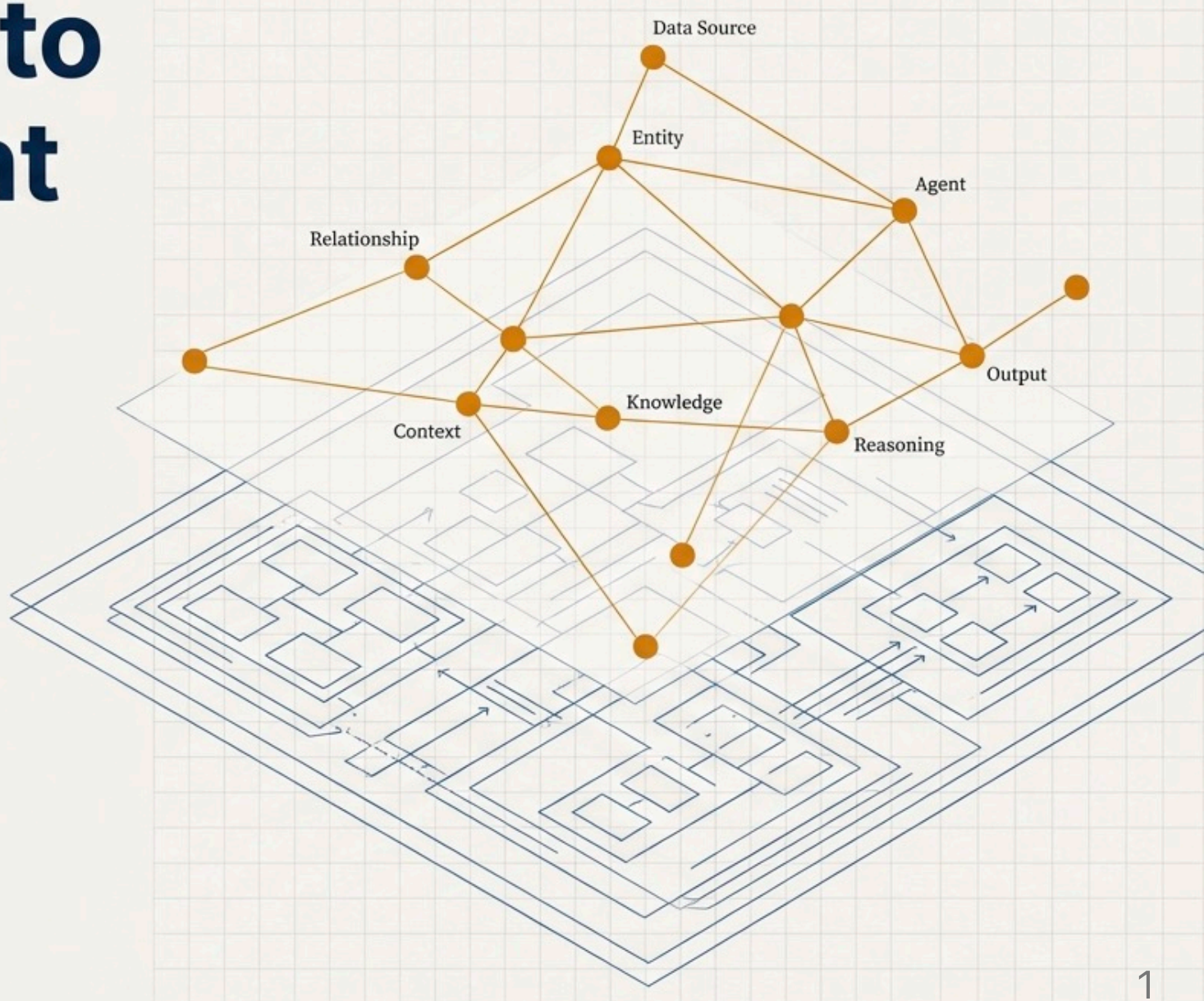
From Raw Data to Intelligent Agent

A Blueprint for Building Production-Ready AI with GraphRAG on Neo4j and Azure



neo4j | Microsoft Azure

Some of the Neo4j's customers who run on Azure

Neo4j Aura: Cloud Graph Database

What is Neo4j Aura?

Neo4j Aura is a **fully managed cloud graph database service** that eliminates the operational overhead of running a graph database.

Key Characteristics:

- **Fully managed** - No infrastructure to maintain
- **Scalable** - Automatically scales with your data and queries
- **Secure** - Enterprise-grade security and compliance
- **Available everywhere** - Deploy in AWS, GCP, or Azure regions

Why Use a Graph Database?

Traditional databases struggle with **connected data**:

Scenario	Relational DB	Graph DB
"Find friends of friends"	Complex JOINs, slow	Natural traversal, fast
"What impacts what?"	Multiple queries	Single query
"How are these connected?"	Hard to express	Native pattern matching

Graphs excel at relationship-heavy queries that would require dozens of JOINs in SQL.

The Value of Aura for AI/GenAI

Neo4j Aura provides unique capabilities for building AI applications:

GraphRAG Foundation:

- Store knowledge graphs that power AI agents
- Vector search for semantic similarity
- Graph traversal for relationship reasoning

Production-Ready:

- Built-in vector indexes for embeddings
- Cypher query language for complex retrieval
- APIs for integration with LLM frameworks

Graph Analytics in Explore

The **Explore** tool includes built-in graph algorithms for visual analysis:

Available in Explore:

Category	Algorithms
Centrality	Betweenness, Degree, Eigenvector, PageRank
Community Detection	Label Propagation, Louvain, Weakly Connected Components

Full Algorithm Library (65+):
Neo4j Aura Graph Analytics provides the complete library via serverless compute with Zero ETL:

Category	Additional Algorithms	Use Cases
Similarity	Node Similarity, K-Nearest Neighbors	Recommendations, duplicate detection
Path Finding	Dijkstra, A*, Yen's K-Shortest	Routing, supply chain optimization
Link Prediction	Common Neighbors, Adamic Adar	Predict future connections
Node Embeddings	FastRP, GraphSAGE, Node2Vec	ML feature generation

Aura Tools: Query Workspace

The **Query Workspace** is a developer-friendly environment for Cypher:

Core Features:

- Write and execute Cypher queries against your database
- Syntax highlighting and auto-completion
- Save and organize query collections
- Export results in multiple formats

Query Log Forwarding:

- Send logs to your cloud logging service
- Better compliance, monitoring, and operational visibility
- Manage directly from Aura console

Aura Tools: Explore

Explore (powered by Neo4j Bloom) is a visual graph exploration tool:

Visual Graph Scene:

- Interactive canvas showing your graph data
- Click and drag nodes to arrange layouts
- Export as PNG, CSV, or shareable scenes

Search-First Experience:

- Natural language and pattern-based search
- "Show me a graph" sample queries
- Find nodes and relationships without Cypher

AI-Powered Features:

- GenAI Copilot for query assistance
- Find hidden connections automatically

Aura Tools: Dashboards

Dashboards in the Neo4j Console provide data visualization capabilities with low code / no code:

Visualization Types:

- Bar charts, line charts, pie charts, etc.
- Geographic maps
- **3D graph visualizations** (WebGL-powered)

GenAI Copilot:

- AI-powered dashboard creation
- Natural language to visualization

Enterprise Ready:

- SSO integration
- Role-based access

Aura Agents: No-Code GraphRAG

Coming up in this workshop: Neo4j Aura Agents

Aura Agents let you build **AI-powered conversational interfaces** to your graph:

- **No code required** - Configure through a simple UI
- **Natural language queries** - Ask questions in plain English
- **Automatic Cypher generation** - LLM translates questions to graph queries
- **Knowledge graph reasoning** - Leverage relationships for better answers

Why Agents matter:

- Democratize access to graph insights
- Build chatbots that understand your domain
- Combine vector search + graph traversal automatically

Summary

Neo4j Aura provides:

- **Managed graph database** - Focus on your data, not infrastructure
- **Graph-native storage** - Relationships are first-class citizens
- **AI/GenAI capabilities** - Vector indexes, GraphRAG support
- **Graph Analytics** - Built-in algorithms for insights
- **Integrated Tools** - Query, Explore, and Dashboards
- **Aura Agents** - No-code conversational AI over your graph

Next: Learn about Aura Agents for no-code GraphRAG applications.

