An Agentic System for Academic Papers: Construction of a GraphDB

1. Overview

This project implements an agentic system that builds a knowledge graph over Gaussian Splatting research papers.

Agents:

- Read research PDFs
- Extract key entities (concepts, methods, datasets, metrics, authors, papers)
- Connect entities with semantic relationships (e.g., introduces, improves_on, evaluates_on)

The backend is a Python proof-of-concept that:

- Ingests a corpus of Gaussian Splatting papers
- Uses an LLM agent to extract structured data
- Stores a typed property graph in Postgres (via Supabase)
- Provides example SQL queries for research-style questions

The focus is on a clean, extensible backend and a graph schema suitable for future UI or analysis.

2. Corpus and Scope

I focused on a small but representative subset of the Gaussian Splatting literature

The system processes 49 Gaussian Splatting papers, starting with the seminal 3D Gaussian Splatting for Real-Time Radiance Field Rendering (2023) and its citation network. Papers were selected based on citation strength and keyword relevance ("Gaussian Splatting" + related terms).

This corpus validates entity extraction, relationship mapping, and graph querying at a realistic scale while remaining computationally manageable for the proof of concept.

My corpus selection strategy was mainly based on the documents that were cited for the main document that I started with: [3D Gaussian Splatting for Real-Time Radiance Field Rendering](). I looked up the over 100 + documents that had cited this main document and selected 49 documents from them that hadd the main keyword “Gaussian Splatting” either in the title, introduction or in the abstract.

3. System Architecture

End-to-end flow:

PDF ingestion: PDFs in data/raw/ are enumerated by src/pipeline/test_pipeline.py.

Text extraction (PDFParser): PDFParser(pdf_path) extract_text () uses a PDF library to extract plain text from each paper.

An Agentic System for Academic Papers: Construction of a GraphDB

LLM extraction agent (ExtractionAgent)

- extract_metadata(text) → title, authors, year, abstract
- extract_entities(text, title) → concepts, methods, datasets, metrics
- extract_relationships(text, title, entities) → typed relationships with evidence + confidence

Graph persistence layer (GraphDatabase, graph_db.py)

- insert_paper(metadata):
- creates a Paper node in nodes
- upserts papers row
- creates Author nodes, authors, and paper_authors rows
- creates authored_by edges

Entity insertion

- get_or_create_node("Concept" | "Method" | "Dataset" | "Metric", {"name": ...})

Relationship insertion

- string → node_id resolution (name_to_id)
- create_edge(type, source_id, target_id, properties={evidence}, confidence)

4. Graph Representation in Postgres

Core design

The graph uses a typed property graph model in Postgres:

- Nodes and edges have explicit types
- Properties are stored as JSONB for flexibility
- Standard relational tables coexist for legacy queries

Node model

It has node types: Paper, Author, Concept, Method, Dataset, Metric.

Node schema: nodes(id, node_type_id, properties jsonb)

For example, Paper: {"title": "3D Gaussian Splatting...", "year": 2023, "abstract": "..."}
Method: {"name": "3D Gaussian Splatting"}

Edge model

It has edge types: authored_by, introduces, improves_on, evaluates_on, measures_with, extends, compares_with

Edge schema: edges(id, edge_type_id, source_node_id, target_node_id, properties jsonb, confidence float)

An Agentic System for Academic Papers: Construction of a GraphDB

Example:

- Type: introduces
- Source: Paper "4D Gaussian Splatting..."
- Target: Concept "4D Gaussian Splatting"
- Properties: {"evidence": "We propose 4D-GS as..."}
- Confidence: 0.95

Duplicates prevented via UNIQUE(edge_type_id, source_node_id, target_node_id)

Legacy Tables

For traditional SQL queries:

- **papers(id, node_id, title, year, abstract)**
- **authors(id, node_id, name)**
- **paper_authors(paper_id, author_id, author_order)**

Entity & Relationship Extraction

Prompting Strategy

The agent uses OpenAI gpt-4o-mini with structured prompts and response_format={"type": "json_object"} to ensure consistent output.

Metadata extraction:

- title: Full paper title
- authors: List of author names
- year: Publication year (integer)
- abstract: Full abstract text

Entity extraction:

extract_entities(paper_text, paper_title) extracts four entity types:

- Concepts: Key ideas or theoretical contributions
- Methods: Algorithms or technical approaches
- Datasets: Evaluation datasets
- Metrics: Performance measurements

This returns a structured JSON.

