Assignment 2

Shreya Sakura Noskor - 1007996563
Ben Marlow - 1006724920
Timothy Pasaribu - 1009714864
Huayin Luo - 1007864517

Part 1: Aspirational Datasets

Our application's core functionality depends on identifying similarities and connections between scientific papers and displaying them to the user. In an ideal world, we would have a series of datasets that could somehow optimally represent these connections depending on what a user requires. Note that a lot of our planning has been around ArXiv, a large open-source repository of academic papers from all fields. Our aspirational datasets are envisioned in this context, with schemas including "arxiv_ids" which can be used to find papers on the ArXiv. Below are a series of datasets with the data schemas we would like:

1.1 Consensus and debate (agreement/disagreement) dataset

One of the edges we want to include between nodes (papers) in our graph is whether two papers agree or disagree and whether the topics covered in papers generally agree with consensus or are still hotly debated. Given a primary paper, this dataset would be able to answer whether other papers support, contradict, or are neutral with respect to the primary paper. Additionally, it would be able to specifically represent the point of disagreement, and how much disagreement quantitatively exists on the subject. Based on all of this, it would also be able to assign a consensus status of either being settled, emerging, disputed, or debunked. This dataset would give us sufficient information to directly answer these questions and build the graph with these edges. Below is a hypothetical schema for this dataset:

Field Name	Type	Description
primary_arxiv_id	STRING	The paper under analysis.
related_arxiv_id	STRING	The paper being referenced or compared.
relationship_type	ENUM	SUPPORT, CONTRADICT, or NEUTRAL
claim_delta	TEXT	Specific point of disagreement
debate_intensity	FLOAT	A scale of 0 (consensus) to 1 (hotly

		contested) based on the volume of contradictory citations.
consensus_status	STRING	SETTLED, EMERGING, DISPUTED, or DEBUNKED

1.2 Concept evolution dataset

Another use case for our application is learning about new concepts and terminology, and what has been published on the subject in the past versus what is being published currently. The concept evolution dataset would provide us with a chronological history of the evolution of concepts in the literature, and what reading would be required to properly understand where the topic is now. This dataset would provide us with sufficient information to display an evolution path (an attribute in this schema), which a user would interpret as a reading list to understand the background and current research on a topic.

Field Name	Type	Description
concept_name	STRING	The specific idea
origin_paper_id	STRING	Paper where the concept was first popularized
evolution_path	VARIANT	A tree structure showing "child concepts" that have spawned from the original concept
sentiment_score	FLOAT	Community consensus (positive/advancement vs. negative/debunked).

1.3 The knowledge bridge/semantic proximity dataset

Another proposed functionality of our application is an interactive user experience, in which a map is continuously built over time to represent subjects that are well understood, and where the gaps may lie. For example, a user might build a map centered on nuclear physics. Over time, they would add papers that they have read to their map,

and the map would identify which subtopics in the subject have not been explored, suggesting papers to fill those gaps. The following dataset would help us implement this use case, taking a target paper and comparing it with a related paper. This comparison would establish the relationship between them (pre-requisite, analogy, future-reading, or **gap-filler**), their semantic differences and similarities, and their relative complexity to one another. Below is the schema for this proposed dataset:

Field Name	Type	Description
target_arxiv_id	STRING	The "destination" paper the user wants to understand.
related_arxiv_id	STRING	The suggested "bridge" paper.
relationship_role	ENUM	PREREQUISITE (fundamental), ANALOGY (simpler domain), FUTURE_READING (more complex) or GAP_FILLER.
semantic_distance	FLOAT	A 0–1 score where 0 is identical and 1 is a different field entirely.
concept_overlap	VARIANT	List of shared concepts and, crucially, **missing concepts** required to bridge the two.
complexity_level	INT	A score (1–5) representing the difficulty of the paper.

