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) (1.3.7)
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from sentence-transformers) (2.9.0+cu126)
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) (4.67.2)
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==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers)
(12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (12.6.80)
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==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (12.5.4.2)
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==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers) (1.11.1.6)
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 [31m91.8 MB/s [0m eta [36m0:00:00 [0m
 [?25hInstalling collected packages: faiss-cpu
Successfully installed faiss-cpu-1.13.2
```

Rag.py

In [2]:
```
import json
import faiss
import numpy as np
from sentence_transformers import SentenceTransformer

DATA_PATH = "/content/rag_chunks_engineered_v2.jsonl"
INDEX_PATH = "faiss_index.bin"
METADATA_PATH = "metadata.json"

model = SentenceTransformer("all-MiniLM-L6-v2")

documents = []
metadata = []

with open(DATA_PATH, "r") as f:
    for line in f:
        obj = json.loads(line)

        # Filter only table_of_cover if desired
        if obj["doc_type"] == "table_of_cover":
            documents.append(obj["text"])
            metadata.append(obj)

embeddings = model.encode(documents, show_progress_bar=True)
embeddings = np.array(embeddings).astype("float32")

# Normalize for cosine similarity
faiss.normalize_L2(embeddings)

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 index built successfully.")

def search(query, k=3):
    query_embedding = model.encode([query])
    query_embedding = np.array(query_embedding).astype("float32")
    faiss.normalize_L2(query_embedding)

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

    results = []
    for idx in indices[0]:
```

```
    results.append(metadata[idx])

return results
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google
Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
modules.json:   0%|          | 0.00/349 [00:00<?, ?B/s]
```

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and faster downloads.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to enable higher rate limits and
faster downloads.
```

```
config_sentence_transformers.json:   0%|          | 0.00/116 [00:00<?, ?B/s]
```

```
README.md: 0.00B [00:00, ?B/s]
```

```
sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/612 [00:00<?, ?B/s]
```

```
model.safetensors:   0%|          | 0.00/90.9M [00:00<?, ?B/s]
```

```
Loading weights:   0%|          | 0/103 [00:00<?, ?it/s]
```

```
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key                     | Status     |  |
------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
```

```
tokenizer_config.json:   0%|          | 0.00/350 [00:00<?, ?B/s]
```

```
vocab.txt: 0.00B [00:00, ?B/s]
```

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

```
special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]

Batches:   0%|          | 0/1 [00:00<?, ?it/s]

FAISS index built successfully.
```

In [3]:
```
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(r["plan_name"])
    print(r["text"][:500])
    print("="*80)
```

```
Horizon 4
Horizon 4 Table of Cover effective from 1st January 2026 You should read this table of cover along with the Tailored Health Plans membership handbook effective from January 2026, which you can find on irishlifehealth.ie/more-info. The hospitals and treatment centres covered on this plan are set out in List A in Part 12 of your Tailored Health Plans membership handbook. IN PATIENT BENEFITS Hospital cover Consultants fees Covered Inpatient scans Covered Public Hospital Semi-private room Covered Pr
================================================================================
Health Plan 26.1
What you're covered for Health Plan 26.1 Effective from 1st January 2026 You should read this table of cover along with the Health Plans membership handbook effective from January 2026, which you can find on irishlifehealth.ie/more-info. The hospitals and treatment centres covered on this plan are set out in List 1 in Part 12 of your Health Plans membership handbook. In-patient Benefits Hospital Cover Inpatient Consultant fees and Inpatient Scans are fully covered Benefits Public Hospital Privat
================================================================================
Plan B (with €150 Excess) (Table of Cover)
Table of Cover - Level Health Plan B (with €150 Excess) Your Hospital Cover Public Hospitals Day case Full cover Semi-private room Full cover Private room Covered at semi-private rate Private Hospitals Day case Covered with a €100 excess per claim Semi-private room Covered with a €150 excess per admission Private room Covered at semi-private rate with €150 excess per admission Specified orthopaedic procedures Co-payment of €2,000 (in addition to excess) Specified ophthalmic procedures Co-payment
================================================================================
```

Evaluation.py

In [4]:
```
import json
import numpy as np
import faiss
from sentence_transformers import SentenceTransformer

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

INDEX_PATH = "faiss_index.bin"
METADATA_PATH = "metadata.json"
TOP_K = 3

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

print("Loading embedding model...")
model = SentenceTransformer("all-MiniLM-L6-v2")

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
# ----------------------------

def search(query, k=TOP_K):
    query_embedding = model.encode([query])
    query_embedding = np.array(query_embedding).astype("float32")

    # Normalize query for cosine similarity
    faiss.normalize_L2(query_embedding)

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

    return [metadata[idx] for idx in indices[0]]


# ----------------------------
# 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]  # assume unique plan names

# ----------------------------
# 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"]

    results = search(query)
    returned_ids = [r["doc_id"] for r in results]

    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%|          | 0/103 [00:00<?, ?it/s]


 BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
 Key                     | Status     |  |
 ------------------------+------------+--+-
 embeddings.position_ids | UNEXPECTED |  |

 Notes:
 - UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.

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', 'irish_life_health_health_plan_26.1_table_of_cover_2026-01-01',
'level_health_plan_b_with_150_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_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: ['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', 'level_health_plan_a_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: ['irish_life_health_health_plan_26.1_table_of_cover_2026-01-01', 'irish_life_health_horizon_4_table_of_cover_2026-01-01',
'level_health_plan_b_with_150_excess_table_of_cover__table_of_cover_2025-06-27']
```

```
------------------------------------------------------------

===== FINAL METRICS =====
Top-1 Accuracy: 0.500
Top-3 Recall: 1.000
MRR: 0.708
```

Phase 3 Clean BM25 Baseline Code: evaluate_bm25.py

In [5]:
```
! 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 [6]:

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

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

METADATA_PATH = "metadata.json"
TOP_K = 3

# ----------------------------
# LOAD + FILTER METADATA
# ----------------------------

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

# Keep only Table of Cover documents
filtered_metadata = [
    m for m in metadata
    if m["doc_type"] == "table_of_cover"
]

print(f"Using {len(filtered_metadata)} table_of_cover documents")

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

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

corpus = [m["text"] for m in filtered_metadata]
tokenized_corpus = [tokenize(doc) for doc in corpus]

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

# ----------------------------
# SEARCH
# ----------------------------

def search_bm25(query, k=TOP_K):
    tokenized_query = tokenize(query)
    scores = bm25.get_scores(tokenized_query)
    top_indices = np.argsort(scores)[::-1][:k]
    return [filtered_metadata[i] for i in top_indices]

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

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

def get_doc_id(plan_keyword):
    matches = [
        m["doc_id"]
        for m in filtered_metadata
        if plan_keyword.lower() in m["plan_name"].lower()
    ]
    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 (clean) evaluation...\n")

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

    results = search_bm25(query)
    returned_ids = [r["doc_id"] for r in results]

    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 (FILTERED) =====")
print("Top-1:", correct_top1 / total)
print("Top-3:", correct_topk / total)
print("MRR:", sum(reciprocal_ranks) / total)
```

```
Loading metadata...
Using 16 table_of_cover documents
Building BM25 index...

Running BM25 (clean) evaluation...

Query: Does Horizon 4 cover inpatient consultant fees?
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_select_starter_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_of_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',
'irish_life_health_health_plan_26.1_table_of_cover_2026-01-01']
------------------------------------------------------------
Query: Are inpatient scans fully covered under Health Plan 26.1?
Returned: ['irish_life_health_health_plan_26.1_table_of_cover_2026-01-01', 'irish_life_health_first_cover_table_of_cover_2026-01-01',
'irish_life_health_select_starter_table_of_cover_2026-01-01']
------------------------------------------------------------

===== BM25 (FILTERED) =====
Top-1: 0.75
Top-3: 0.75
MRR: 0.75
```

