Difference: {score_diff}
Next Plan Risk Score: {round(next_plan['risk_score'],2)}
Next Plan Dense Score: {round(next_plan['dense_score'],3)}
"""

    prompt = f"""
You are a medical insurance ranking explanation engine.

IMPORTANT RULES:
- Use ONLY the structured evidence provided.
- Do NOT invent benefits.
- Do NOT assume coverage beyond evidence.
- Explain ranking strictly according to the scoring model.
- Mention inpatient impact, excess penalty, disease alignment,
and dense relevance.
- Clarify that ranking is model-based.
- Maximum 170 words.

PATIENT PROFILE:
{json.dumps(patient_profile)}

PLAN ID: {rank_data['doc_id']}
FINAL SCORE: {round(rank_data['final_score'],2)}
RISK SCORE: {round(rank_data['risk_score'],2)}
DENSE SCORE: {round(rank_data['dense_score'],3)}

SCORE BREAKDOWN:
{json.dumps(rank_data['breakdown'], indent=2)}

EVIDENCE:
{json.dumps(evidence, indent=2)}

{comparison_text}

Explain why this plan ranked at this position.
"""

    inputs = tokenizer(
        prompt,
        return_tensors="pt",
        truncation=True,
        padding=True
    ).to(llm_model.device)

    with torch.no_grad():
        outputs = llm_model.generate(
            **inputs,
            max_new_tokens=MAX_NEW_TOKENS,
            temperature=0.0,
            do_sample=False,
            pad_token_id=tokenizer.eos_token_id
        )
```

```
    decoded = tokenizer.decode(outputs[0],
skip_special_tokens=True)

    # Remove prompt echo if present
    if decoded.startswith(prompt):
        decoded = decoded[len(prompt):]

    return decoded.strip()

# ================================================
# STEP 5 — RUN EXPLANATIONS
# ================================================

print("\n====================================")
print("DENSE-AWARE COMPARATIVE EXPLANATIONS")
print("====================================")

for i, plan_data in enumerate(top_plans):

    next_plan = top_plans[i+1] if i+1 < len(top_plans) else None

    explanation = generate_explanation(plan_data, next_plan)

    print("\n====================================")
    print("PLAN:", plan_data["doc_id"])
    print("====================================")
    print(explanation)
```

```
Loading Explanation LLM...

Loading weights:   0%|          | 0/434 [00:00<?, ?it/s]

WARNING:accelerate.big_modeling:Some parameters are on
the meta device because they were offloaded to the cpu.
The following generation flags are not valid and may be
ignored: ['temperature', 'top_p', 'top_k']. Set
`TRANSFORMERS_VERBOSITY=info` for more details.

LLM loaded.

====================================
DENSE-AWARE COMPARATIVE EXPLANATIONS
====================================

====================================
PLAN: irish_life_health_first_cover_table_of_cover_2026-01-01
====================================
This plan ranks at position 1 due to its strong alignment
with the patient's chronic condition of psychiatric disorder,
which contributes significantly to the dense score of 0.79.
The patient profile indicates a 42-year-old with a
psychiatric disorder, making the psychiatric treatment
benefits highly relevant. Despite the low inpatient score of
```

8, the dense disease score of 30.55, heavily influenced by the psychiatric disorder, outweighs this factor. The plan does not cover psychiatric days, leading to a negative dense score of -0.79, but the overall dense score is positive due to the high relevance of psychiatric benefits. The risk score of 38.55 is relatively high, indicating potential higher costs, but the dense score and specific alignment with the patient's condition justify its placement at the top. The next ranked plan has a lower dense score and a more

==========
PLAN: irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
==========
This plan ranked 15.68 due to its strong inpatient coverage score of 8, which mitigates the risk associated with psychiatric conditions. The dense disease score of 18.8 aligns well with the patient's chronic condition of psychiatric disorder, indicating high relevance. However, it underperforms in excess penalty (-1.25), which could be beneficial for patients with higher out-of-pocket expenses. The dense relevance score of 0.354 further supports its position, emphasizing the plan's suitability for the patient's specific needs. [Structured Evidence Follows] This plan ranks at 15.68 based on several key factors derived from the structured evidence:

- **Inpatient Coverage:** The plan scores 8 for inpatient coverage, which is crucial given the patient's chronic condition of psychiatric disorder. This robust inpatient coverage helps mitigate potential financial burdens

==========
PLAN: irish_life_health_benefit_table_of_cover_2026-01-01
==========
This plan ranked at 14.43 due to its strong alignment with the patient's chronic condition of psychiatric disorder, as evidenced by the coverage for psychiatric days and the cancer support benefit. The dense relevance score of 0.351 indicates moderate coverage for mental health, which is crucial given the patient's age and condition. Despite the inpatient score of 8, the plan's dense disease score of 18.8 highlights high density in psychiatric benefits, outweighing the inpatient impact. The absence of cardiac penalties further supports its ranking, as it avoids any potential excess penalty associated with cardiac conditions. Overall, the combination of dense psychiatric benefits and moderate inpatient coverage contributed to the final score. The plan also offers high-tech benefits, but these were not scored as they did not align directly with the patient's profile. The risk score of 23.47 suggests