

Lab 7: Graph Augmentation Agent

AI-Driven Schema Evolution

Concepts Introduced

Graph Augmentation - Using AI to discover and suggest improvements to graph schemas based on document analysis.

Key Concepts:

- **Schema Evolution** - Iteratively improving graph structure based on new insights
- **Entity Extraction** - Identifying new node types from unstructured text
- **Relationship Discovery** - Finding implicit connections in documents
- **DSPy** - Framework for programming language models with signatures

Why DSPy?

- Declarative approach: define *what* you want, not *how* to prompt
- Native structured output via Pydantic models
- Works reliably with Databricks Multi-Agent Supervisor endpoints

The Goal

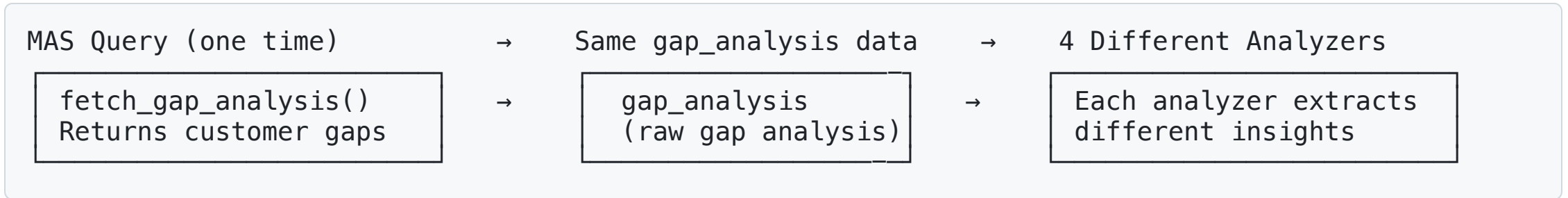
Build a **DSPy Agent** that uses the Multi-Agent Supervisor to **analyze documents** and **suggest graph enrichments**.



Why this architecture?

- DSPy provides type-safe structured output from LLM calls
- Multi-Agent Supervisor routes queries to specialized agents (Genie + Knowledge)

Data Flow: Two-Stage Analysis



Stage 1: MAS Query (Direct API Call)

- Sends comprehensive prompt to Multi-Agent Supervisor
- MAS coordinates Genie (structured) + Knowledge Agent (documents)
- Returns **raw text** analysis comparing both sources

Stage 2: DSPy Analyzers (Structured Extraction)

- Same text passed to 4 different analyzers
- Each extracts different insights into **typed Pydantic models**
- Investment Themes, New Entities, Missing Attributes, Implied Relationships

DSPy: Signatures as Function Contracts

DSPy treats LLM calls like function definitions. You declare the **input** and **output** types, and DSPy figures out how to prompt the model.

```
class ImpliedRelationshipsSignature(dspy.Signature):  
    """Find relationships implied in documents but not captured in the graph."""  
  
    document_context: str = dspy.InputField(  
        desc="Documents containing information about entity relationships"  
    )  
  
    analysis: ImpliedRelationshipsAnalysis = dspy.OutputField(  
        desc="Structured suggestions for new relationship types"  
    )
```

What DSPy handles for you:

- Prompt generation from signature docstrings and field descriptions
- Output parsing into Pydantic models (type-safe structured output)
- Automatic prompt optimization when you run DSPy's optimizers

DSPy Implementation

Why DSPy? The key advantage is **structured output** via Pydantic models.

```
import dspy
from pydantic import BaseModel

class InvestmentTheme(BaseModel):
    theme_name: str
    confidence: float
    evidence: list[str]

class ThemeSignature(dspy.Signature):
    """Extract investment themes from documents."""
    document: str = dspy.InputField()
    themes: list[InvestmentTheme] = dspy.OutputField()

# DSPy handles prompt generation + JSON parsing automatically
extract = dspy.Predict(ThemeSignature)
result = extract(document="...") # Returns typed Pydantic objects!
```

Result: Provides reliable structured output from Multi-Agent Supervisor - no manual JSON parsing.

What the Lab Creates

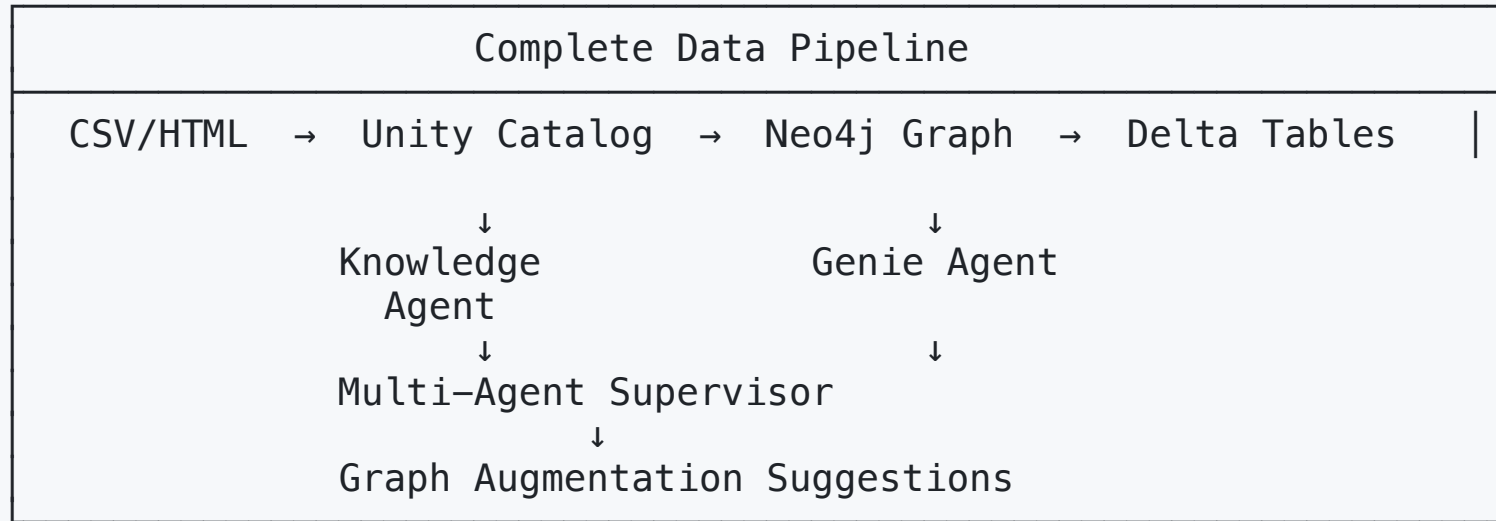
The DSPy Agent performs **four types of analysis** on your documents:

Analysis Type	What It Finds
Investment Themes	Emerging trends in market research
New Entities	Node types missing from schema
Missing Attributes	Properties mentioned but not captured
Implied Relationships	Connections in documents but not in graph

Example: Customer profiles mention "ESG investing interest" but no ESG relationship exists in the graph.

Each analysis returns **typed Pydantic objects** with structured suggestions for graph improvements.

What You've Built



The cycle: Documents inform graph → Graph serves agents → Agents suggest improvements