The GenAI Promise and Its Limits

What Generative AI Does Well

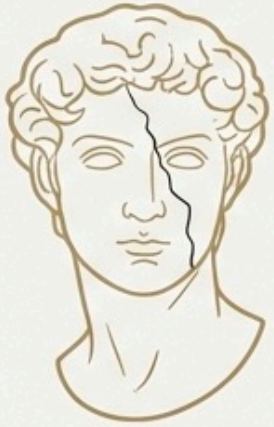
LLMs excel at tasks that rely on pattern recognition and language fluency:

- **Text generation**: Creating human-like responses, summaries, explanations
- **Language understanding**: Parsing intent, extracting meaning, following instructions
- **Pattern completion**: Continuing sequences, filling in blanks, generating variations
- **Translation and transformation**: Converting between formats, styles, languages

These capabilities emerge from training on vast amounts of text data.

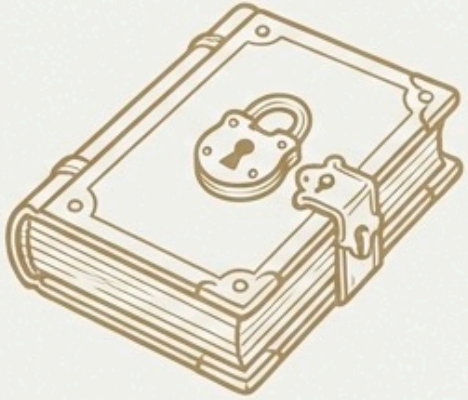
The Three Gaps in the GenAI Promise

LLMs excel at text generation, language understanding, and pattern completion. However, production systems must address three fundamental limitations.



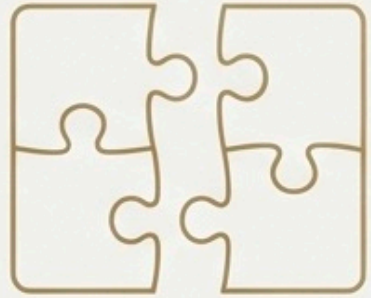
Hallucination (Confident but Wrong)

LLMs generate statistically *probable* text, not factually *verified* text. When they don't know, they fabricate with full confidence.



Knowledge Cutoff (Your Data is Invisible)

Models are trained on public data with a cutoff date. They have no access to your private company documents, databases, or real-time events.



Relationship Blindness (Can’t Connect the Dots)

LLMs process text sequentially and struggle to answer questions that require reasoning over relationships across multiple documents or sources.

1. Hallucination: Confident But Wrong

LLMs generate responses based on statistical likelihood, not factual verification.

The Problem:

- Produces the most *probable* continuation, not the most *accurate*
- Doesn't say "I don't know"—generates plausible-sounding text instead
- Complete with fabricated details and citations

Real Example: In 2023, US lawyers were sanctioned for submitting an LLM-generated brief with six fictitious case citations.

2. Knowledge Cutoff: No Access to Your Data

LLMs are trained at a specific point in time on publicly available data.

They don't know:

- Recent events after their training cutoff
- Your company's documents, databases, or internal knowledge
- Real-time data: current prices, live statistics, changing conditions

The Risk: Ask about your Q3 results or last week's board meeting, and the LLM may still generate a confident (and wrong) response.

3. Relationship Blindness: Can't Connect the Dots

LLMs process text sequentially and treat each piece in isolation.

Questions they struggle with:

- "Which asset managers own companies facing cybersecurity risks?"
- "What products are mentioned by companies that share risk factors?"
- "How are these two companies connected through their executives?"

These questions require *reasoning over relationships*—connecting entities across documents and traversing chains of connections.

The Solution: Providing Context

All three limitations have a common solution—**providing context**.

When you give an LLM relevant information in its prompt:

- It has facts to work with (reduces hallucination)
- It can access your specific data (overcomes knowledge cutoff)
- You can structure that information to show relationships (enables reasoning)

This is the foundation of **Retrieval-Augmented Generation (RAG)**.

Summary

In this lesson, you learned about the fundamental limitations of LLMs:

- **Hallucination**: LLMs generate probable responses, not verified facts
- **Knowledge cutoff**: LLMs can't access recent events or your private data
- **Relationship blindness**: LLMs struggle with cross-document reasoning

The solution is providing context—which leads us to RAG.

Next: Learn how traditional RAG works and why it has its own limitations.

Traditional RAG: Chunking and Vector Search

Why RAG Was Adopted

Remember the LLM limitations we discussed:

- **Hallucination** - Generates confident but wrong information
- **Knowledge cutoff** - No access to your private data
- **Relationship blindness** - Can't connect information

The insight: If we could provide LLMs with relevant context, we could address these limitations.

This led to **Retrieval-Augmented Generation (RAG)**.

The Power of Context

Providing context in prompts dramatically improves LLM responses.

When you include relevant information, the model can:

- Generate accurate summaries grounded in actual documents
- Answer questions about your specific data
- Reduce hallucination by having facts to reference

RAG automates this: Instead of manually adding context, retrieve it automatically based on the user's question.

How Traditional RAG Works

Traditional RAG follows a simple pattern:

1. **Index documents**: Break documents into chunks and create embeddings
2. **Receive query**: User asks a question
3. **Retrieve context**: Find chunks with embeddings similar to the query
4. **Generate response**: Pass retrieved chunks to LLM as context

Let's understand each component: chunking, embeddings, and vector search.

Why Chunking Matters

LLMs have **context window limits**—they can only process so much text at once.

The problem:

- Documents can be thousands of pages
- You can't send everything to the LLM
- You need to find the *relevant* parts

The solution: Break documents into smaller **chunks** that can be:

- Indexed for search
- Retrieved when relevant
- Fit within the LLM's context window

Common Chunking Strategies

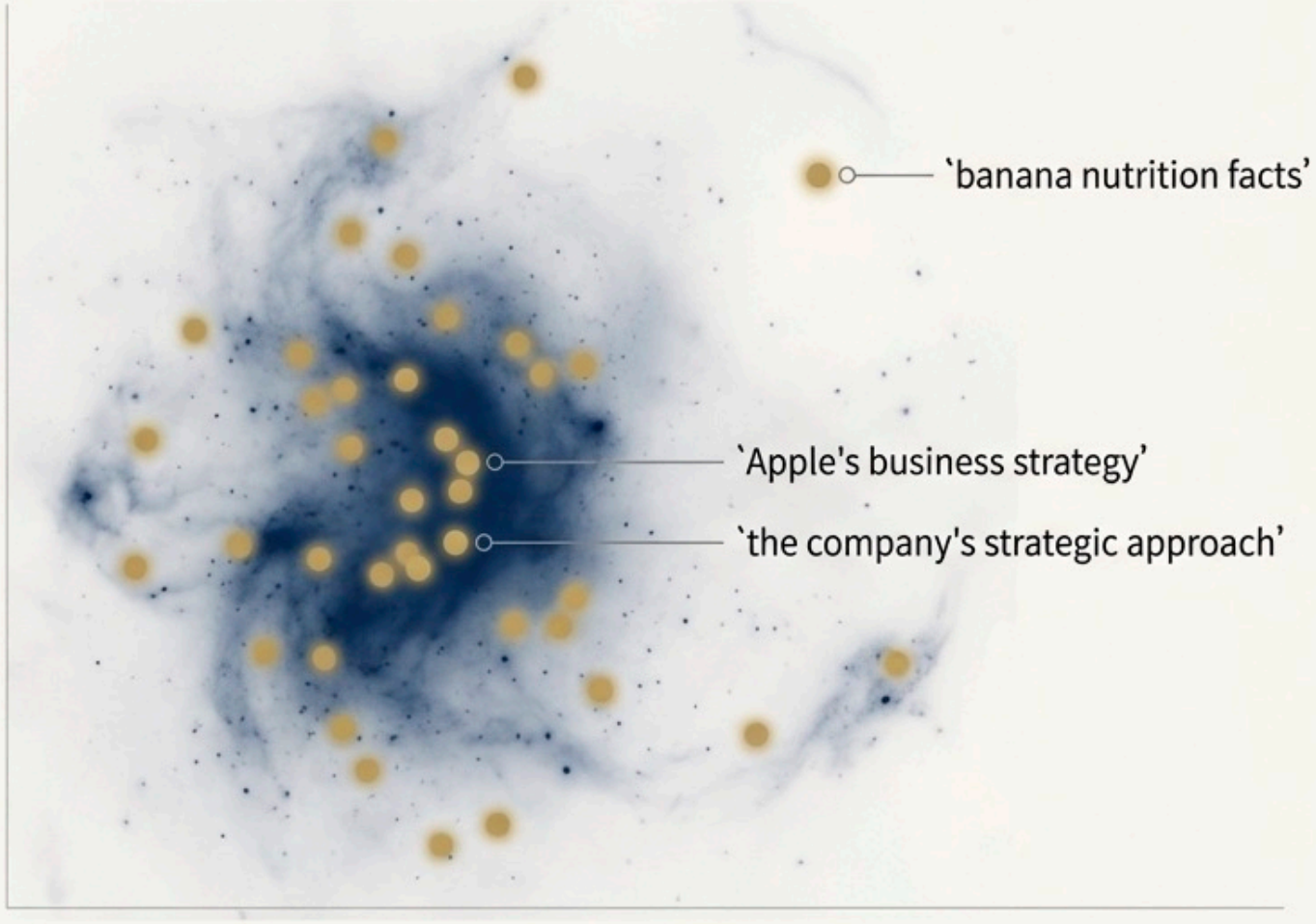
Strategy	How It Works	Best For
Fixed-size	Split every N characters/tokens	Simple, predictable
Sentence	Split on sentence boundaries	Preserving complete thoughts
Paragraph	Split on paragraph breaks	Structured documents
Semantic	Split when topic changes	Long-form content
Recursive	Try multiple strategies in order	General purpose

Overlap between chunks helps preserve context at boundaries.