Hybrid Strategy.py

In [7]:
```
import json
import numpy as np
import faiss
import re
from rank_bm25 import BM25Okapi
from sentence_transformers import SentenceTransformer

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

INDEX_PATH = "faiss_index.bin"
METADATA_PATH = "metadata.json"
TOP_K = 3
ALPHA = 0.6
BETA = 0.4

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

print("Loading model...")
model = SentenceTransformer("all-MiniLM-L6-v2")

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

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

# Filter to table_of_cover only
filtered_indices = [
    i for i, m in enumerate(metadata)
    if m["doc_type"] == "table_of_cover"
]

filtered_metadata = [metadata[i] for i in filtered_indices]

print(f"Using {len(filtered_metadata)} table_of_cover documents")

# ------------------------------
# BM25 SETUP
# ------------------------------

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

```
corpus = [m["text"] for m in filtered_metadata]
tokenized_corpus = [tokenize(doc) for doc in corpus]

bm25 = BM25Okapi(tokenized_corpus)

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

def normalize(scores):
    min_s = np.min(scores)
    max_s = np.max(scores)
    if max_s - min_s == 0:
        return np.zeros_like(scores)
    return (scores - min_s) / (max_s - min_s)

def hybrid_search(query, k=TOP_K):

    # Dense scores for all docs
    query_embedding = model.encode([query])
    query_embedding = np.array(query_embedding).astype("float32")

    dense_scores_all, _ = index.search(query_embedding, len(metadata))
    dense_scores_all = -dense_scores_all[0]

    # Filter dense scores
    dense_scores = np.array([dense_scores_all[i] for i in filtered_indices])

    # BM25 scores
    tokenized_query = tokenize(query)
    bm25_scores = bm25.get_scores(tokenized_query)

    # Normalize
    dense_norm = normalize(dense_scores)
    bm25_norm = normalize(bm25_scores)

    # Fusion
    final_scores = ALPHA * dense_norm + BETA * bm25_norm

    top_indices = np.argsort(final_scores)[::-1][:k]
    return [filtered_metadata[i] for i in top_indices]

# ----------------------------
# EVALUATION
# ----------------------------

def get_doc_id(plan_keyword):
    matches = [
        m["doc_id"]
        for m in filtered_metadata
        if plan_keyword.lower() in m["plan_name"].lower()
    ]
    return matches[0]

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")
    }
]

correct_top1 = 0
correct_topk = 0
reciprocal_ranks = []

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

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

    results = hybrid_search(query)
    returned_ids = [r["doc_id"] for r in results]

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

```
Loading model...

Loading weights:   0%|          | 0/103 [00:00<?, ?it/s]
```

```
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key                     | Status     |  |
------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
```

```
Loading FAISS index...
Loading metadata...
Using 16 table_of_cover documents

Running CLEAN Hybrid evaluation...

Query: Does Horizon 4 cover inpatient consultant fees?
Returned: ['level_health_plan_d_table_of_cover__table_of_cover_2025-06-27', 'irish_life_health_horizon_4_table_of_cover_2026-01-01',
'level_health_plan_c_table_of_cover__table_of_cover_2025-06-27']
------------------------------------------------------------
Query: Which plan has a €300 excess for semi-private room admission?
Returned: ['level_health_plan_b_with_300_excess_table_of_cover__table_of_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: ['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',
'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?
Returned: ['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',
'irish_life_health_health_plan_26.1_table_of_cover_2026-01-01']
------------------------------------------------------------

===== HYBRID (FILTERED) =====
Top-1: 0.25
Top-3: 0.75
MRR: 0.4583333333333333
```

phase5_risk_scoring.py

In [8]:
```
import json
import re
from collections import defaultdict

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

METADATA_PATH = "metadata.json"

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

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

# Keep only table_of_cover docs
metadata = [m for m in metadata if m["doc_type"] == "table_of_cover"]

print(f"Loaded {len(metadata)} table_of_cover documents")

# ----------------------------
# GROUP CHUNKS BY DOCUMENT
# ----------------------------

documents = defaultdict(list)

for m in metadata:
    documents[m["doc_id"]].append(m["text"])

# Merge chunks per document
merged_documents = {
    doc_id: " ".join(texts)
    for doc_id, texts in documents.items()
}

# ----------------------------
# PLAN FEATURE EXTRACTION
# ----------------------------

def extract_plan_features(doc_id, text):
    text_lower = text.lower()

    # Inpatient cover
    has_full_inpatient = (
        "inpatient consultant fees covered" in text_lower
        or "consultants fees covered" in text_lower
```

```
        or "fully covered" in text_lower
    )

    # Psychiatric days
    psych_match = re.search(r"(\d+)\s+days", text_lower)
    psychiatric_days = int(psych_match.group(1)) if psych_match else 0

    # Semi-private excess
    excess_match = re.search(r"€\s?(\d+)\s+excess", text_lower)
    semi_private_excess = int(excess_match.group(1)) if excess_match else 0

    return {
        "doc_id": doc_id,
        "has_full_inpatient_cover": has_full_inpatient,
        "psychiatric_days": psychiatric_days,
        "semi_private_excess": semi_private_excess
    }

# Extract features for all plans
plans = []

for doc_id, text in merged_documents.items():
    features = extract_plan_features(doc_id, text)
    plans.append(features)

print("Extracted features for all plans")

# ----------------------------
# PATIENT PROFILE
# ----------------------------

patient_profile = {
    "age": 52,
    "chronic_conditions": ["heart_disease"],
    "medication_frequency": "daily",
    "gp_visits_per_year": 8,
    "specialist_visits_per_year": 6,
    "hospital_admissions_last_2_years": 2,
    "budget_per_month": 100,
    "risk_tolerance": 4
}

# ----------------------------
# RISK SCORING
# ----------------------------

def compute_risk_score(profile):
    score = 0

    if profile["age"] > 60:
        score += 3
    elif profile["age"] > 45:
        score += 2

    score += len(profile["chronic_conditions"]) * 3
```

```
    if profile["medication_frequency"] == "daily":
        score += 3
    elif profile["medication_frequency"] == "weekly":
        score += 2

    score += profile["hospital_admissions_last_2_years"] * 2

    if profile["specialist_visits_per_year"] > 5:
        score += 2

    return score


def classify_risk(score):
    if score >= 12:
        return "high"
    elif score >= 6:
        return "medium"
    return "low"


risk_score = compute_risk_score(patient_profile)
risk_level = classify_risk(risk_score)

print(f"Risk score: {risk_score}")
print(f"Risk level: {risk_level}")

# ----------------------------
# PLAN SCORING
# ----------------------------

def score_plan(plan, profile, risk_level):
    score = 0

    if risk_level == "high":
        if plan["has_full_inpatient_cover"]:
            score += 5
        else:
            score -= 5

        if plan["semi_private_excess"] <= 150:
            score += 3

        if plan["psychiatric_days"] >= 100:
            score += 2

    elif risk_level == "medium":
        if plan["has_full_inpatient_cover"]:
            score += 3
        if plan["semi_private_excess"] <= 200:
            score += 2

    else:  # low risk
        score -= plan["semi_private_excess"] / 100

    return score
```

```
# Rank plans
ranked_plans = sorted(
    plans,
    key=lambda p: score_plan(p, patient_profile, risk_level),
    reverse=True
)

# ----------------------------
# OUTPUT TOP 3
# ----------------------------

print("\nTop Recommended Plans:\n")

for i, plan in enumerate(ranked_plans[:3], 1):
    print(f"{i}. {plan['doc_id']}")
    print(f"   Full inpatient: {plan['has_full_inpatient_cover']}")
    print(f"   Psychiatric days: {plan['psychiatric_days']}")
    print(f"   Semi-private excess: €{plan['semi_private_excess']}")
    print("-" * 60)
```

```
Loading metadata...
Loaded 16 table_of_cover documents
Extracted features for all plans
Risk score: 14
Risk level: high

Top Recommended Plans:

1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Full inpatient: True
   Psychiatric days: 100
   Semi-private excess: €0
------------------------------------------------------------
2. irish_life_health_horizon_4_table_of_cover_2026-01-01
   Full inpatient: True
   Psychiatric days: 100
   Semi-private excess: €50
------------------------------------------------------------
3. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Full inpatient: True
   Psychiatric days: 120
   Semi-private excess: €75
------------------------------------------------------------
```

phase5_risk_scoring_v2.py

Now we stop being generic and become condition-aware + clinically aware + financially aware.