Relationship extraction

extract_relationships(paper_text, paper_title, entities) identifies semantic connections:

- introduces: Paper introduces a concept or method
- improves_on: Method improves upon another
- evaluates_on: Paper evaluates on a dataset
- measures_with: Paper uses a metric
- extends: Work extends previous research
- compares_with: Compares against another method

Each relationship includes:

- source: Paper title or entity name
- target: Entity from extracted list
- confidence: 0.0-1.0 score from LLM
- evidence: 1-2 sentence justification from paper text

Validation and error handling

Duplicate prevention:

- get_or_create_node checks for existing entities by name/title before creating
- ON CONFLICT in edge insertion prevents duplicate relationships

Error handling:

- Failed JSON parsing returns empty structures rather than crashing
- Relationships with unmapped source/target are skipped
- PDF parsing errors are logged, allowing pipeline to continue

Use Cases and User Experience

Real World Cases

- Semantic literature mapping: Explore how Gaussian Splatting techniques relate to and extend NeRF variants through conceptual relationships beyond citations.
- Method-centric exploration: Search for a method (e.g., "3D Gaussian Splatting") to discover papers that introduce or improve it, datasets and metrics used for evaluation, and evidence snippets explaining each relationship.
- Research comparison: Compare papers by identifying overlapping concepts, extension relationships, and different evaluation approaches.
- Novelty discovery: Track emerging concepts by monitoring which ideas are being introduced in recent papers and their adoption patterns.

Explainable insights

Each relationship stores evidence (text snippet) and confidence score, enabling transparent, verifiable connections:

Example: "4D Gaussian Splatting improves on 3D Gaussian Splatting by introducing temporal coherence. Evidence: 'Our method achieves 5× faster training...' (confidence: 0.95)"

Researchers can verify extraction accuracy, understand relationship basis, and filter by confidence thresholds.

Future UI

Interactive graph visualization with filters (relationship type, confidence, year range) and evidence-backed search interface.

Scalability and Maintenance

Current State: The POC uses a synchronous pipeline processing 50 papers in 15-20 minutes, sufficient for validation but not production-scale.

Scaling Strategy

- Architecture: Migrate to queue-based system (Celery/RQ) with one job per paper, enabling 10-50 concurrent workers with rate-limited LLM calls
- Optimization: Cache LLM responses, batch database writes, separate PDF storage (S3/GCS) from compute
- Staying current: Monitor arXiv RSS feeds and citation graphs; trigger incremental ingestion with idempotent design (get_or_create + ON CONFLICT)
- Quality assurance: Track confidence scores over time; alert on low-confidence batches for manual review

Performance & Consistency

- Database: JSONB + GIN indexes on frequently queried properties; UNIQUE constraints prevent duplicates
- Transactions: Context-managed DB access with atomic commits/rollbacks
- Fault tolerance: Failed papers logged without breaking pipeline; retry logic for transient errors

An Agentic System for Academic Papers: Construction of a GraphDB

This design scales to 1,000+ papers while maintaining extraction quality and data consistency.

Limitations and Trade-offs

Extraction Quality

- LLM noise: occasional missed entities and hallucinated relationships
- No ground truth validation or citation database cross-referencing
- Context window limited to first ~5-8K characters per paper

Entity Normalization

- No synonym handling ("3DGS" vs "3D Gaussian Splatting" treated as distinct)
- No controlled vocabulary, fuzzy matching, or canonical IDs

System Design

- No extraction provenance (can't trace which LLM version/prompt produced results)
- Fixed node/edge types, not dynamically extensible
- Single Postgres instance without sharding or replication
- Synchronous pipeline not optimized for 1000+ paper batches
- No production monitoring, metrics, or alerting

Future Roadmap

Improved extraction

- Add few-shot examples to prompts with curated Gaussian Splatting papers
- Fine-tune smaller model on domain-specific labeled data
- Implement entity normalization (synonym mapping, lowercasing, acronym expansion)

Scalable ingestion

- Migrate to job queue (Celery + Redis or cloud-native)
- Add parallel processing for 100+ concurrent papers
- Cache LLM responses to reduce API costs

User interface

- Minimal web app: Search for papers or concepts, have a interactive graph visualization, and display evidence snippet display
- Filters: relationship type, confidence threshold, year range
- Export: subgraph as JSON/GraphML

Advanced features

- Semantic search: Add pgvector for embedding-based paper similarity
- Trend analysis: Track concept/method frequency over time