From Words to Meaning: The Power of Embeddings

Key Concept
Embeddings are numerical representations of text as high-dimensional vectors (e.g., 1,536 dimensions for OpenAI models).
Core Principle
The most important property of embeddings is that **similar meanings produce similar vectors**.

GALAXY OF MEANING (2D Projection of Semantic Space)



Semantic Space (2D Projection)

The phrases 'Apple's business strategy' and 'the company's strategic approach' have very different words, but their embeddings are nearly identical because their meaning is the same.

What is a Vector?

Vectors are lists of numbers.

The vector `[1, 2, 3]` represents a point in three-dimensional space.

In machine learning, vectors can represent much more complex data—including the *meaning* of text.

The Smart Librarian Analogy

Think of embeddings like having a **really smart librarian** who has read every book in the library.

Traditional catalog (keywords):

- Books organized by title, author, subject
- Search for "dogs" only finds books with "dogs" in the title/subject
- Miss books about "canines," "puppies," or "pets"

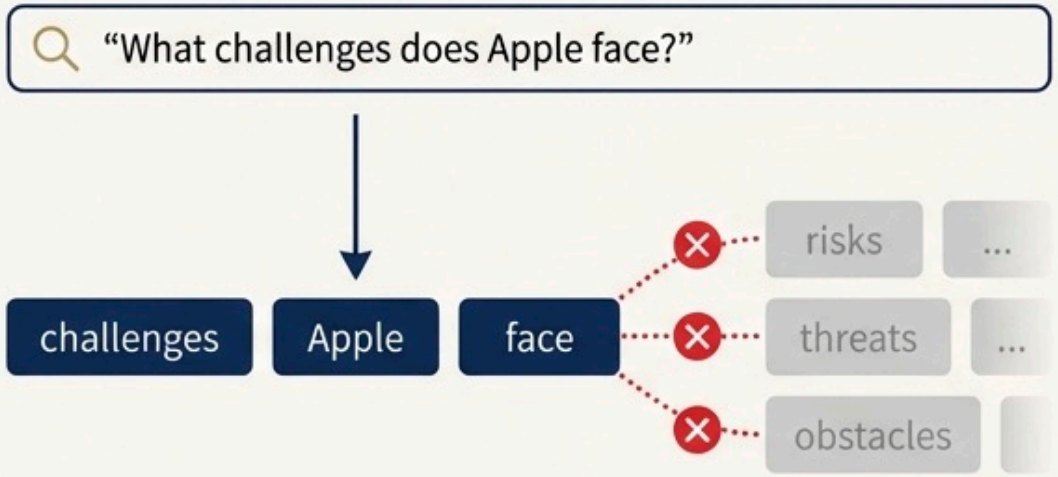
Smart librarian (embeddings):

- Understands what each book is *about*
- "I want something about loyal companions" → finds dog books, even without the word "dog"
- Organizes by meaning, not just labels

Beyond Keywords: Searching by Meaning

Without Vectors (Keyword Search)

Brittle Keyword Matching

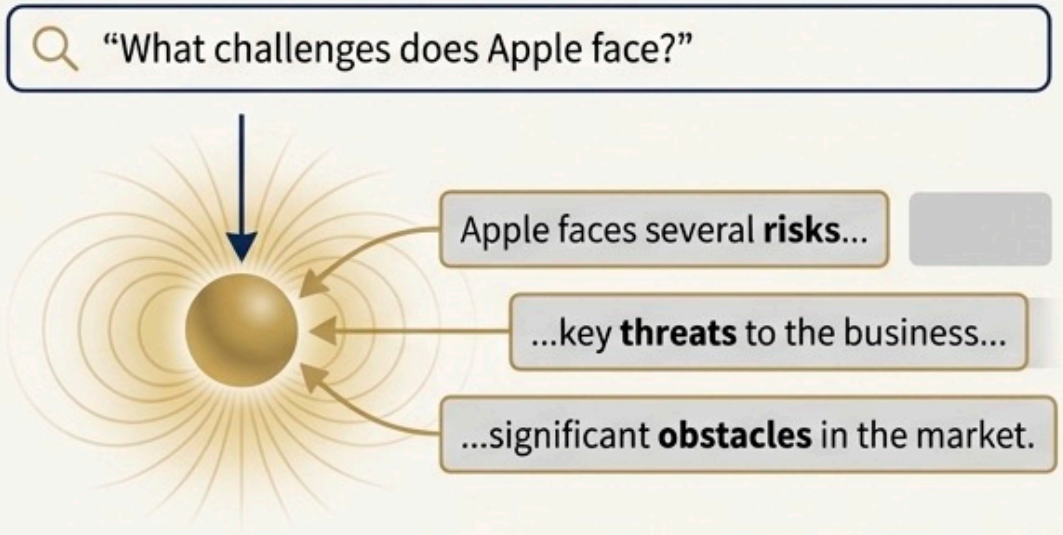


Logic: The system requires exact keyword matches.

✗ Fails to find relevant documents that use synonyms, leading to incomplete or no results.

With Vectors (Semantic Search)

Flexible Semantic Search



Logic: The system converts the query to an embedding and finds text chunks with similar *meaning*.

✓ Successfully retrieves chunks discussing 'risks,' 'threats,' and 'obstacles,' providing a comprehensive answer regardless of the specific words used.

Similarity Search

Vector similarity is typically measured by **cosine similarity**—the angle between two vectors:

Score	Meaning
Near 1.0	Very similar meanings
Near 0.5	Somewhat related
Near 0.0	Unrelated

When you search, your question becomes an embedding, and the system finds chunks with embeddings close to your question.

The RAG Retrieval Flow

```
User Question
      ↓
Create embedding of question
      ↓
Compare to all chunk embeddings
      ↓
Return top K most similar chunks
      ↓
Send chunks + question to LLM
      ↓
LLM generates answer using chunks as context
```

Traditional RAG: What It Enables

Works well for:

- "What does this document say about X?"
- Finding relevant passages by topic
- Answering questions within a single document

The foundation of modern AI assistants, but as we'll see, it has important limitations when dealing with connected information.

Summary

In this lesson, you learned:

- **Chunking** breaks documents into searchable pieces
- **Embeddings** encode text meaning as vectors
- **Similar meanings** produce similar vectors
- **Semantic search** finds relevant content by meaning, not keywords
- **Traditional RAG** combines these to provide context to LLMs

Next: Understanding the limits of traditional RAG and how GraphRAG addresses them.

The Limits of Traditional RAG

RAG Helps, But Introduces New Challenges

We've seen how RAG provides context to LLMs:

- Retrieves relevant chunks based on semantic similarity
- Grounds responses in actual documents
- Reduces hallucination

But traditional RAG also introduced new problems:

- Retrieves similar content, not necessarily *relevant* content
- Misses relationships between pieces of information
- Can actually make responses *worse* when context is poor

The Problem with Traditional RAG

Traditional RAG treats documents as isolated, unstructured blobs.

What traditional RAG sees:

```
Chunk 1: "Apple Inc. faces cybersecurity risks including..."
Chunk 2: "BlackRock Inc. holds shares in technology companies..."
Chunk 3: "The semiconductor supply chain impacts..."
```

What traditional RAG misses:

- Which specific companies does BlackRock own?
- Do any of those companies face cybersecurity risks?
- How are supply chain issues connected to specific products?

