

פרויקט — Metadata-Driven Hybrid RAG

רקע והסבר מושגים

- **RAG (Retrieval-Augmented Generation)** — שילוב של אחזור מידע ממסמכים עם מודל שפה גדול (LLM). המודל מקבל גם הקשר שנשלף מהמסמכים וגם את השאלה, ומחזיר תשובה מדויקת יותר.
- **Hybrid Retrieval (Dense + Sparse)** — שילוב בין אחזור סמנטי (ווקטורים צפופים) לבין אחזור מבוסס מילות (keywords) זה נותן איזון בין משמעות מילולית (BM25 לדוגמה, sparse) מפתח לבין קרבה סמנטית (keywords) זה נותן איזון בין משמעות מילולית (sparse, לדוגמה, embedding).
- **Metadata-Driven Retrieval** — לסינון (PageNumber, ClientId, SectionType) שימוש בשדות מטא-דאטה (כמו PageNumber, ClientId, SectionType) "תן לי רק טבלאות משנת 2023".
- **Reranker** — שמדרג מחדש את תוצאות האחזור כדי להבטיח שהקונטקסט הכי רלוונטי (cross-encoder כמו) מודל LLM-יישלה ל.
- **Agents & Routing** — Router כאשר (Summary Agent, Table-QA Agent) בניית מספר סוכנים ייעודיים. מנתב את השאלה לתת-הסוכן המתאים Agent.
- **Pinecone** — ים בסקייל גבוה, עם תמיכה-embedding לוווקטורים המאפשר שמירה, אחזור וניהול DB שירות — מובנית בסינון מטא-דאטה.

ארכיטקטורה (תרשים)

 קובץ נלווה: [hybrid_rag_architecture.png](#)


זרימת תהליך (Flowchart)

 קובץ נלווה: [hybrid_rag_flow.png](#)

Agent Router Flow (Flowchart)

אל הסוכן המתאים **Router Agent** בתרשים זה ניתן לראות כיצד השאלה עוברת דרך:

- **Router Agent** Classifier (Intent + Entity recognition) מקבל את השאלה ומבצע.
- **Summary Agent** → אם מדובר בשאלה כללית.
- **Table-QA Agent** → אם נדרשת תשובה מתוך טבלה.
- **Needle Agent** → אם מדובר במידע מאוד ספציפי.

 קובץ נלווה: [agent_router_flow.png](#) — לחיצה על הקישור תאפשר להוריד את הקובץ למחשב שלך

שלבי פיתוח

1) Parsing & Chunking

- חלוקה לפי מבנה טבעי: כותרות, סעיפים, טבלאות, תרשימים.
- Distillation: (עד ~5% מהמסמך) תקצוב הצ'אנק (עד ~5% מהמסמך).
- Anchors: PageNumber, TableId/FigureId, SectionType, מיקום, בצ'אנק.

2) Metadata (לפחות 5 שדות מתוך הרשימה)

- FileName
- PageNumber
- ChunkSummary
- Keywords
- CriticalEntities
- IncidentType (אם רלוונטי)
- IncidentDate (אם קיים)
- SectionType (Summary / Timeline / Table / Figure / Analysis / Conclusion)
- AmountRange (אם רלוונטי)
- TableId / FigureId (אם קיים)
- ClientId / CaseId

3) Indexing

- Markdown או CSV-טבלאות: המרה ל
- Anchors (TableId, PageNumber) שמירת
- של טקסט + תיאור מילולי לטבלה embedding יצירת
- (retrieval לסינון) שמירת מטאדאטה יחד עם הווקטורים
- metadata filtering, מרכזי לשמירת כל הווקטורים עם Vector DB כ-Pinecone-שימוש ב: **בנוסף** (ClientId למשל לכל) namespaces ב