Below is a complete upgraded Phase 5 engine.

This version:

Detects cardiac coverage

Detects high-tech hospital cover

Detects exclusions

Detects chronic illness references

Uses specialist visits

Uses admission history

Uses budget filtering

Scores intelligently

In [9]:

```
import json
import re
from collections import defaultdict

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

METADATA_PATH = "metadata.json"

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

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

metadata = [m for m in metadata if m["doc_type"] == "table_of_cover"]
print(f"Loaded {len(metadata)} table_of_cover documents")

# ----------------------------
# GROUP CHUNKS BY DOCUMENT
# ----------------------------

documents = defaultdict(list)

for m in metadata:
    documents[m["doc_id"]].append(m["text"])

merged_documents = {
    doc_id: " ".join(texts).lower()
    for doc_id, texts in documents.items()
}

# ----------------------------
# PRICE EXTRACTION
# ----------------------------

def extract_price(text):
    # Match €39.02 per month / €39 per month / €39 monthly
    match = re.search(r"€\s?([\d,.]+)\s*(per month|monthly)", text)
    if match:
        return float(match.group(1).replace(",", ""))
    return None

# ----------------------------
# FEATURE EXTRACTION
# ----------------------------

def extract_plan_features(doc_id, text):
    features = {}

    features["doc_id"] = doc_id
```

```
    # Inpatient cover
    features["full_inpatient_cover"] = (
        "consultant fees covered" in text
        or "inpatient consultant fees covered" in text
        or "fully covered" in text
    )

    # Cardiac coverage
    features["cardiac_covered"] = (
        "cardiac" in text and "not covered" not in text
    )

    # High-tech hospital
    features["high_tech_access"] = (
        "high-tech hospital" in text and "not covered" not in text
    )

    # Psychiatric days
    psych_match = re.search(r"(\d+)\s+days", text)
    features["psychiatric_days"] = int(psych_match.group(1)) if psych_match else 0

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

    # Exclusions
    features["has_exclusions"] = "not covered" in text

    # Chronic mention
    features["mentions_chronic"] = "chronic" in text

    # Auto price extraction
    features["monthly_price"] = extract_price(text)

    return features

# Extract all plans
plans = []

for doc_id, text in merged_documents.items():
    features = extract_plan_features(doc_id, text)
    plans.append(features)

print("Extracted enhanced plan features.")

# ----------------------------
# PATIENT PROFILE
# ----------------------------

patient_profile = {
    "age": 52,
    "chronic_conditions": ["heart_disease"],
    "medication_frequency": "daily",
    "gp_visits_per_year": 8,
    "specialist_visits_per_year": 6,
    "hospital_admissions_last_2_years": 2,
```

```
    "budget_per_month": 100,
    "risk_tolerance": 4
}

# ----------------------------
# RISK COMPUTATION
# ----------------------------

def compute_risk_score(profile):
    score = 0

    if profile["age"] > 60:
        score += 3
    elif profile["age"] > 45:
        score += 2

    score += len(profile["chronic_conditions"]) * 3

    if profile["medication_frequency"] == "daily":
        score += 3

    score += profile["hospital_admissions_last_2_years"] * 2

    if profile["specialist_visits_per_year"] > 5:
        score += 3

    return score


def classify_risk(score):
    if score >= 12:
        return "high"
    elif score >= 6:
        return "medium"
    return "low"


risk_score = compute_risk_score(patient_profile)
risk_level = classify_risk(risk_score)

print(f"Risk score: {risk_score}")
print(f"Risk level: {risk_level}")

# ----------------------------
# PLAN SCORING
# ----------------------------

def score_plan(plan, profile, risk_level):
    score = 0

    # Budget filtering (only if price available)
    if plan["monthly_price"] is not None:
        if plan["monthly_price"] > profile["budget_per_month"]:
            score -= 10
        else:
            score += 2  # reward affordable plan
```

```
    # High risk logic
    if risk_level == "high":
        if plan["full_inpatient_cover"]:
            score += 6
        else:
            score -= 6

        if plan["semi_private_excess"] <= 150:
            score += 3

        if plan["psychiatric_days"] >= 100:
            score += 2

    # Condition-aware logic
    if "heart_disease" in profile["chronic_conditions"]:
        if plan["cardiac_covered"]:
            score += 6
        else:
            score -= 5

        if plan["high_tech_access"]:
            score += 4

    # Specialist-heavy
    if profile["specialist_visits_per_year"] > 5:
        if plan["full_inpatient_cover"]:
            score += 2

    # Admission history
    if profile["hospital_admissions_last_2_years"] > 1:
        if plan["semi_private_excess"] <= 100:
            score += 3

    # Penalize exclusions
    if plan["has_exclusions"]:
        score -= 2

    return score

# Rank plans
ranked_plans = sorted(
    plans,
    key=lambda p: score_plan(p, patient_profile, risk_level),
    reverse=True
)

# ----------------------------
# OUTPUT
# ----------------------------

print("\nTop Recommended Plans:\n")

for i, plan in enumerate(ranked_plans[:3], 1):
    print(f"{i}. {plan['doc_id']}")
    print(f"   Price: {plan['monthly_price']}")
```

```
print(f"   Cardiac Covered: {plan['cardiac_covered']}")
print(f"   High-Tech Access: {plan['high_tech_access']}")
print(f"   Inpatient Cover: {plan['full_inpatient_cover']}")
print(f"   Psychiatric Days: {plan['psychiatric_days']}")
print(f"   Excess: €{plan['semi_private_excess']}")
print("-" * 60)
```

```
Loading metadata...
Loaded 16 table_of_cover documents
Extracted enhanced plan features.
Risk score: 15
Risk level: high

Top Recommended Plans:

1. irish_life_health_horizon_4_table_of_cover_2026-01-01
   Price: None
   Cardiac Covered: True
   High-Tech Access: True
   Inpatient Cover: True
   Psychiatric Days: 100
   Excess: €50
------------------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Price: None
   Cardiac Covered: True
   High-Tech Access: True
   Inpatient Cover: True
   Psychiatric Days: 120
   Excess: €75
------------------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Price: None
   Cardiac Covered: True
   High-Tech Access: True
   Inpatient Cover: True
   Psychiatric Days: 100
   Excess: €200
------------------------------------------------------------
```