Retrieves Similar Content, Not Connected Information

Traditional RAG can find text about cybersecurity and text about BlackRock.

But it can't tell you:

- Which asset managers are exposed to cybersecurity risks through their holdings

Why? Each chunk is independent—there's no understanding of how information connects.

Context ROT: When More Context Makes Things Worse

A surprising discovery: **too much irrelevant context degrades LLM performance**.

What happens:

- RAG retrieves chunks that are *similar* but not truly *relevant*
- The LLM's context window fills with tangentially related information
- The model gets confused, distracted, or misled by the noise

This became known as "Context ROT" (Retrieval of Tangents)—the retrieved context actually *rots* the quality of the response.

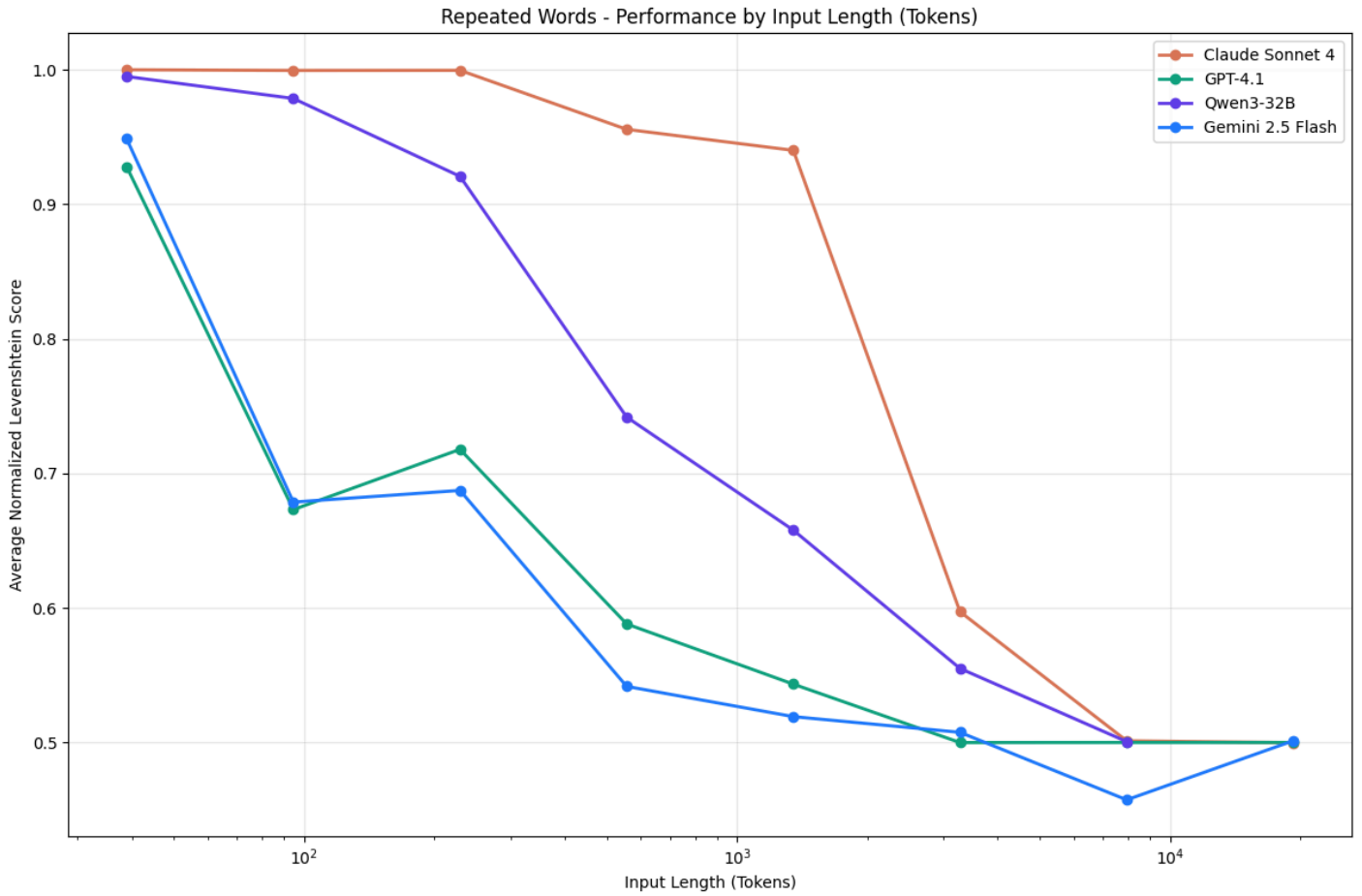
Context ROT: The Research

Research shows that as irrelevant context increases, LLM accuracy **decreases dramatically**.

The graph shows how adding more retrieved chunks often hurts rather than helps.

Key insight: Quality of context matters more than quantity.

Source: Chroma Research - Context ROT



Questions Traditional RAG Can't Answer

Question	Why Traditional RAG Struggles
"Which asset managers own companies facing cyber risks?"	Requires connecting ownership data to risk mentions
"What products are mentioned by companies that share risk factors?"	Requires finding shared entities across documents
"How many companies mention supply chain issues?"	Requires aggregation, not similarity search
"What executives work for companies in the tech sector?"	Requires traversing entity relationships

These questions need *structured context* that preserves relationships.

From Unstructured to Structured

The core insight: Information isn't truly unstructured.

Documents contain:

- **Entities**: Companies, people, products, risks
- **Relationships**: Owns, faces, mentions, works for

Traditional RAG ignores this structure. It treats a document as a bag of words to embed and search.

The GraphRAG Solution

GraphRAG extracts structure, creating a *knowledge graph* that preserves:

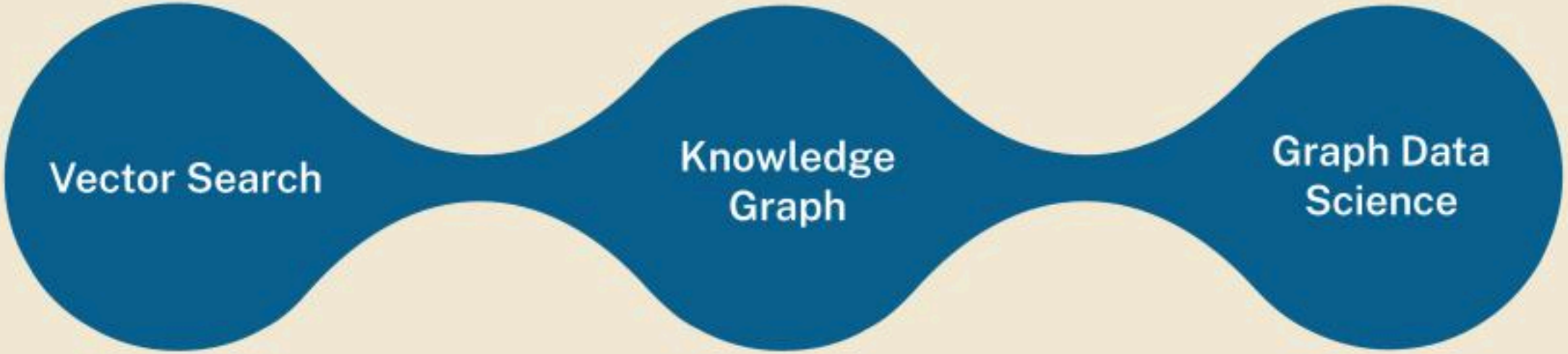
- **Entities**: The things mentioned in documents
- **Relationships**: How those things connect
- **Properties**: Attributes and details about entities

Traditional RAG asks: "What chunks are similar to this query?"

GraphRAG asks: "What entities and relationships are relevant to this query?"