An Agentic System for Academic Papers: Construction of a GraphDB

- Recommendation: "Papers similar to X that introduce Y"

Example Queries and Results

Query1: Which papers improve on 3D Gaussian Splatting?

```sql query
SELECT DISTINCT p.properties->>'title' AS paper_title,
       e.confidence,
       e.properties->>'evidence' AS evidence
FROM nodes m
JOIN edges e ON e.target_node_id = m.id
JOIN edge_types et ON et.id = e.edge_type_id
JOIN nodes p ON p.id = e.source_node_id
WHERE m.properties->>'name' ILIKE '%3D Gaussian Splatting%'
  AND et.type_name = 'improves_on'
ORDER BY e.confidence DESC
LIMIT 5;
```

Result:
HUGS: Human Gaussian Splats (0.95 confidence): "achieves state-of-the-art rendering quality with 60 FPS while being ~100× faster to train"
[Evidence 2]: "Novel Dual-Domain Deformation Model explicitly models attribute deformations" (0.95)
[Evidence 3]: "Achieves 5× faster training and rendering speed compared with per-frame 3DGS" (0.95)
[Evidence 4]: "VastGaussian achieves higher quality and much faster rendering than SOTA" (0.95)
[Evidence 5]: "3D Gaussian splatting for 3D head avatar modeling with low computational consumption" (0.95)

Query2: What are the most commonly used evaluation datasets?

```sql query
SELECT d.properties->>'name' AS dataset,
       COUNT(*) as usage_count
FROM nodes d
JOIN node_types nt ON nt.id = d.node_type_id
JOIN edges e ON e.target_node_id = d.id
```

~~~
JOIN edge_types et ON et.id = e.edge_type_id
WHERE nt.type_name = 'Dataset'
  AND et.type_name = 'evaluates_on'
GROUP BY d.properties->>'name'
ORDER BY usage_count DESC
LIMIT 10;
~~~

Result:

Dataset	Usage Count
Mip-NeRF 360	2
Synthetic datasets	2
ACID	1
CO3D	1
DTU	1
KITTI	1
HyperNeRF Dataset	1
Blender	1

Query3: What are the most frequently used performance metrics?

~~~
```sql query
SELECT m.properties->>'name' AS metric,
      COUNT(*) as usage_count
FROM nodes m
JOIN node_types nt ON nt.id = m.node_type_id
JOIN edges e ON e.target_node_id = m.id
JOIN edge_types et ON et.id = e.edge_type_id
WHERE nt.type_name = 'Metric'
  AND et.type_name = 'measures_with'
GROUP BY m.properties->>'name'
ORDER BY usage_count DESC
LIMIT 10;
~~~

Result:

Metric	Usage Count
PSNR	8
Rendering speed	4
Rendering quality	3
Training speed	2
Accuracy	2
FPS	2
Rendering speed (FPS)	2
Compression Rate	1
Editing precision	1

Query4: How are papers distributed by year? (Research trend analysis)

```sql query
SELECT year, COUNT(*) as paper_count
FROM papers
WHERE year IS NOT NULL
GROUP BY year
ORDER BY year DESC;
```

Result:

Year	Paper Count
2025	2
2024	43
2023	2
2022	1

An Agentic System for Academic Papers: Construction of a GraphDB

2021	1

Query5: What are the most frequently introduced concepts across all papers?

``sql query
```
SELECT c.properties->>'name' AS concept,
    COUNT(DISTINCT p.id) as paper_count
FROM nodes c
JOIN node_types nt ON nt.id = c.node_type_id
JOIN edges e ON e.target_node_id = c.id
JOIN edge_types et ON et.id = e.edge_type_id
JOIN nodes p ON p.id = e.source_node_id
WHERE nt.type_name = 'Concept'
  AND et.type_name = 'introduces'
GROUP BY c.properties->>'name'
HAVING COUNT(DISTINCT p.id) > 1
ORDER BY paper_count DESC
LIMIT 10;
```

Result: No rows returned (most concepts are unique to individual papers, reflecting the novelty-focused nature of the field)

Final Stats: 49 papers, 280 authors, 803 nodes, 524 edges

Note: Implementation Language Choice

Selected: Python

The assignment prefers TypeScript for the agent layer. I chose Python for:

1. Rapid prototyping: Rich ecosystem (PyPDF2, OpenAI SDK, psycopg2)
2. ML/data pipeline experience: My background in Python data engineering, prototyping, and writing clean code.
3. Language-agnostic design: Agent prompts use JSON schemas portable to any language