Part 2: Reality Check

The datasets that we provided in Part 1 are unfortunately not feasible to obtain, so we need to settle for something more realistic. Below are a series of datasets that we can obtain and use to achieve similar functionalities to what was described above:

Relevant Item	Description	Commentary
ArXiv Open Access Dataset (Accessible via Python API)	2.9M+ scholarly articles including full-text PDFs	Provides the raw text for the Bronze layer. Essential for

source	metadata and abstracts.	extracting conclusions and initial citation strings via PDF parsing. Foundation of our data.
Semantic Scholar API source	A database of 200M+ academic papers, providing AI-generated summaries, topic analysis, and structured citation lists.	Crucial for retrieving structured citations to help automate the agreement/contradiction logic and understand connections between papers.
OpenAlex source	Open scholarly knowledge graph containing papers, authors, institutions, venues, and citation relationships across hundreds of millions of works.	Provides large-scale structured metadata beyond arXiv. Useful for enriching the Silver layer with author networks, topic hierarchies, and citation edges that power the mindmap structure.
OpenCitations source	Open database of citation relationships between scholarly publications, represented as a graph of referencing and referenced works.	Enables construction of explicit paper-to-paper edges without relying solely on PDF parsing. Supports the agreement/contradiction logic and improves recommendation quality.
S2ORC (Semantic Scholar Open Research Corpus) source	Large dataset containing parsed full text, metadata, and citation information for millions of research papers.	Allows future extension beyond abstracts to section-level embeddings and richer semantic representations. Useful for scaling embedding pipelines in the Compute layer.
CORE Dataset source	Aggregated open access research papers collected from repositories worldwide, including metadata and sometimes full text.	Expands coverage beyond arXiv to improve recall in search and recommendation tasks. Helps evaluate ingestion pipeline scalability.
SciDocs Benchmark source	Benchmark dataset designed to evaluate scientific document embeddings across multiple	Provides an evaluation framework for the related-paper retrieval feature. Useful for validating embedding quality in

	tasks such as citation prediction and classification.	the Gold layer.

Part 3 - Data-processing pipelines

Our pipeline follows the Medallion Architecture, implemented via Modal workers and Snowflake.

3.1 Data Schemas (Snowflake)

- Bronze Layer (Raw): Stores the raw response from the ArXiv/PubMed APIs.
- Silver Layer (Cleaned): Contains stripped LaTeX, cleaned figure metadata, and structured abstracts.
- Gold Layer (Insights): Stores the final "MindMap" edges with the paper relationships (edges: CITES, SIMILAR, etc.)

```
-- Initial Setup
DROP TABLE IF EXISTS MINDMAP_DB.PUBLIC.GOLD_PAPER_RELATIONSHIPS;
DROP TABLE IF EXISTS MINDMAP_DB.PUBLIC.SILVER_PAPERS;
DROP TABLE IF EXISTS MINDMAP_DB.PUBLIC.BRONZE_PAPERS;

CREATE DATABASE IF NOT EXISTS MINDMAP_DB;
CREATE WAREHOUSE IF NOT EXISTS MINDMAP_WH WAREHOUSE_SIZE = 'XSMALL';

USE DATABASE MINDMAP_DB;
USE SCHEMA PUBLIC;

-- BRONZE LAYER: Raw Data Lake (Stores unedited JSON)
CREATE TABLE IF NOT EXISTS MINDMAP_DB.PUBLIC.BRONZE_PAPERS (
  ingestion_id UUID DEFAULT UUID_STRING(),
  raw_payload VARIANT, -- Native JSON storage
  ingested_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()
);

-- SILVER LAYER: Transformed data
CREATE TABLE IF NOT EXISTS MINDMAP_DB.PUBLIC.SILVER_PAPERS (
  id INT IDENTITY(1,1) PRIMARY KEY, -- Starts at 1, increments by 1
```

```
  arxiv_id STRING UNIQUE,
  ss_id STRING UNIQUE,
  title STRING,
  abstract STRING,
  conclusion TEXT,
  reference_list VARIANT,
  citation_list VARIANT,
  embedding VECTOR(FLOAT, 384),
  similar_embeddings_ids VARIANT -- List of paper IDs with similar embeddings (optional, can be populated later)
);

-- GOLD LAYER: Paper Relationships
CREATE TABLE IF NOT EXISTS MINDMAP_DB.PUBLIC.GOLD_PAPER_RELATIONSHIPS (
  source_paper_id INT,
  target_paper_id INT,
  relationship_type VARCHAR(50), -- 'CITES', 'SIMILAR'
  strength FLOAT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
  FOREIGN KEY (source_paper_id) REFERENCES MINDMAP_DB.PUBLIC.SILVER_PAPERS(id),
  FOREIGN KEY (target_paper_id) REFERENCES MINDMAP_DB.PUBLIC.SILVER_PAPERS(id),
  PRIMARY KEY (source_paper_id, target_paper_id, relationship_type)
);
```