RAG with Neo4j - graphRAG

Unify vector search, knowledge graph and data science capabilities to improve RAG quality and effectiveness



Find similar documents and content

Identify entities associated to content and patterns in connected data

Improve GenAI inferences and insights. Discover new relationships and entities

GraphRAG Patterns

Vector & Hybrid Search
basic vector/full-text search
graph-enhanced
graph/metadata filtering

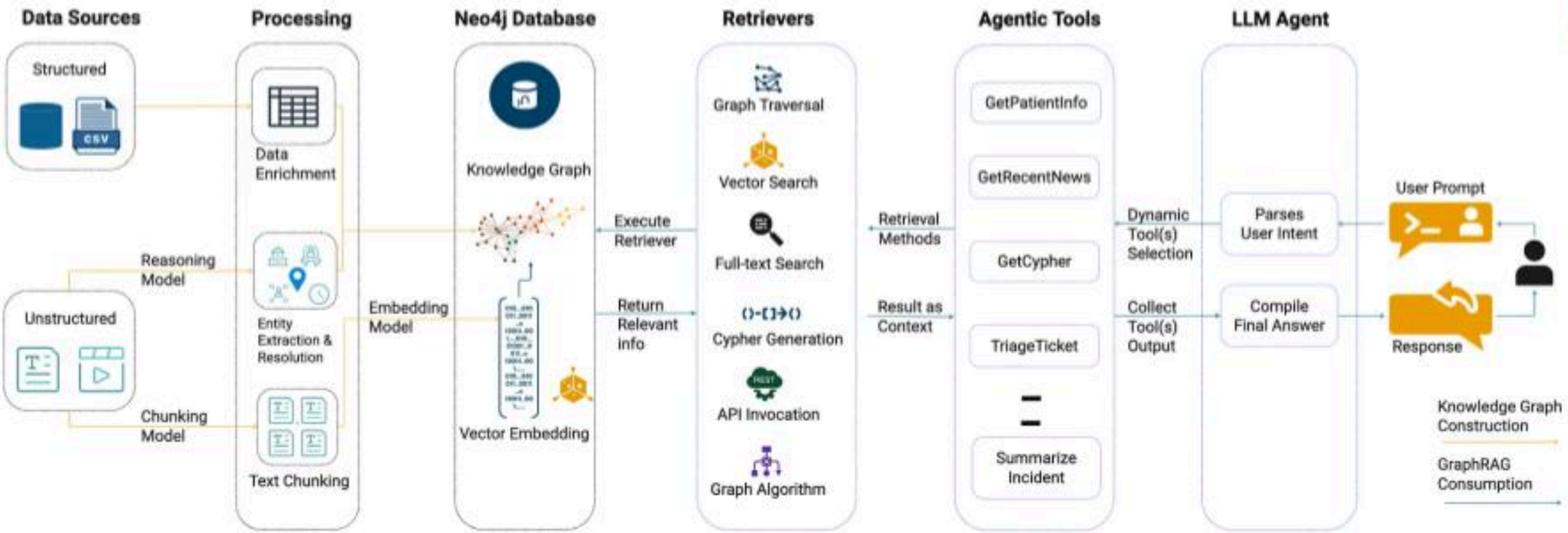
Query Templates (Deterministic)
Cypher templates
pattern matching

Query Generation
dynamic Cypher gen.
text2Cypher

Graph Enrichment
community summaries
graph embeddings
pagerank

Agent Brain
Memory
Reasoning & Planning
Decision & Execution

Agentic Tools



Neo4j Inc. All rights reserved 2025

Summary

In this lesson, you learned:

- **Traditional RAG helps** but introduces new challenges
- **Context ROT**: Poor retrieval can make responses *worse* than no retrieval
- **The limitation**: Traditional RAG treats documents as isolated blobs, missing relationships
- **Questions requiring relationships** can't be answered with similarity search alone
- **GraphRAG** extracts structure from documents, preserving entities and relationships

Next: See how this applies to SEC filings and the knowledge graph you'll explore.

The SEC Filings Knowledge Graph

The SEC Filings Example

Throughout this workshop, you'll work with a knowledge graph built from SEC 10-K filings.

These documents contain:

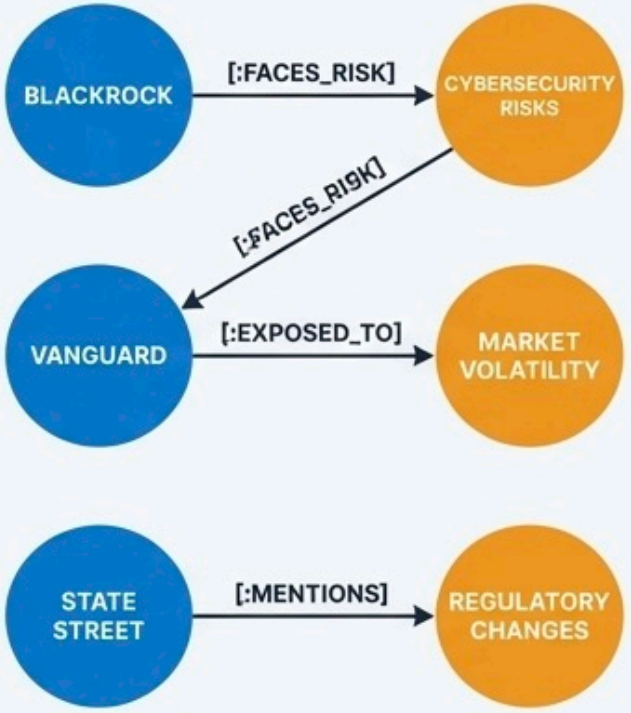
- Companies and their business descriptions
- Risk factors they face
- Financial metrics they report
- Products and services they mention
- Executives who lead them

The Core Insight: Structure is the Key to Intelligence

Unstructured Text (SEC Filings)

GraphRAG

Structured Knowledge Graph



Documents aren't just bags of words. They are containers for entities and the relationships between them. Traditional RAG throws away this inherent structure. **GraphRAG extracts, preserves, and leverages it.** By transforming unstructured text into a structured Knowledge Graph, we enable the AI to reason over a network of connected facts, not just a list of similar text snippets. This is the foundational shift that unlocks deeper, more accurate insights.

What the Graph Enables

Question Type	How the Graph Helps
"What risks does Apple face?"	Traverse FACES_RISK relationships
"Which companies mention AI?"	Find MENTIONS relationships to AI products
"Who owns Apple?"	Follow OWNS relationships from asset managers
"How many risk factors are there?"	Count RiskFactor nodes

The Pre-Built Knowledge Graph

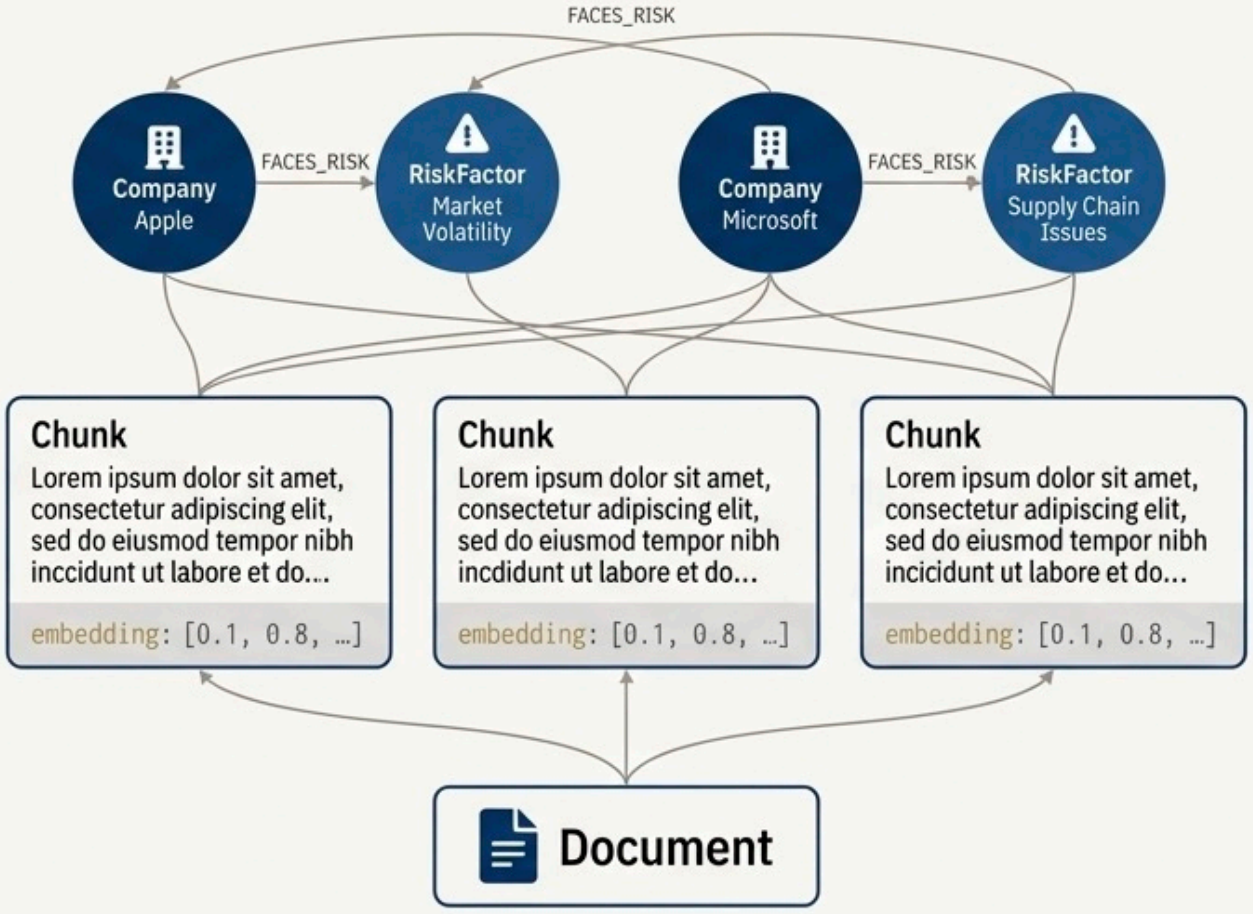
The graph you restored to your Aura instance has already been processed:

What was done (Lab 3 covers this in detail):

- SEC 10-K filing PDFs were ingested
- Documents were chunked into smaller pieces
- An LLM extracted entities and relationships
- Vector embeddings were generated for semantic search

You get the finished result ready for exploration with Aura Agents.

