Neo4j GraphRAG Retrievers Overview

From Knowledge Graph to Answers

You have a knowledge graph with:

- **Entities**: Companies, products, risks, executives
- **Relationships**: OWNS, FACES_RISK, MENTIONS, WORKS_FOR
- **Embeddings**: Vector representations for semantic search

The question: How do you *retrieve* the right information to answer user questions?

What is a Retriever?

A **retriever** searches your knowledge graph and returns relevant information.

Three retrieval patterns:

Retriever	What It Does
Vector	Semantic similarity search across text chunks
Vector Cypher	Semantic search + graph traversal for relationships
Text2Cypher	Natural language → Cypher query for precise facts

Each pattern excels at different question types.

The GraphRAG Class

Retrievers work with the **GraphRAG** class, which combines retrieval with LLM generation:

```
User Question
    ↓
Retriever finds relevant context
    ↓
Context passed to LLM
    ↓
LLM generates grounded answer
```

The retriever's job is finding the right context. The LLM's job is generating a coherent answer from that context.

Vector Retriever

How it works:

- Converts your question to an embedding
- Finds chunks with similar embeddings
- Returns semantically related content

Best for:

- "What is Apple's strategy?"
- "Tell me about cybersecurity threats"
- Conceptual, exploratory questions

Limitation: Returns text chunks only—no entity relationships.

Vector Cypher Retriever

How it works:

- Vector search finds relevant chunks
- Custom Cypher query traverses from chunks to related entities
- Returns content + structured data

Best for:

- "Which asset managers are affected by crypto regulations?"
- "What risks do tech companies face?"
- Questions needing both content and relationships

Key insight: The chunk is the anchor—you traverse from what vector search finds.

Text2Cypher Retriever

How it works:

- LLM converts natural language to Cypher
- Query executes against the graph
- Returns precise, structured results

Best for:

- "How many risk factors does Apple face?"
- "List all companies owned by BlackRock"
- Counts, lists, specific lookups

Limitation: Question must map to graph schema.

Choosing the Right Retriever

Question Pattern	Best Retriever
"What is...", "Tell me about..."	Vector
"Which [entities] are affected by..."	Vector Cypher
"How many...", "List all..."	Text2Cypher
Content about topics	Vector
Content + relationships	Vector Cypher
Facts, counts, aggregations	Text2Cypher

The Decision Framework

Ask yourself:

1. **Am I looking for content or facts?**
    - Content → Vector or Vector Cypher
    - Facts → Text2Cypher
2. **Do I need related entities?**
    - No → Vector
    - Yes → Vector Cypher
3. **Is this about relationships?**
    - Traversals → Vector Cypher or Text2Cypher
    - Semantic → Vector

Summary

In this lesson, you learned:

- **Retrievers** search and return relevant information from your knowledge graph
- **Vector Retriever**: Semantic similarity search across chunks
- **Vector Cypher Retriever**: Semantic search + graph traversal
- **Text2Cypher Retriever**: Natural language to precise database queries
- **Each excels at different question types**—choosing the right one matters

Next: Deep dive into each retriever type.

Vector Retriever

What is a Vector Retriever?

The **simplest retriever**—finds content by meaning, not keywords.

How it works:

1. Convert your question to an embedding
2. Search vector index for similar chunk embeddings
3. Return the most semantically similar chunks

Key insight: "Cybersecurity threats" finds content about "data breaches" and "hacking risks" even without exact word matches.

Creating a Vector Retriever

```
from neo4j_graphrag.retrievers import VectorRetriever

vector_retriever = VectorRetriever(
    driver=driver,                      # Neo4j connection
    index_name='chunkEmbeddings',       # Vector index name
    embedder=embedder,                  # Embedding model
    return_properties=['text']          # Properties to return
)
```

Components:

- **Driver**: Connection to Neo4j
- **Index**: Where embeddings are stored
- **Embedder**: Model that creates embeddings (e.g., OpenAI)

Performing a Search

```
query = "What are the risks that Apple faces?"

results = vector_retriever.search(
    query_text=query,
    top_k=5  # Return 5 most similar chunks
)

for record in results.records:
    print(f"Score: {record['score']:.4f}")
    print(f"Text: {record['text'][:200]}...")
```

Each result includes:

- **text**: The chunk content
- **score**: Similarity score (0-1, higher = more similar)

Understanding Similarity Scores

Score Range	Interpretation
0.95-1.0	Extremely similar (near-exact match)
0.90-0.95	Highly relevant
0.85-0.90	Relevant
0.80-0.85	Moderately relevant
< 0.80	Weak relevance

Higher scores indicate stronger semantic matches.