4) Hybrid Retrieval

1. (NER + keyword extraction) שאילתה → חילוץ מילות מפתח, ישויות, תאריכים.
2. Dense Retrieval ($K \approx 10$) \cup Sparse Retrieval ($K \approx 10$).
3. Metadata (כמו ClientId, Year, SectionType) שימוש בפילטרים לפי.
4. Reranker לצמצום ל-6-8 תוצאות סופיות

5) Multi-Document Support

- (ClientId / CaseId) תמיכה במסמכים מרובים לכל לקוח.
- אפשרות לבצע אחזור רק מתיק אחד או מכלל המסמכים.
- namespaces בעזרת Pinecone-ניהול ב

6) Agent Structure

- **Router Agent** — מקבל שאילתה ומנתב
- **Summary Agent** — מסכם חלקים נבחרים
- **Needle Agent** — ספציפי Anchor / מאתר פסקה

קוד לדוגמה (Python + LangChain + Pinecone)

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_pinecone import PineconeVectorStore
from rank_bm25 import BM25Okapi
import pinecone

# 1. Chunking
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=800,
    chunk_overlap=100
)

chunks = text_splitter.split_text(document_text)

# 2. Metadata enrichment
chunk_metadata = []
for i, chunk in enumerate(chunks):
    meta = {
        "FileName": "case123.pdf",
        "PageNumber": i // 3 + 1,
        "SectionType": "Analysis",
        "ChunkSummary": chunk[:100],
        "Keywords": ["contract", "client"],
        "ClientId": "C123"
    }
    chunk_metadata.append(meta)

# 3. Indexing (Dense with Pinecone)
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
pinecone.init(api_key="YOUR_KEY", environment="us-east1-gcp")

# צירת אינדקס (פעם אחת)
if "hybrid-rag" not in pinecone.list_indexes():
    pinecone.create_index("hybrid-rag", dimension=1536, metric="cosine")

index = pinecone.Index("hybrid-rag")

vectorstore = PineconeVectorStore.from_texts(
    texts=chunks,
    embedding=embeddings,
    metadatas=chunk_metadata,
```

```

    index_name="hybrid-rag",
    namespace="C123" # Namespace לפי ClientId
)

# 4. Sparse Index (BM25)
tokenized_corpus = [chunk.split(" ") for chunk in chunks]
bm25 = BM25Okapi(tokenized_corpus)

# 5. Hybrid Retrieval function
def hybrid_search(query, top_k=10, namespace="C123"):
    # Dense retrieval from Pinecone
    dense_results = vectorstore.similarity_search(
        query, k=top_k, namespace=namespace, filter={"ClientId": "C123"}
    )

    # Sparse retrieval
    tokenized_query = query.split(" ")
    scores = bm25.get_scores(tokenized_query)
    sparse_results = [chunks[i] for i in scores.argsort()[-top_k:][::-1]]

    # Combine + rerank (placeholder for cross-encoder)
    combined = dense_results + sparse_results
    return combined[:6]

```

קריטריוני הערכה (Evaluation)

- **Answer Correctness** – השוואה ל-Ground Truth.
- **Context Precision** ≥ 0.75
- **Context Recall** ≥ 0.70
- **Faithfulness** ≥ 0.85
- **Table-QA Accuracy** ≥ 0.90

סיכום

אחזור היברידי, ניהול מטא-דאטה עשיר: **Metadata-Driven Hybrid RAG** בפרויקט זה הראינו כיצד ניתן להקים מערכת עם שלושה תתי-סוכנים ייעודיים כדי לטפל Router Agent ריראנקר, ותמיכה במסמכים מרובים. בנוסף, שילבנו מבנה של בסוגי שאילתות שונים.

ים בסקייל גדול, כולל סינון לפי-embedding מרכזי לניהול Vector Database כ-Pinecone -כבנווס, שולב שימוש ב-Metadata ו-namespaces לקוח.

קובץ נלווה: [hybrid_rag_architecture.png](#) \ קובץ נלווה: [hybrid_rag_flow.png](#) \ קובץ נלווה: [agent_router_flow.png](#)