The Complete Knowledge Graph: Structure Meets Meaning



Your graph now contains:

- **Structured Entities & Relationships:** From schema-driven extraction.
- **Appropriately Sized Chunks:** A result of your chunking strategy.
- **Resolved Entities:** Canonical nodes with no duplicates.
- **Vector Embeddings:** Enabling semantic search on all text content.

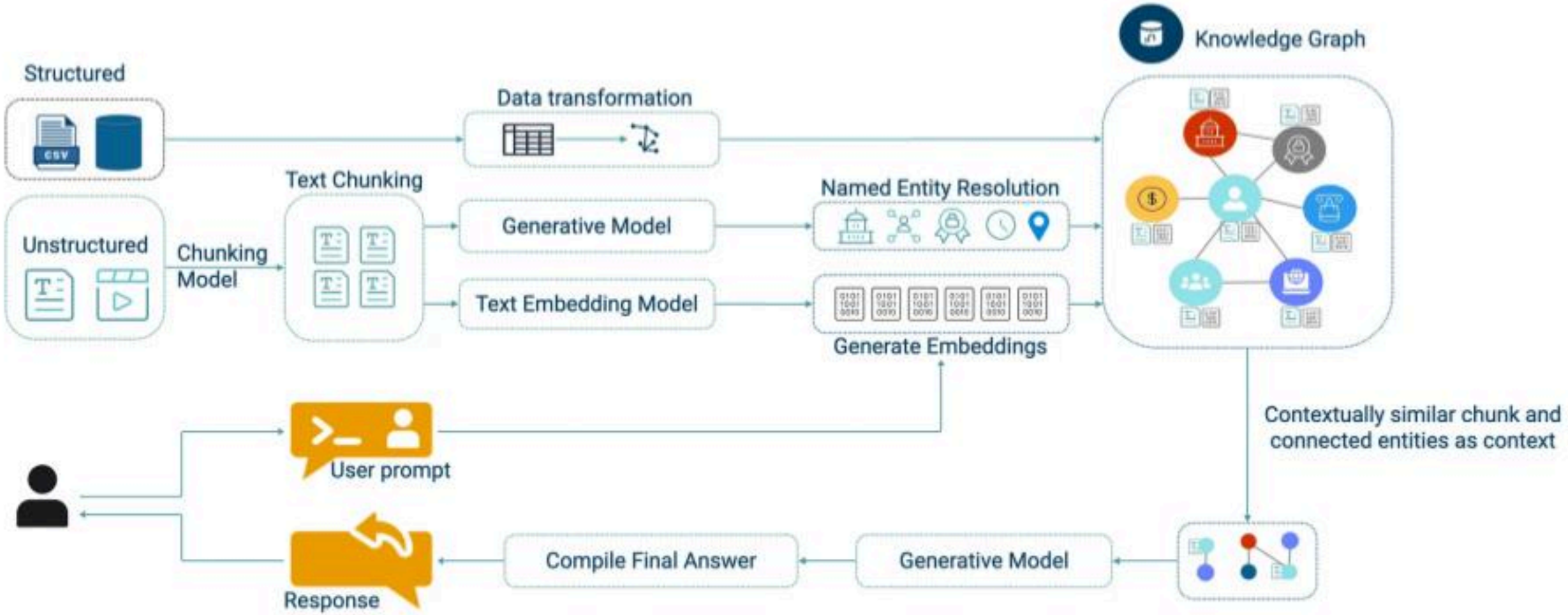
This unified structure enables sophisticated retrieval patterns that leverage both structured connections and semantic meaning.

The Processing Pipeline (Preview)

This is what Lab 3 covers in depth:

Step	What Happens
Document Ingestion	Read SEC 10-K filing PDFs
Chunking	Break into smaller pieces for processing
Entity Extraction	LLM identifies companies, products, risks
Relationship Extraction	LLM finds connections between entities
Graph Storage	Save entities and relationships to Neo4j
Vector Embeddings	Generate embeddings for semantic search

Building the Knowledge Graph



Neo4j Inc. All rights reserved 2025

Ready to Explore with Aura Agents

With the knowledge graph pre-built, you can now:

1. **Create an Aura Agent** connected to your graph
2. **Configure retrieval tools** (Cypher templates, similarity search, Text2Cypher)
3. **Ask complex questions** that combine graph traversal and semantic search
4. **See how GraphRAG works** before diving into the code

Next: Build your SEC Filings Analyst agent with Aura Agents.

Neo4j Aura Agents

Building GraphRAG Applications Without Code

What Are Aura Agents?

Aura Agents is a **no-code platform** for building AI agents grounded in your Neo4j graph data.

Key Capabilities:

- Build agents directly in the Aura Console
- Configure retrieval tools without writing code
- Test in an integrated chat playground
- Deploy to production via API

The Three-Tool Architecture

Aura Agents combine three complementary retrieval methods:

Tool	Purpose	Best For
Cypher Templates	Precise, parameterized queries	Known question patterns
Similarity Search	Semantic vector search	Finding relevant content
Text2Cypher	Natural language to Cypher	Ad-hoc exploration

The agent **automatically selects** the right tool for each question.

Tool 1: Cypher Templates

Pre-defined queries with parameters for common questions.

Example: Company Overview Tool

```
MATCH (c:Company {name: $company_name})
OPTIONAL MATCH (c)-[:FILED]->(d:Document)
OPTIONAL MATCH (c)-[:FACES_RISK]->(r:RiskFactor)
RETURN c.name AS company,
       collect(DISTINCT r.name)[0..10] AS risks
```

User asks: "Tell me about Apple's risks"

Agent: Extracts "APPLE INC", executes template, returns structured answer

Tool 2: Similarity Search

Semantic search using vector embeddings stored in Neo4j.

How It Works:

1. User question is embedded into a vector
2. Vector index finds similar content
3. Retrieval query enriches with graph context
4. Agent synthesizes the response

Example Question: "What do companies say about AI and machine learning?"

Returns semantically relevant passages with company context.

Tool 3: Text2Cypher

Converts natural language to Cypher queries dynamically.

How It Works:

1. User asks a question in plain English
2. LLM generates appropriate Cypher query
3. Query executes against the database
4. Results are returned to the agent

Example: "Which company has the most risk factors?"

```
MATCH (c:Company)-[:FACES_RISK]->(r:RiskFactor)
RETURN c.name, count(r) AS risk_count
ORDER BY risk_count DESC LIMIT 1
```

Why Graph Context Matters

Traditional RAG retrieves text chunks. GraphRAG retrieves **connected knowledge**.

Vector-Only RAG:

- "Find documents about cybersecurity risks"
- Returns: text chunks mentioning cybersecurity

GraphRAG with Aura Agents:

- "Which asset managers own companies facing cybersecurity risks?"
- Returns: traverses OWNS and FACES_RISK relationships

Graphs enable questions that pure vector search cannot answer.

Creating an Aura Agent

Building an agent takes minutes:

1. **Define the agent** - Name, description, system instructions
2. **Select your database** - Connect to your AuraDB instance
3. **Add tools** - Configure Cypher templates, similarity search, Text2Cypher
4. **Test in playground** - Interact with your agent
5. **Deploy** - Get an authenticated API endpoint

No coding required - just configuration.