Best For

Use Vector Retriever when:

- Finding conceptually similar content
- Questions like "What is...", "Tell me about...", "Explain..."
- Exploratory questions about topics
- When exact keywords don't match but meaning does

Example questions:

- "What is Apple's business strategy?"
- "Describe cybersecurity threats"
- "What challenges do tech companies face?"

Limitations

Vector Retriever returns text only:

- No entity relationships
- No structured data from the graph
- Can't aggregate across entities
- Can't traverse connections

Example limitation:

- Question: "What risks does Apple face?"
- Returns: Chunks about risks (may not be Apple-specific)
- Missing: Structured FACES_RISK relationships

When you need more: Use Vector Cypher Retriever.

The top_k Parameter

Controls how many results to return:

top_k	Trade-off
1-3	Fastest, most relevant only
5-10	Balanced coverage
15-20	Maximum coverage, may include less relevant

Rule of thumb: Start with 5, adjust based on results.

Summary

Vector Retriever is your foundation for semantic search:

- **Converts queries to embeddings** for meaning-based search
- **Returns semantically similar chunks** regardless of keywords
- **Best for** content questions, topic exploration
- **Limitation:** No graph relationships or structured data

Next: Learn how Vector Cypher Retriever adds graph intelligence.

Vector Cypher Retriever

Beyond Basic Vector Search

Vector Retriever: Returns text chunks only.

Vector Cypher Retriever: Returns text chunks + related entities from graph traversal.

```
Query: "What risks affect companies?"
    ↓
Vector Search: Find relevant chunks
    ↓
Graph Traversal: From chunks → Companies → RiskFactors → AssetManagers
    ↓
Result: Content + structured entity data
```

How It Works

Two-step process:

1. **Vector Search** (semantic)
    - Find chunks similar to your question
    - Same as Vector Retriever
2. **Cypher Traversal** (structural)
    - From each chunk, traverse the graph
    - Gather related entities and relationships
    - Return enriched context

The combination: Semantic relevance + graph intelligence.

Creating a Vector Cypher Retriever

```
from neo4j_graphrag.retrievers import VectorCypherRetriever

retrieval_query = """
MATCH (node)-[:FROM_DOCUMENT]-(doc:Document)-[:FILED]-(company:Company)
OPTIONAL MATCH (company)-[:FACES_RISK]->(risk:RiskFactor)
WITH node, score, company, collect(risk.name)[0..20] AS risks
RETURN node.text AS text, score,
       {company: company.name, risks: risks} AS metadata
ORDER BY score DESC
"""

retriever = VectorCypherRetriever(
    driver=driver,
    index_name='chunkEmbeddings',
    embedder=embedder,
    retrieval_query=retrieval_query
)
```

Understanding the Retrieval Query

The library provides automatically:

```
CALL db.index.vector.queryNodes($index_name, $top_k, $embedding)
YIELD node, score
-- Your query starts here with node and score --
```

Your retrieval_query:

- Receives `node` (matched chunk) and `score` (similarity)
- Traverses from node to related entities
- Returns enriched results

Query Breakdown

```
-- Traverse from chunk to company
MATCH (node)-[:FROM_DOCUMENT]-(doc:Document)-[:FILED]-(company:Company)

-- Get related risks (OPTIONAL so companies without risks still appear)
OPTIONAL MATCH (company)-[:FACES_RISK]->(risk:RiskFactor)

-- Aggregate risks, limit to 20
WITH node, score, company, collect(risk.name)[0..20] AS risks

-- Return chunk text + metadata
RETURN node.text AS text, score,
       {company: company.name, risks: risks} AS metadata
```

Why OPTIONAL MATCH Matters

Without OPTIONAL MATCH:

```
MATCH (company)-[:FACES_RISK]->(risk)
```

Only returns companies that *have* risk factors.

With OPTIONAL MATCH:

```
OPTIONAL MATCH (company)-[:FACES_RISK]->(risk)
```

Returns *all* companies; risks list is empty if none exist.

Use OPTIONAL MATCH for complete results.

Best For

Use Vector Cypher Retriever when:

- You need content AND related entities
- Questions involve relationships
- You want to traverse from relevant content to connected data

Example questions:

- "Which asset managers are affected by cryptocurrency policies?"
- "What products do companies mention alongside AI?"
- "What risks connect to companies in the tech sector?"

The Chunk as Anchor

Critical concept: You can only traverse from what vector search finds.

Example problem:

- Query: "What risks does Apple face?"
- Vector search finds: Chunks about "risk management" (not Apple-specific)
- Traversal: Goes to companies mentioned in those chunks
- Result: May not include Apple!

Solution: Ensure your question surfaces relevant chunks, or use Text2Cypher for entity-specific queries.