In [10]:
```
import json
import re
from collections import defaultdict

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

METADATA_PATH = "metadata.json"

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

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

metadata = [m for m in metadata if m["doc_type"] == "table_of_cover"]
print(f"Loaded {len(metadata)} table_of_cover documents")

# ----------------------------
# GROUP DOCUMENT TEXT
# ----------------------------

documents = defaultdict(list)

for m in metadata:
    documents[m["doc_id"]].append(m["text"])

merged_documents = {
    doc_id: " ".join(texts).lower()
    for doc_id, texts in documents.items()
}

# ----------------------------
# DISEASE RULES ENGINE
# ----------------------------

DISEASE_RULES = {

    "heart_disease": {
        "keywords": ["cardiac", "cardiology", "angioplasty", "stent"],
        "requires_high_tech": True,
        "weight": 7
    },

    "diabetes": {
        "keywords": ["diabetes", "insulin", "endocrinology"],
        "requires_outpatient": True,
        "weight": 6
    },

    "cancer": {
        "keywords": ["oncology", "chemotherapy", "radiotherapy", "oncotype"],
```

```
        "requires_high_tech": True,
        "weight": 9
    },

    "psychiatric_disorder": {
        "keywords": ["psychiatric", "mental health", "substance abuse"],
        "requires_psych_cover": True,
        "weight": 6
    },

    "chronic_kidney_disease": {
        "keywords": ["dialysis", "renal"],
        "requires_high_tech": True,
        "weight": 8
    },

    "copd": {
        "keywords": ["respiratory", "pulmonary"],
        "requires_hospital": True,
        "weight": 6
    },

    "autoimmune_disorder": {
        "keywords": ["immunology", "biologic therapy"],
        "requires_specialist": True,
        "weight": 7
    },

    "pregnancy": {
        "keywords": ["maternity", "obstetric"],
        "requires_maternity": True,
        "weight": 8
    },

    "neurological_disorder": {
        "keywords": ["neurology", "mri", "ct scan"],
        "requires_high_tech": True,
        "weight": 7
    },

    "orthopaedic_condition": {
        "keywords": ["orthopaedic", "joint replacement"],
        "requires_hospital": True,
        "weight": 6
    }
}

# ----------------------------
# FEATURE EXTRACTION
# ----------------------------

def extract_plan_features(doc_id, text):

    features = {}
    features["doc_id"] = doc_id
```

```
    features["full_inpatient_cover"] = "consultant fees covered" in text or "fully covered" in text
    features["high_tech_access"] = "high-tech hospital" in text and "not covered" not in text
    features["has_psych_cover"] = "psychiatric" in text
    features["has_maternity"] = "maternity" in text
    features["has_exclusions"] = "not covered" in text

    psych_match = re.search(r"(\d+)\s+days", text)
    features["psychiatric_days"] = int(psych_match.group(1)) if psych_match else 0

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

    features["text"] = text  # store for keyword matching

    return features


plans = []

for doc_id, text in merged_documents.items():
    plans.append(extract_plan_features(doc_id, text))

print("Extracted plan features.")

# ----------------------------
# PATIENT PROFILE
# ----------------------------

patient_profile = {
    "age": 52,
    "chronic_conditions": ["heart_disease", "diabetes"],
    "medication_frequency": "daily",
    "gp_visits_per_year": 8,
    "specialist_visits_per_year": 6,
    "hospital_admissions_last_2_years": 2,
    "budget_per_month": 100
}

# ----------------------------
# RISK SCORE
# ----------------------------

def compute_risk_score(profile):

    score = 0

    if profile["age"] > 60:
        score += 3
    elif profile["age"] > 45:
        score += 2

    score += len(profile["chronic_conditions"]) * 3

    if profile["medication_frequency"] == "daily":
        score += 3
```

```
    score += profile["hospital_admissions_last_2_years"] * 2

    if profile["specialist_visits_per_year"] > 5:
        score += 3

    return score


risk_score = compute_risk_score(patient_profile)

print("Risk Score:", risk_score)

# ----------------------------
# DISEASE-AWARE PLAN SCORING
# ----------------------------

def score_plan(plan, profile):

    score = 0

    # Generic inpatient logic
    if plan["full_inpatient_cover"]:
        score += 4
    else:
        score -= 4

    if plan["semi_private_excess"] <= 150:
        score += 2

    if plan["has_exclusions"]:
        score -= 2

    # Disease-specific logic
    for condition in profile["chronic_conditions"]:

        rules = DISEASE_RULES.get(condition)

        if not rules:
            continue

        # Keyword coverage
        for keyword in rules["keywords"]:
            if keyword in plan["text"]:
                score += rules["weight"]

        # High-tech requirement
        if rules.get("requires_high_tech") and plan["high_tech_access"]:
            score += 3

        # Psychiatric requirement
        if rules.get("requires_psych_cover") and plan["has_psych_cover"]:
            score += 3

        # Maternity requirement
        if rules.get("requires_maternity") and plan["has_maternity"]:
            score += 3
```

```
    return score


# Rank plans
ranked_plans = sorted(
    plans,
    key=lambda p: score_plan(p, patient_profile),
    reverse=True
)

# ----------------------------
# OUTPUT
# ----------------------------

print("\nTop Recommended Plans:\n")

for i, plan in enumerate(ranked_plans[:3], 1):
    print(f"{i}. {plan['doc_id']}")
    print(f"   Excess: €{plan['semi_private_excess']}")
    print(f"   High-Tech: {plan['high_tech_access']}")
    print(f"   Psychiatric Cover: {plan['has_psych_cover']}")
    print("-" * 60)
```

```
Loading metadata...
Loaded 16 table_of_cover documents
Extracted plan features.
Risk Score: 18

Top Recommended Plans:

1. irish_life_health_horizon_4_table_of_cover_2026-01-01
   Excess: €50
   High-Tech: True
   Psychiatric Cover: True
------------------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Excess: €75
   High-Tech: True
   Psychiatric Cover: True
------------------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Excess: €200
   High-Tech: True
   Psychiatric Cover: True
------------------------------------------------------------
```