Agent Configuration Example

Agent Name: SEC Filings Analyst

System Instructions:

```
You are an expert financial analyst specializing in
SEC 10-K filings analysis. You help users understand:
- Company risk factors and comparisons
- Asset manager ownership patterns
- Financial metrics and products
- Relationships between entities

Ground your responses in actual data from SEC filings.
```

Testing Your Agent

The Aura Console provides an integrated chat playground:

Cypher Template Test:

> "What risks do Apple and Microsoft share?"

- Agent selects `find_shared_risks` template
- Executes with company parameters
- Returns comparison results

Semantic Search Test:

> "What do filings say about supply chain?"

- Agent uses similarity search tool
- Finds relevant passages across companies

Tool Selection in Action

The agent reasons about which tool to use:

Question	Tool Selected	Why
"Tell me about NVIDIA"	Cypher Template	Matches known pattern
"Find content about climate"	Similarity Search	Semantic search needed
"How many products total?"	Text2Cypher	Aggregation query
"Compare Apple and Google risks"	Cypher Template	Two-company comparison

The agent explains its reasoning in each response.

Deployment Options

Internal Testing:

- Share with team members in Aura Console
- Test different configurations

Production API:

- Deploy to authenticated REST endpoint
- Client credentials from user profile
- Integrate into your applications

Coming Soon:

- MCP (Model Context Protocol) support
- Additional integration protocols

From No-Code to Code

Aura Agents demonstrates the same patterns you'll implement programmatically:

Aura Agent Tool	Python Implementation
Cypher Template	Parameterized queries
Similarity Search	VectorCypherRetriever
Text2Cypher	Text2CypherRetriever
Agent orchestration	Microsoft Agent Framework

Labs 5 and 6 implement these patterns in Python.

The Value of Aura Agents

For Prototyping:

- Validate GraphRAG approach quickly
- Test retrieval strategies without code
- Iterate on prompts and tools

For Production:

- Deploy agents in minutes
- Secure, authenticated endpoints
- Scalable infrastructure

Summary

Aura Agents provide:

- **No-code GraphRAG** – Build agents without programming
- **Three retrieval tools** – Cypher templates, similarity search, Text2Cypher
- **Intelligent orchestration** – Automatic tool selection
- **Easy deployment** – Test in playground, deploy to API

The fastest path from knowledge graph to AI agent.

Next Steps

1. **Create your Aura instance** (if not done already)
2. **Build the knowledge graph** (Labs 2-3)
3. **Create an Aura Agent** using the console
4. **Test with sample questions**
5. **Continue to Python implementation** (Labs 5-6)

Neo4j Graph Workload for Microsoft Fabric

Bringing Graph Analytics to Your OneLake Data

What is Microsoft Fabric?

Microsoft Fabric is a **unified analytics platform** that brings together all data and analytics tools in one place.

Key Components:

- **Data Engineering** - Build data pipelines and transformations
- **Data Warehousing** - SQL-based analytics at scale
- **Data Science** - Machine learning and AI workloads
- **Real-Time Analytics** - Streaming data processing
- **Power BI** - Business intelligence and visualization

Everything runs on a unified foundation.

What is OneLake?

OneLake is the **single, unified data lake** for all of Microsoft Fabric.

Key Characteristics:

- **One copy of data** - No data duplication across services
- **Open format** - Delta Lake / Parquet for interoperability
- **Unified governance** - Single security and compliance model
- **Automatic organization** - Data organized by workspace

Think of OneLake as the "OneDrive for data" - all your organizational data in one place.

What is a Fabric Workload?

A **Workload** is a specialized capability that extends Microsoft Fabric's functionality.

Native Workloads:

- Data Factory, Synapse, Power BI, etc.

Partner Workloads:

- Third-party tools integrated natively into Fabric
- Access OneLake data directly
- Use Fabric's security, identity, and compute
- Appear as first-class experiences in the Fabric portal

Partner workloads extend Fabric without leaving the platform.

How Workloads Use OneLake

Partner workloads integrate deeply with the Fabric ecosystem:

Integration Point	What It Provides
Entra ID	Seamless authentication and authorization
Workspaces	Object persistence and sharing
Lakehouses	Secure access to tabular data
Fabric Capacity	In-platform computing resources

Workloads operate on your data where it lives.

Introducing Neo4j Graph Workload

Neo4j provides a **native graph analytics workload** for Microsoft Fabric.

What It Enables:

- Transform tabular OneLake data into graph models
- Run graph algorithms and analytics
- Visualize and explore data relationships
- Query with Cypher directly in Fabric

Available now as a first-class Fabric experience.

Graph Intelligence Natively in Microsoft Fabric

Bring graph intelligence into Fabric workflows

- **Reveal hidden insights in Fabric OneLake data**: Use advanced graph algorithms to analyze connected relationships across your existing structured data.
- **Easily create graphs:** Transform relational data into graphs in a few clicks, with AI assist graph data modelling.
- **Simplified experience:** Import, query, visualize and explore your graph data natively in Microsoft Fabric.
- **Fully managed:** Scalable Fabric experience that grows with your analytics needs, without managing infrastructure.
- **Consumption pricing:** Monthly subscription, and the option to pay for running graph algorithms on demand.
- **Keep analytics in sync:** Schedule Spark jobs to seamlessly move data in and out of OneLake tables.

Testimonials

"We really like the seamless login experience all the way to the AuraDB database because it makes the whole Fabric experience effortless for our users."

– Preview Customer

"The generative AI assistance for creating the graph is a game changer for getting started if you are new to graphs."

– Preview Customer

Customers are "starting to see Fabric as their strategic data platform. This workload gives us the ability to offer our customers a graph solution that integrates into Fabric."

– Neo4j & Microsoft Partner

Fully Managed, Secure, Scalable Environment

Simplify operations with fully managed scaling and lifecycle management

Streamline procurement by running directly in Fabric and purchase through Azure Marketplace

Strengthen security with Azure enterprise controls and Neo4j's node-level access control

Why Graph Analytics on OneLake Data?

Your OneLake data contains **hidden relationships** that tabular analysis misses.

Analysis Type	Traditional BI	Graph Analytics
Customer segments	Static groupings	Community detection
Influence patterns	Not possible	Centrality algorithms
Connected risks	Manual joins	Path traversal
Recommendations	Rule-based	Similarity algorithms
Fraud detection	After the fact	Pattern matching

Graph analytics reveals the connections in your data.

Neo4j Graph Workload Capabilities

Transform & Load:

- Map tabular data from Lakehouses to graph models
- Create nodes and relationships from your data
- Load into a Neo4j AuraDB Professional database

Analyze:

- Full Cypher query language
- 65+ built-in graph algorithms
- Centrality, community detection, similarity, path finding

Visualize:

- Explore interface for interactive graph visualization
- Discover patterns visually