When to Use Vector Cypher vs Text2Cypher

Question Type	Best Retriever
Content + related entities	Vector Cypher
Specific entity facts	Text2Cypher
"What does [topic] affect?"	Vector Cypher
"How many [entities]?"	Text2Cypher
Semantic + relationships	Vector Cypher
Precise counts/lists	Text2Cypher

Summary

Vector Cypher Retriever combines semantic search with graph traversal:

- **Two-step process:** Vector search → Graph traversal
- **Custom Cypher query** defines what entities to gather
- **Returns:** Text chunks + structured metadata
- **Best for:** Questions needing content AND relationships
- **Key insight:** The chunk is the anchor for traversal

Next: Learn Text2Cypher for precise, entity-specific queries.

Text2Cypher Retriever

From Natural Language to Database Queries

The problem: Some questions need precise facts, not semantic search.

Text2Cypher solution:

1. User asks a question in natural language
2. LLM generates a Cypher query from the question
3. Query executes against the graph
4. Precise, structured results returned

Example:

- Question: "How many risk factors does Apple face?"
- Generated: `MATCH (c:Company {name:'APPLE INC'})-[:FACES_RISK]->(r) RETURN count(r)`
- Result: `45`

How It Works

```
User: "Which companies does BlackRock own?"
    ↓
[LLM + Schema] → Generate Cypher
    ↓
MATCH (am:AssetManager {managerName: 'BlackRock Inc.'})
      -[:OWNS]->(c:Company)
RETURN c.name
    ↓
[Execute Query]
    ↓
Result: Apple Inc., Microsoft Corp., Alphabet Inc., ...
```

Creating a Text2Cypher Retriever

```
from neo4j_graphrag.retrievers import Text2CypherRetriever
from neo4j_graphrag.schema import get_schema

# Schema tells LLM what's queryable
schema = get_schema(driver)

text2cypher_retriever = Text2CypherRetriever(
    driver=driver,
    llm=llm,                    # LLM for Cypher generation
    neo4j_schema=schema         # Graph structure
)
```

The schema is critical: Without it, the LLM guesses (often incorrectly).

The Role of Schema

Schema tells the LLM:

```
Node properties:
  Company {name: STRING, ticker: STRING}
  RiskFactor {name: STRING}
  AssetManager {managerName: STRING}

Relationships:
  (:Company)-[:FACES_RISK]->(:RiskFactor)
  (:AssetManager)-[:OWNS]->(:Company)
```

With schema: LLM knows exactly what entities and relationships exist.

Without schema: LLM invents non-existent properties and relationships.

Best For

Use Text2Cypher when:

- You need precise facts, counts, or lists
- Question is about specific entities
- Aggregations are needed
- Direct graph queries (no semantic search)

Example questions:

- "How many risk factors does Apple face?"
- "List all companies owned by Vanguard"
- "Which company has the most products?"
- "What is the average number of risks per company?"

Performing a Search

```
query = "What companies does BlackRock own?"

results = text2cypher_retriever.search(query_text=query)

# Results contain:
# - The generated Cypher query
# - The query results
for record in results.records:
    print(record)
```

Behind the scenes: LLM analyzes your question, generates Cypher, executes it.

Limitations

Text2Cypher requires questions that map to schema:

- Question: "What's the sentiment about AI regulation?"
- Problem: No "sentiment" property in schema
- Result: Cannot generate valid query

Text2Cypher may struggle with:

- Ambiguous questions
- Questions requiring interpretation
- Content that lives in text chunks (use Vector instead)

Security Considerations

Text2Cypher executes LLM-generated queries. Important safeguards:

- **Use read-only credentials**: Prevent accidental data modification
- **Validate queries**: Check for dangerous operations (DELETE, DROP)
- **Limit results**: Ensure LIMIT clauses prevent unbounded returns
- **Monitor usage**: Log generated queries for review
- **Trust boundaries**: Don't expose to untrusted users

Generated Query Quality

LLMs may generate imperfect Cypher:

- Syntax errors
- Deprecated syntax
- Non-existent properties
- Inefficient patterns

Mitigation:

- Use custom prompts to guide Cypher generation
- Validate generated queries
- Handle errors gracefully

Comparing All Three Retrievers

Question	Best Retriever	Why
"What is AI safety?"	Vector	Semantic content
"Which companies mention AI?"	Vector Cypher	Content + entities
"How many companies mention AI?"	Text2Cypher	Precise count
"Tell me about Apple"	Vector	Exploratory content
"List Apple's risks"	Text2Cypher	Specific entity facts

Summary

Text2Cypher Retriever converts natural language to database queries:

- **LLM generates Cypher** from your question
- **Schema guides generation** for accuracy
- **Best for:** Facts, counts, lists, specific entities
- **Limitation:** Questions must map to graph schema

You now know all three retrieval patterns:

- Vector: Semantic content
- Vector Cypher: Content + relationships
- Text2Cypher: Precise facts

Next: Learn to build agents that choose the right retriever automatically.