Phase 5 Pro

In [31]:
```
import json
import re
from collections import defaultdict

# ============================
# LOAD METADATA
# ============================

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

metadata = [m for m in metadata if m["doc_type"] == "table_of_cover"]
print(f"Loaded {len(metadata)} table_of_cover documents")

# ============================
# MERGE TEXT PER PLAN
# ============================

documents = defaultdict(list)

for m in metadata:
    documents[m["doc_id"]].append(m["text"].lower())

merged_documents = {
    doc_id: " ".join(texts)
    for doc_id, texts in documents.items()
}

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

DISEASE_RULES = {

    "heart_disease": {
        "keywords": ["cardiac", "angioplasty", "stent"],
        "weight": 7,
        "requires_high_tech": True
    },

    "diabetes": {
        "keywords": ["diabetes", "insulin", "endocrinology"],
        "weight": 5,
        "requires_outpatient": True
    },

    "cancer": {
        "keywords": ["oncology", "chemotherapy", "radiotherapy", "oncotype"],
```

```
        "weight": 9,
        "requires_high_tech": True
    },

    "psychiatric_disorder": {
        "keywords": ["psychiatric", "mental health"],
        "weight": 6,
        "requires_psych": True
    },

    "orthopaedic_condition": {
        "keywords": ["orthopaedic", "joint replacement"],
        "weight": 6,
        "requires_hospital": True
    },

    "neurological_disorder": {
        "keywords": ["neurology", "mri", "ct scan"],
        "weight": 7,
        "requires_high_tech": True
    },

    "respiratory_disease": {
        "keywords": ["respiratory", "pulmonary"],
        "weight": 5
    },

    "chronic_kidney_disease": {
        "keywords": ["dialysis", "renal"],
        "weight": 8,
        "requires_high_tech": True
    },

    "autoimmune_disorder": {
        "keywords": ["immunology", "biologic"],
        "weight": 7
    },

    "pregnancy": {
        "keywords": ["maternity", "obstetric"],
        "weight": 8,
        "requires_maternity": True
    }
}


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

def extract_plan_features(doc_id, text):

    features = {}
    features["doc_id"] = doc_id
    features["text"] = text
```

```
# Inpatient
features["full_inpatient_cover"] = (
    "inpatient consultant fees" in text and "covered" in text
)

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

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

# Cardiac copay
cardiac_match = re.search(r"€\s?(\d{2,5})\s*(?:co-?payment|copayment)[^\n]{0,40}cardiac", text)
features["cardiac_copay"] = int(cardiac_match.group(1)) if cardiac_match else 0

# ========================
# HIGH TECH PARSING
# ========================

hightech_match = re.search(
    r"high-tech hospital(.*?)(outpatient|day to day|members benefits|$)",
    text,
    re.DOTALL
)

hightech_text = hightech_match.group(1) if hightech_match else ""

features["high_tech_percent"] = 0
features["high_tech_restriction"] = 1.0
features["high_tech_excess"] = 0
features["high_tech_cardiac_copay"] = 0

if hightech_text:

    # Extract % cover
    percents = re.findall(r"(\d+)%\s*cover", hightech_text)
    if percents:
        features["high_tech_percent"] = sum(int(p) for p in percents) / len(percents)

    elif "covered" in hightech_text:
        features["high_tech_percent"] = 100

    # Beacon only = restricted network
    if "beacon only" in hightech_text:
        features["high_tech_restriction"] = 0.4

    if "50% cover in mater private" in hightech_text:
        features["high_tech_restriction"] = 0.6

    # High-tech excess
    ht_excess = re.search(r"€\s?(\d+)\s+excess", hightech_text)
    if ht_excess:
        features["high_tech_excess"] = int(ht_excess.group(1))
```

```
        # High-tech cardiac copay
        ht_cardiac = re.search(r"€\s?(\d+)[^\n]{0,40}cardiac", hightech_text)
        if ht_cardiac:
            features["high_tech_cardiac_copay"] = int(ht_cardiac.group(1))

    return features


plans = [
    extract_plan_features(doc_id, text)
    for doc_id, text in merged_documents.items()
]

print("Extracted calibrated plan features.")

# ============================
# PATIENT PROFILE
# ============================

patient_profile = {
    "age": 52,
    "chronic_conditions": ["heart_disease", "diabetes"],
    "hospital_admissions_last_2_years": 2,
    "specialist_visits_per_year": 6
}

# ============================
# PLAN SCORING
# ============================

def score_plan(plan, profile):

    score = 0

    # Inpatient
    score += 5 if plan["full_inpatient_cover"] else -5

    # Psychiatric strength
    score += plan["psychiatric_days"] / 40

    # Excess penalty
    score -= plan["semi_private_excess"] / 80

    # Cardiac copay penalty (stronger)
    score -= plan["cardiac_copay"] / 100

    # Disease logic
    for condition in profile["chronic_conditions"]:

        rules = DISEASE_RULES.get(condition)
        if not rules:
            continue

        for keyword in rules["keywords"]:
            if keyword in plan["text"]:
                score += rules["weight"]
```

```
        if rules.get("requires_high_tech"):

            # Nonlinear high-tech boost
            ht_score = (
    (plan["high_tech_percent"] / 100) ** 1.5
) * 6



            ht_score *= plan["high_tech_restriction"]

            ht_score -= plan["high_tech_excess"] / 150
            ht_score -= plan["high_tech_cardiac_copay"] / 200

            score += ht_score

    return round(score, 2)


ranked_plans = sorted(
    plans,
    key=lambda p: score_plan(p, patient_profile),
    reverse=True
)

# ============================
# OUTPUT
# ============================

print("\nTop Recommended Plans:\n")

for i, plan in enumerate(ranked_plans[:3], 1):
    print(f"{i}. {plan['doc_id']}")
    print(f"   Score: {score_plan(plan, patient_profile)}")
    print(f"   Excess: €{plan['semi_private_excess']}")
    print(f"   Cardiac Co-pay: €{plan['cardiac_copay']}")
    print(f"   High-Tech %: {plan['high_tech_percent']}")
    print(f"   High-Tech Restriction: {plan['high_tech_restriction']}")
    print(f"   Psychiatric Days: {plan['psychiatric_days']}")
    print("-" * 60)
```

```
Loading metadata...
Loaded 16 table_of_cover documents
Extracted calibrated plan features.

Top Recommended Plans:

1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 16.62
   Excess: €0
   Cardiac Co-pay: €0
   High-Tech %: 50.0
   High-Tech Restriction: 1.0
   Psychiatric Days: 100
------------------------------------------------------------
```

```
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 14.33
   Excess: €75
   Cardiac Co-pay: €0
   High-Tech %: 47.142857142857146
   High-Tech Restriction: 0.4
   Psychiatric Days: 120
------------------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 10.66
   Excess: €200
   Cardiac Co-pay: €0
   High-Tech %: 35.0
   High-Tech Restriction: 0.4
   Psychiatric Days: 100
------------------------------------------------------------
```