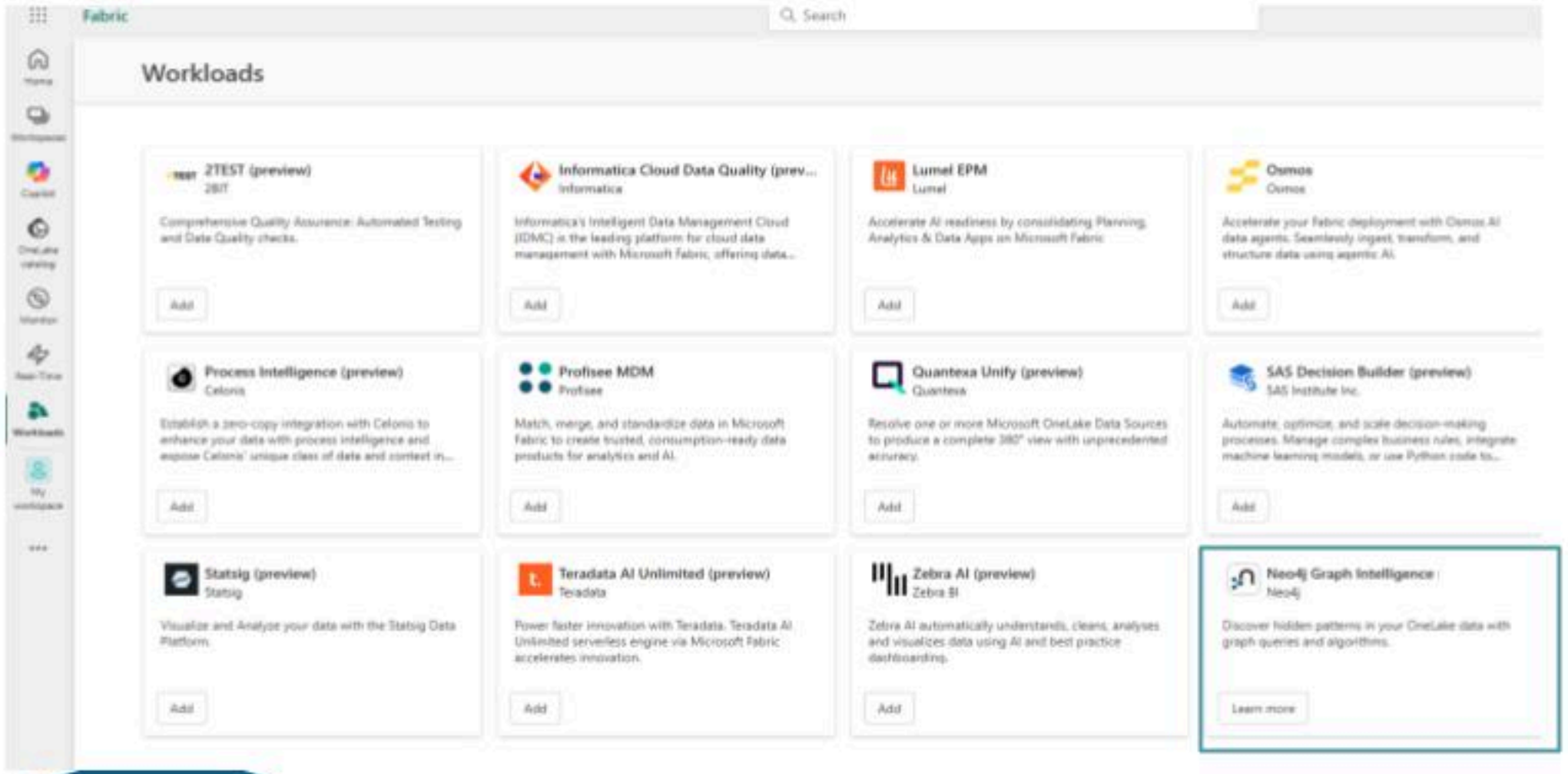
Available Graph Algorithms

The Neo4j Graph Workload includes the full algorithm library:

Category	Algorithms	Use Cases
Centrality	PageRank, Betweenness, Degree, Eigenvector	Find influential nodes
Community	Louvain, Label Propagation, WCC	Detect clusters and groups
Similarity	Node Similarity, KNN	Recommendations, deduplication
Path Finding	Dijkstra, A*, Shortest Path	Routing, dependencies
Link Prediction	Common Neighbors, Adamic Adar	Predict future connections
Embeddings	FastRP, Node2Vec, GraphSAGE	ML feature generation

Run any algorithm directly from the Fabric console.

Find Graph Intelligence in the Workload Hub



Neo4j Graph Intelligence

3 steps to finding Graph Powered Insights from Tabular Data

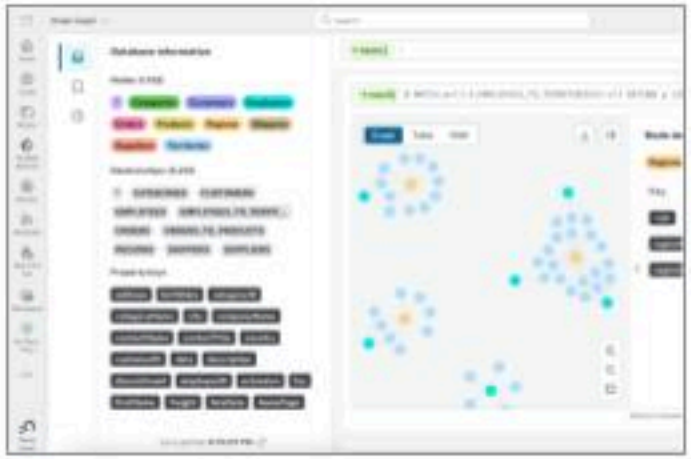
Select Lakehouse

Choose any OneLake lakehouse in the same region as Fabric Capacity.

Create Graph

Select the tables you want to transform into a graph model, Generative AI Assist will propose a data model that users can adjust before performing the import.

Analyze Graph



Business users and analysts can explore the graph and run graph algorithms on the data without using code.

Table to Graph Conversion

Transform relational data into nodes and relationships.

Graph Model Interface:

- Guides the data mapping process visually
- Preview graph structure before loading

AI-Assisted Modeling:

- AI Assistant suggests node and relationship types for review
- Generative AI uses schema analysis and Azure OpenAI for suggestions
- Accept, modify, or reject recommendations

Execution:

- A Spark job executes the transformation into graph objects

Example: Customer 360 Graph

Transform customer data into a connected view:

Source Tables:

- `customers` - Customer profiles
- `orders` - Purchase history
- `products` - Product catalog
- `interactions` - Support tickets, emails

Graph Model:

```
(Customer)-[:PURCHASED]->(Product)
(Customer)-[:CONTACTED]->(Support)
(Product)-[:SIMILAR_TO]->(Product)
```

Insights: Community detection reveals customer segments, PageRank identifies influential customers.

Explore: Visual Graph Analysis

The **Explore** interface provides interactive visualization:

Visual Capabilities:

- Interactive graph canvas
- Drag and arrange nodes
- Filter and highlight patterns
- Export visualizations

Search & Discovery:

- Pattern-based search
- Find connections between entities
- Expand neighborhoods

Built into Fabric - no external tools needed.

Neo4j Graph Intelligence

3 ways to analyze the graph

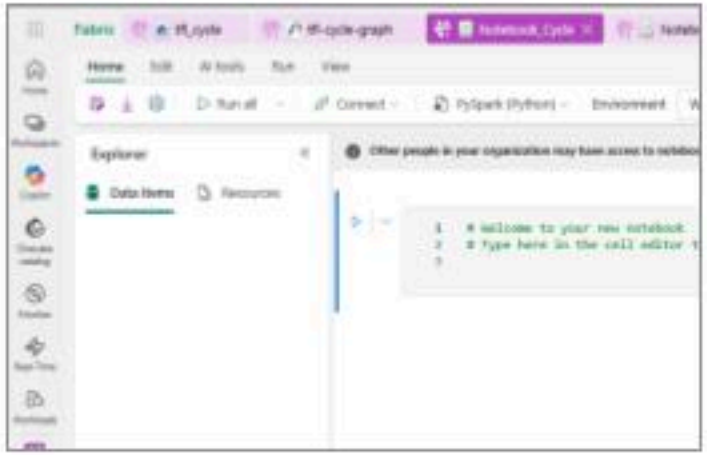
Explore

No coding required - business users and analysts can point & click to explore the graph and run graph algorithms.

Query

AI-powered Cypher queries - no longer just for power users who know Cypher, data analysts can use text to Cypher to write GQL Cypher and run algorithms.

Notebooks



Fabric Notebooks - data scientists who prefer Python can run algorithms from their Notebooks, create spark jobs and write results back to OneLake.

Integration with Fabric Security

The Neo4j Graph Workload inherits Fabric's enterprise security:

Authentication:

- Microsoft Entra ID (Azure AD)
- Single sign-on across Fabric

Authorization:

- Workspace-level permissions
- Role-based access control

Data Governance:

- Data stays in your tenant
- Audit logging

Use Cases for Graph Analytics on OneLake

Supply Chain:

- Map supplier relationships
- Identify bottlenecks and risks
- Optimize logistics paths

Financial Services:

- Detect fraud patterns
- Analyze transaction networks
- Risk propagation analysis

Manufacturing:

- Bill of materials graphs
- Quality issue tracing
- Asset relationship mapping

Current Capabilities & Roadmap

Available Now:

- Graph creation from Lakehouse tables
- Full Cypher query support
- 65+ graph algorithms
- Explore visualization

Coming Soon:

- Writeback of insights to OneLake
- Additional data source support

Summary

Neo4j Graph Workload for Microsoft Fabric:

- **Native Integration** - First-class workload in the Fabric portal
- **Zero ETL** - Transform OneLake data directly to graphs
- **Full Analytics** - 65+ algorithms, Cypher queries, visualization
- **Enterprise Ready** - Inherits Fabric security and governance
- **One-Click Setup** - Install from the Workload Hub

Unlock the relationships hidden in your OneLake data.

Learn More

Resources:

- Neo4j Graph Analytics for Microsoft Fabric
- Neo4j Fabric Workload Documentation
- Announcing the Neo4j Graph Workload

Next Steps:

1. Install the Neo4j workload in your Fabric workspace
2. Connect to a Lakehouse with your data
3. Create your first graph model
4. Run algorithms and explore connections