In [38]:
```
import json
import re
from collections import defaultdict

# ============================
# LOAD METADATA
# ============================

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

metadata = [m for m in metadata if m["doc_type"] == "table_of_cover"]
print(f"Loaded {len(metadata)} table_of_cover documents")

# ============================
# MERGE TEXT PER PLAN
# ============================

documents = defaultdict(list)

for m in metadata:
    documents[m["doc_id"]].append(m["text"].lower())

merged_documents = {
    doc_id: " ".join(texts)
    for doc_id, texts in documents.items()
}

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

DISEASE_RULES = {
    "heart_disease": {
        "keywords": ["cardiac", "angioplasty", "stent"],
        "weight": 7,
        "requires_high_tech": True
    },
    "diabetes": {
        "keywords": ["diabetes", "insulin", "endocrinology"],
        "weight": 5
    },
    "cancer": {
        "keywords": ["oncology", "chemotherapy", "radiotherapy", "oncotype"],
        "weight": 9,
        "requires_high_tech": True
    },
    "psychiatric_disorder": {
        "keywords": ["psychiatric", "mental health"],
        "weight": 6
    },
    "neurological_disorder": {
        "keywords": ["neurology", "mri", "ct scan"],
        "weight": 7,
```

```
        "requires_high_tech": True
    },
    "orthopaedic_condition": {
        "keywords": ["orthopaedic", "joint replacement"],
        "weight": 6
    },
    "pregnancy": {
        "keywords": ["maternity", "obstetric"],
        "weight": 8
    }
}

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

def extract_plan_features(doc_id, text):

    features = {}
    features["doc_id"] = doc_id
    features["text"] = text

    # Inpatient coverage
    features["full_inpatient_cover"] = (
        "inpatient consultant fees" in text and "covered" in text
    )

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

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

    # Cardiac copay
    cardiac_match = re.search(
        r"€\s?(\d{2,5})\s*(?:co-?payment|copayment)[^\n]{0,40}cardiac",
        text
    )
    features["cardiac_copay"] = int(cardiac_match.group(1)) if cardiac_match else 0

    # ========================
    # HIGH-TECH PARSING
    # ========================

    hightech_match = re.search(
        r"high-tech hospital(.*?)(outpatient|day to day|members benefits|$)",
        text,
        re.DOTALL
    )

    hightech_text = hightech_match.group(1) if hightech_match else ""

    features["high_tech_percent"] = 0
    features["high_tech_restriction"] = 1.0
```

```
    features["high_tech_excess"] = 0
    features["high_tech_cardiac_copay"] = 0

    if hightech_text:

        percents = re.findall(r"(\d+)%\s*cover", hightech_text)

        if percents:
            features["high_tech_percent"] = sum(int(p) for p in percents) / len(percents)
        elif "covered" in hightech_text:
            features["high_tech_percent"] = 100

        if "beacon only" in hightech_text:
            features["high_tech_restriction"] = 0.4

        if "mater private" in hightech_text:
            features["high_tech_restriction"] = min(features["high_tech_restriction"], 0.6)

        ht_excess = re.search(r"€\s?(\d+)\s+excess", hightech_text)
        if ht_excess:
            features["high_tech_excess"] = int(ht_excess.group(1))

        ht_cardiac = re.search(r"€\s?(\d+)[^\n]{0,40}cardiac", hightech_text)
        if ht_cardiac:
            features["high_tech_cardiac_copay"] = int(ht_cardiac.group(1))

    return features


plans = [
    extract_plan_features(doc_id, text)
    for doc_id, text in merged_documents.items()
]

print("Extracted calibrated plan features.")

# ============================
# PATIENT PROFILE
# ============================

patient_profile = {
    "age": 52,
    "chronic_conditions": ["heart_disease", "diabetes"],
    "hospital_admissions_last_2_years": 2,
    "specialist_visits_per_year": 6
}

# ============================
# RISK CLASSIFICATION
# ============================

def compute_risk_score(profile):

    score = 0

    if profile["age"] > 60:
```

```
        score += 4
    elif profile["age"] > 45:
        score += 2

    score += len(profile["chronic_conditions"]) * 3
    score += profile["hospital_admissions_last_2_years"] * 2

    if profile["specialist_visits_per_year"] > 5:
        score += 2

    return score


def classify_risk(score):

    if score >= 12:
        return "high"
    elif score >= 6:
        return "medium"
    else:
        return "low"


risk_score = compute_risk_score(patient_profile)
risk_level = classify_risk(risk_score)

print("Risk Score:", risk_score)
print("Risk Level:", risk_level)

# ============================
# SCORING ENGINE
# ============================

def score_plan(plan, profile):

    score = 0

    # Risk scaling
    if risk_level == "high":
        hospital_weight = 1.5
        excess_weight = 1.0
        hightech_weight = 1.5
    elif risk_level == "medium":
        hospital_weight = 1.0
        excess_weight = 1.0
        hightech_weight = 1.0
    else:
        hospital_weight = 0.6
        excess_weight = 1.5
        hightech_weight = 0.6

    # Inpatient
    score += (5 if plan["full_inpatient_cover"] else -5) * hospital_weight

    # Psychiatric differentiation
    psych_boost = min(plan["psychiatric_days"], 150) / 25
```

```
    score += psych_boost

    # Excess penalty
    score -= (plan["semi_private_excess"] / 80) * excess_weight

    # Cardiac copay penalty
    score -= plan["cardiac_copay"] / 100

    # Disease logic
    for condition in profile["chronic_conditions"]:

        rules = DISEASE_RULES.get(condition)
        if not rules:
            continue

        for keyword in rules["keywords"]:
            if keyword in plan["text"]:
                score += rules["weight"]

        if rules.get("requires_high_tech"):

            ht_score = (
                (plan["high_tech_percent"] / 100) ** 1.5
            ) * 6

            ht_score *= plan["high_tech_restriction"]
            ht_score *= hightech_weight

            ht_score -= plan["high_tech_excess"] / 150
            ht_score -= plan["high_tech_cardiac_copay"] / 200

            score += ht_score

    # Over-insurance penalty for low risk
    if risk_level == "low":

        if plan["high_tech_percent"] > 80:
            score -= 2

        if plan["psychiatric_days"] > 120:
            score -= 2

    return round(score, 2)


ranked_plans = sorted(
    plans,
    key=lambda p: score_plan(p, patient_profile),
    reverse=True
)

# ============================
# OUTPUT
# ============================

print("\nTop Recommended Plans:\n")
```

```
for i, plan in enumerate(ranked_plans[:3], 1):
    print(f"{i}. {plan['doc_id']}")
    print(f"   Score: {score_plan(plan, patient_profile)}")
    print(f"   Excess: €{plan['semi_private_excess']}")
    print(f"   Cardiac Co-pay: €{plan['cardiac_copay']}")
    print(f"   High-Tech %: {plan['high_tech_percent']}")
    print(f"   High-Tech Restriction: {plan['high_tech_restriction']}")
    print(f"   Psychiatric Days: {plan['psychiatric_days']}")
    print("-" * 60)
```

```
Loading metadata...
Loaded 16 table_of_cover documents
Extracted calibrated plan features.
Risk Score: 14
Risk Level: high

Top Recommended Plans:

1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 21.68
   Excess: €0
   Cardiac Co-pay: €0
   High-Tech %: 50.0
   High-Tech Restriction: 1.0
   Psychiatric Days: 100
------------------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 19.02
   Excess: €75
   Cardiac Co-pay: €0
   High-Tech %: 47.142857142857146
   High-Tech Restriction: 0.4
   Psychiatric Days: 120
------------------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 14.91
   Excess: €200
   Cardiac Co-pay: €0
   High-Tech %: 35.0
   High-Tech Restriction: 0.4
   Psychiatric Days: 100
------------------------------------------------------------
```

In [39]:

```
# ============================
# SCENARIO EVALUATION
# ============================

SCENARIOS = [

    {
        "name": "High-Risk Cardiac + Diabetes",
        "profile": {
            "age": 55,
            "chronic_conditions": ["heart_disease", "diabetes"],
            "medication_frequency": "daily",
            "specialist_visits_per_year": 6,
            "hospital_admissions_last_2_years": 2
        }
    },

    {
        "name": "Cancer Patient",
        "profile": {
            "age": 48,
            "chronic_conditions": ["cancer"],
            "medication_frequency": "daily",
            "specialist_visits_per_year": 8,
            "hospital_admissions_last_2_years": 3
        }
    },

    {
        "name": "Maternity Case",
        "profile": {
            "age": 32,
            "chronic_conditions": ["pregnancy"],
            "medication_frequency": "monthly",
            "specialist_visits_per_year": 4,
            "hospital_admissions_last_2_years": 0
        }
    },

    {
        "name": "Psychiatric Condition",
        "profile": {
            "age": 40,
            "chronic_conditions": ["psychiatric_disorder"],
            "medication_frequency": "daily",
            "specialist_visits_per_year": 5,
            "hospital_admissions_last_2_years": 1
        }
    },

    {
        "name": "Young Low-Risk Adult",
        "profile": {
            "age": 25,
            "chronic_conditions": [],
```

```
            "medication_frequency": "none",
            "specialist_visits_per_year": 1,
            "hospital_admissions_last_2_years": 0
        }
    },

    {
        "name": "Neurological Disorder",
        "profile": {
            "age": 60,
            "chronic_conditions": ["neurological_disorder"],
            "medication_frequency": "daily",
            "specialist_visits_per_year": 7,
            "hospital_admissions_last_2_years": 2
        }
    }
]

print("\n==============================")
print("SCENARIO EVALUATION")
print("==============================")

for scenario in SCENARIOS:

    name = scenario["name"]
    profile = scenario["profile"]

    ranked = sorted(
        plans,
        key=lambda p: score_plan(p, profile),
        reverse=True
    )

    print(f"\n--- Scenario: {name} ---")

    for i, plan in enumerate(ranked[:3], 1):
        print(f"{i}. {plan['doc_id']}")
        print(f"   Score: {score_plan(plan, profile)}")
        print(f"   Excess: €{plan['semi_private_excess']}")
        print(f"   High-Tech %: {plan['high_tech_percent']}")
        print(f"   Psychiatric Days: {plan['psychiatric_days']}")
        print("-" * 50)
```

```
==============================
SCENARIO EVALUATION
==============================

--- Scenario: High-Risk Cardiac + Diabetes ---
1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 21.68
   Excess: €0
   High-Tech %: 50.0
   Psychiatric Days: 100
--------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
```

```
   Score: 19.02
   Excess: €75
   High-Tech %: 47.142857142857146
   Psychiatric Days: 120
--------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 14.91
   Excess: €200
   High-Tech %: 35.0
   Psychiatric Days: 100
--------------------------------------------------

--- Scenario: Cancer Patient ---
1. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 30.02
   Excess: €75
   High-Tech %: 47.142857142857146
   Psychiatric Days: 120
--------------------------------------------------
2. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 25.91
   Excess: €200
   High-Tech %: 35.0
   Psychiatric Days: 100
--------------------------------------------------
3. laya_healthcare_pmi_plan_b_table_of_benefits_table_of_cover_2021-10-01
   Score: 25.5
   Excess: €0
   High-Tech %: 0
   Psychiatric Days: 180
--------------------------------------------------

--- Scenario: Maternity Case ---
1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 19.5
   Excess: €0
   High-Tech %: 50.0
   Psychiatric Days: 100
--------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 19.36
   Excess: €75
   High-Tech %: 47.142857142857146
   Psychiatric Days: 120
--------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 17.0
   Excess: €200
   High-Tech %: 35.0
   Psychiatric Days: 100
--------------------------------------------------

--- Scenario: Psychiatric Condition ---
1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 23.5
   Excess: €0
```

```
   High-Tech %: 50.0
   Psychiatric Days: 100
--------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 23.36
   Excess: €75
   High-Tech %: 47.142857142857146
   Psychiatric Days: 120
--------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 21.0
   Excess: €200
   High-Tech %: 35.0
   Psychiatric Days: 100
--------------------------------------------------

--- Scenario: Young Low-Risk Adult ---
1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 11.5
   Excess: €0
   High-Tech %: 50.0
   Psychiatric Days: 100
--------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 11.36
   Excess: €75
   High-Tech %: 47.142857142857146
   Psychiatric Days: 120
--------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 9.0
   Excess: €200
   High-Tech %: 35.0
   Psychiatric Days: 100
--------------------------------------------------

--- Scenario: Neurological Disorder ---
1. irish_life_health_first_cover_table_of_cover_2026-01-01
   Score: 28.68
   Excess: €0
   High-Tech %: 50.0
   Psychiatric Days: 100
--------------------------------------------------
2. irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
   Score: 26.02
   Excess: €75
   High-Tech %: 47.142857142857146
   Psychiatric Days: 120
--------------------------------------------------
3. irish_life_health_benefit_table_of_cover_2026-01-01
   Score: 21.91
   Excess: €200
   High-Tech %: 35.0
   Psychiatric Days: 100
--------------------------------------------------
```

Phase 6

In [12]:
```
! pip install transformers accelerate bitsandbytes rank_bm25
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (5.0.0)
Requirement already satisfied: accelerate in /usr/local/lib/python3.12/dist-packages (1.12.0)
Collecting bitsandbytes
  Downloading bitsandbytes-0.49.1-py3-none-manylinux_2_24_x86_64.whl.metadata (10 kB)
Requirement already satisfied: rank_bm25 in /usr/local/lib/python3.12/dist-packages (0.2.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers) (3.20.3)
Requirement already satisfied: huggingface-hub<2.0,>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (1.3.7)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (26.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2025.11.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.2)
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from accelerate) (2.9.0+cu126)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<2.0,>=1.3.0->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<2.0,>=1.3.0->transformers) (1.2.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<2.0,>=1.3.0->transformers) (0.28.1)
Requirement already satisfied: shellingham in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<2.0,>=1.3.0->transformers) (1.5.4)
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<2.0,>=1.3.0->transformers) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.6.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (1.11.1.6)
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.5.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from typer-slim->transformers) (8.3.1)
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub<2.0,>=1.3.0->transformers) (4.12.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub<2.0,>=1.3.0->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<2.0,>=1.3.0->transformers)
(1.0.9)
```

```
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub<2.0,>=1.3.0->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<2.0,>=1.3.0-
>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>=2.0.0->accelerate) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=2.0.0->accelerate) (3.0.3)
Downloading bitsandbytes-0.49.1-py3-none-manylinux_2_24_x86_64.whl (59.1 MB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m  [32m59.1/59.1 MB [0m  [31m14.8 MB/s [0m eta  [36m0:00:00 [0m
 [?25hInstalling collected packages: bitsandbytes
Successfully installed bitsandbytes-0.49.1
```

In [50]:

```
import torch
import gc
import os

# Delete variables if they exist
for var in ["model", "tokenizer"]:
    if var in globals():
        del globals()[var]

gc.collect()
torch.cuda.empty_cache()

# Force defragment
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "expandable_segments:True"

print("GPU cleaned.")
print("Allocated:", torch.cuda.memory_allocated() / 1e9, "GB")
print("Reserved :", torch.cuda.memory_reserved() / 1e9, "GB")
```

```
GPU cleaned.
Allocated: 5.006795264 GB
Reserved : 8.510242816 GB
```

In [51]:
```
import json
import numpy as np
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

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

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

# ====================================================
# RISK LEVEL LABEL
# ====================================================

def get_risk_label(risk_score):
    if risk_score >= 14:
        return "HIGH"
    elif risk_score >= 8:
        return "MEDIUM"
    else:
        return "LOW"

risk_level = get_risk_label(risk_score)

# ====================================================
# RANK PLANS
# ====================================================

ranked_plans = sorted(
    plans,
    key=lambda p: score_plan(p, patient_profile),
    reverse=True
)

top_plans = ranked_plans[:TOP_K_PLANS]

# ====================================================
# STRUCTURED EVIDENCE BUILDER
# ====================================================

def build_structured_evidence(plan, profile):

    evidence = {}

    text = plan.get("text", "")

    # Inpatient
    if plan.get("full_inpatient", False):
        evidence["inpatient"] = "Inpatient consultant fees and scans are covered."
    else:
        evidence["inpatient"] = "Inpatient consultant fees are not fully covered."

    # Excess
```

```
    excess = plan.get("excess", 0)
    evidence["excess"] = f"Plan excess is €{excess}."

    # Disease relevance
    disease_evidence = []

    for condition in profile["chronic_conditions"]:
        rules = DISEASE_RULES.get(condition)
        if not rules:
            continue

        matched = False
        for kw in rules["keywords"]:
            if kw in text:
                disease_evidence.append(
                    f"Mentions '{kw}' related to {condition}."
                )
                matched = True

        if not matched:
            disease_evidence.append(
                f"No explicit mention of {condition} coverage."
            )

    evidence["disease"] = disease_evidence

    # High-tech
    if plan.get("high_tech", False):
        evidence["hightech"] = "High-tech hospital coverage available."
    else:
        evidence["hightech"] = "High-tech hospital coverage not available or restricted."

    return evidence

# ====================================================
# LOAD MODEL (4-bit)
# ====================================================

print("Loading LLM...")

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16
)

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_config,
    device_map="auto"
)

print("LLM loaded.")

# ====================================================
```

```
# EXPLANATION GENERATOR
# ==================================================

def generate_explanation(rank, plan, next_plan=None):

    breakdown, total_score = compute_score_breakdown(plan, patient_profile)
    evidence = build_structured_evidence(plan, patient_profile)

    comparison_line = ""
    if next_plan:
        comparison_line = (
            f"This plan ranked above {next_plan['doc_id']} "
            f"because it achieved a higher total score."
        )

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

RULES:
- Use ONLY the structured evidence provided.
- Do NOT invent benefits.
- Connect patient conditions to coverage.
- Explain ranking logically.
- Be concise and factual.
- Max 180 words.

PATIENT PROFILE:
{json.dumps(patient_profile)}

RISK LEVEL: {risk_level}

PLAN RANK: #{rank}
PLAN ID: {plan["doc_id"]}
TOTAL SCORE: {total_score}

SCORE BREAKDOWN:
{json.dumps(breakdown)}

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

TASK:
Explain why this plan ranked #{rank}.
Link:
- inpatient component
- excess penalty
- disease match
- high-tech impact
Mention limitations clearly.
{comparison_line}
"""

    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    outputs = model.generate(
        **inputs,
```

```
        max_new_tokens=220,
        temperature=0.0,
        do_sample=False
    )

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

# ================================================
# RUN EXPLANATIONS
# ================================================

print("\n==============================")
print("DECISION-AWARE EXPLANATIONS")
print("==============================")

for i, plan in enumerate(top_plans):

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

    explanation = generate_explanation(
        rank=i+1,
        plan=plan,
        next_plan=next_plan
    )

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

```
Loading LLM...

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

LLM loaded.

==============================
DECISION-AWARE EXPLANATIONS
==============================

====================================
PLAN: irish_life_health_first_cover_table_of_cover_2026-01-01
====================================

You are a medical insurance ranking explanation engine.

RULES:
- Use ONLY the structured evidence provided.
- Do NOT invent benefits.
- Connect patient conditions to coverage.
- Explain ranking logically.
- Be concise and factual.
- Max 180 words.

PATIENT PROFILE:
```

```
{"age": 52, "chronic_conditions": ["heart_disease", "diabetes"], "hospital_admissions_last_2_years": 2, "specialist_visits_per_year": 6}

RISK LEVEL: HIGH

PLAN RANK: #1
PLAN ID: irish_life_health_first_cover_table_of_cover_2026-01-01
TOTAL SCORE: 2.0

SCORE BREAKDOWN:
{"inpatient_component": -5, "excess_penalty": 0.0, "disease_match": 7, "hightech_bonus": 0}

STRUCTURED EVIDENCE:
{
  "inpatient": "Inpatient consultant fees are not fully covered.",
  "excess": "Plan excess is \u20ac0.",
  "disease": [
    "Mentions 'cardiac' related to heart_disease.",
    "No explicit mention of diabetes coverage."
  ],
  "hightech": "High-tech hospital coverage not available or restricted."
}

TASK:
Explain why this plan ranked #1.
Link:
- inpatient component
- excess penalty
- disease match
- high-tech impact
Mention limitations clearly.
This plan ranked above irish_life_health_health_plan_26.1_table_of_cover_2026-01-01 because it achieved a higher total score.
The inpatient component penalizes this plan by not fully covering inpatient consultant fees, making it less covered for inpatient care. The excess penalty of £0
means there's no additional cost if the patient exceeds their coverage limit. This plan scores highly on the disease match component with 7 points for matching
the patient's chronic conditions (heart disease and diabetes). However, the high-tech coverage is not available, which further reduces its score. Despite the
low disease match score due to the absence of diabetes coverage, the overall score of 2.0 indicates that the disease match component outweighs the lack of high-
tech coverage, making this plan the top choice among the options. Therefore, this plan is ranked # #1. To rank this plan as the #1, the inpatient component
negatively impacts it, while the disease match component significantly boosts it due to the close match to the patient's chronic conditions. The lack of high-
tech coverage further enhances its ranking. The low disease match score for diabetes is compensated for by the strong match for heart disease, leading to an
overall score of 2.

==================================
PLAN: irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
==================================

You are a medical insurance ranking explanation engine.

RULES:
- Use ONLY the structured evidence provided.
- Do NOT invent benefits.
- Connect patient conditions to coverage.
- Explain ranking logically.
- Be concise and factual.
- Max 180 words.

PATIENT PROFILE:
{"age": 52, "chronic_conditions": ["heart_disease", "diabetes"], "hospital_admissions_last_2_years": 2, "specialist_visits_per_year": 6}
```

```
RISK LEVEL: HIGH

PLAN RANK: #2
PLAN ID: irish_life_health_health_plan_26.1_table_of_cover_2026-01-01
TOTAL SCORE: 2.0

SCORE BREAKDOWN:
{"inpatient_component": -5, "excess_penalty": 0.0, "disease_match": 7, "hightech_bonus": 0}

STRUCTURED EVIDENCE:
{
  "inpatient": "Inpatient consultant fees are not fully covered.",
  "excess": "Plan excess is \u20ac0.",
  "disease": [
    "Mentions 'cardiac' related to heart_disease.",
    "No explicit mention of diabetes coverage."
  ],
  "hightech": "High-tech hospital coverage not available or restricted."
}

TASK:
Explain why this plan ranked #2.
Link:
- inpatient component
- excess penalty
- disease match
- high-tech impact
Mention limitations clearly.
This plan ranked above irish_life_health_select_starter_table_of_cover_2026-01-01 because it achieved a higher total score.
The reasoning for the ranking is as follows:

1. Inpatient Component: The inpatient component score is -5, indicating that the plan has a significant cost burden for inpatient care, which is a major factor
in risk level assessment.
2. Excess Penalty: The excess penalty is 0, meaning there's no additional cost for using the excess benefit, which is less critical but still important.
3. Disease Match: The disease match score is 7, showing strong coverage for heart disease but not for diabetes, which is a key factor in the high-risk level.
4. High-Tech Impact: There's no mention of high-tech coverage, which could be beneficial for advanced treatments but isn't a deciding factor here.
5. Total Score: The total score of 2.0 indicates the plan is ranked with a moderate risk level, placing it below the highest plan (total score of 3.0 from the
next plan).

Hence, this plan is ranked #2 due to its moderate risk level and lower scores in all components compared to the highest plan. The inpatient

====================================
PLAN: irish_life_health_select_starter_table_of_cover_2026-01-01
====================================

You are a medical insurance ranking explanation engine.

RULES:
- Use ONLY the structured evidence provided.
- Do NOT invent benefits.
- Connect patient conditions to coverage.
- Explain ranking logically.
- Be concise and factual.
- Max 180 words.
```

```
PATIENT PROFILE:
{"age": 52, "chronic_conditions": ["heart_disease", "diabetes"], "hospital_admissions_last_2_years": 2, "specialist_visits_per_year": 6}

RISK LEVEL: HIGH

PLAN RANK: #3
PLAN ID: irish_life_health_select_starter_table_of_cover_2026-01-01
TOTAL SCORE: 2.0

SCORE BREAKDOWN:
{"inpatient_component": -5, "excess_penalty": 0.0, "disease_match": 7, "hightech_bonus": 0}

STRUCTURED EVIDENCE:
{
  "inpatient": "Inpatient consultant fees are not fully covered.",
  "excess": "Plan excess is \u20ac0.",
  "disease": [
    "Mentions 'cardiac' related to heart_disease.",
    "No explicit mention of diabetes coverage."
  ],
  "hightech": "High-tech hospital coverage not available or restricted."
}

TASK:
Explain why this plan ranked #3.
Link:
- inpatient component
- excess penalty
- disease match
- high-tech impact
Mention limitations clearly.

This plan ranks #3 due to significant inpatient costs and lack of high-tech coverage despite the patient having multiple chronic conditions. The disease match
score of 7 reflects the close match between diseases and coverage, but the high-risk profile (52 years old with 2 hospital admissions in last 2 years) impacts
negatively on the ranking. The excess penalty is low at 0, indicating no additional cost for high utilizers. The high-tech bonus is not applicable as it's not
offered by this plan. The patient's chronic conditions (heart disease and diabetes) are well-matched,as in plan to rank #3. The excess penalty is low,
indicating the plan is not penalized for high utilization. The high-risk profile impacts negatively, as the patient is older and has multiple chronic
conditions, increasing the risk level. The disease match score of 7 indicates that the diseases are well-matched, covered plan. The high-tech bonus is not
applicable, as it's not offered by this plan. The excess penalty is low, indicating the plan is not penalized for high utilization. The
```