Figure: Our SQL Schema for the 3 databases

3.2 Pipeline diagram

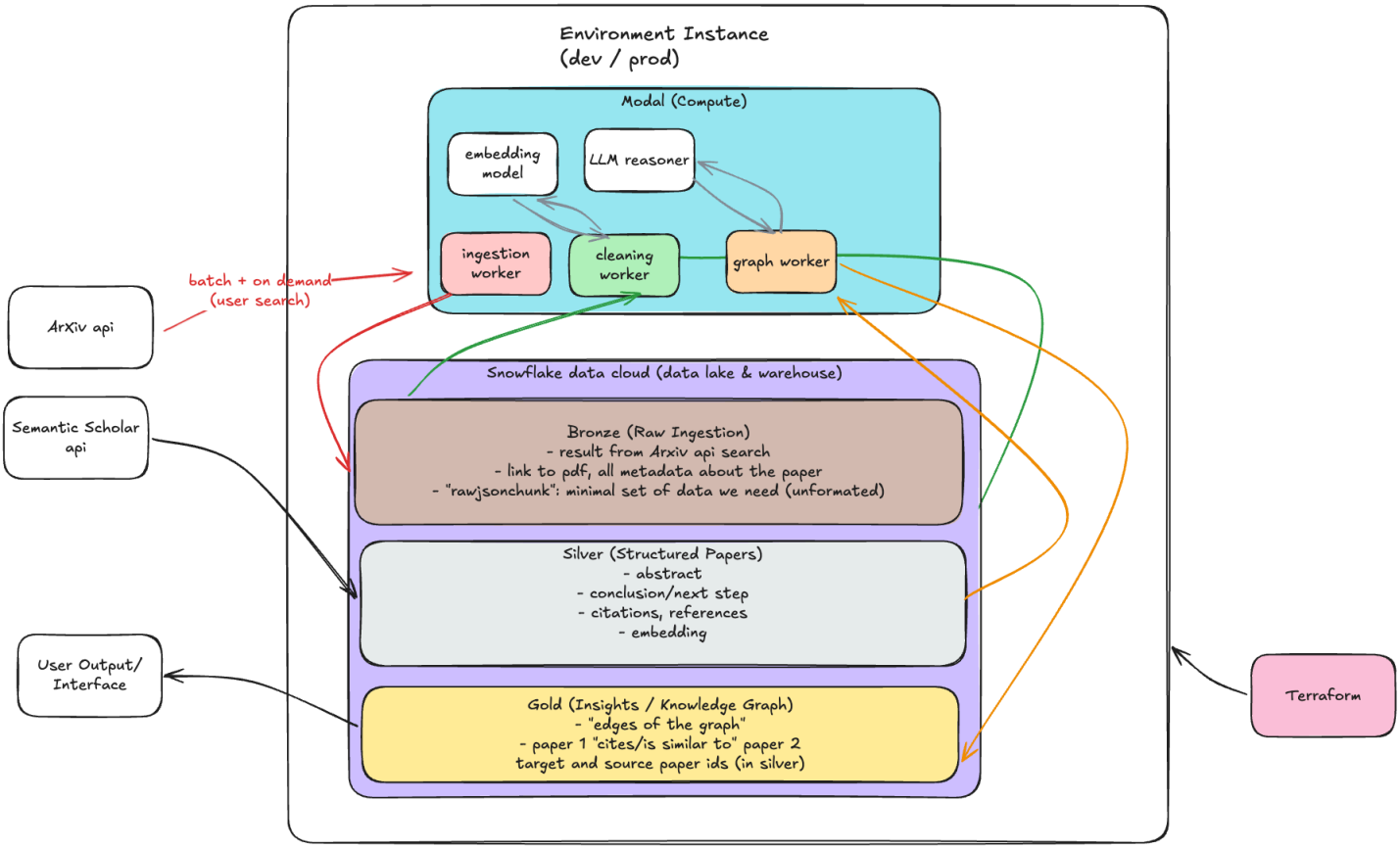

***Figure 1**: Overall pipeline diagram*

- Data ingestion: through arxiv api, filter for subset of papers (max results)
- Cleaning & transformation: extract title, metadata (citations - which papers it references), abstract , next steps section (scrape for conclusion/discussion) → embeddings
- Graph creation: use the citations and embeddings from cleaning/transformation to populate gold layer with edges

3.3 When the pipelines will run and for which use case:

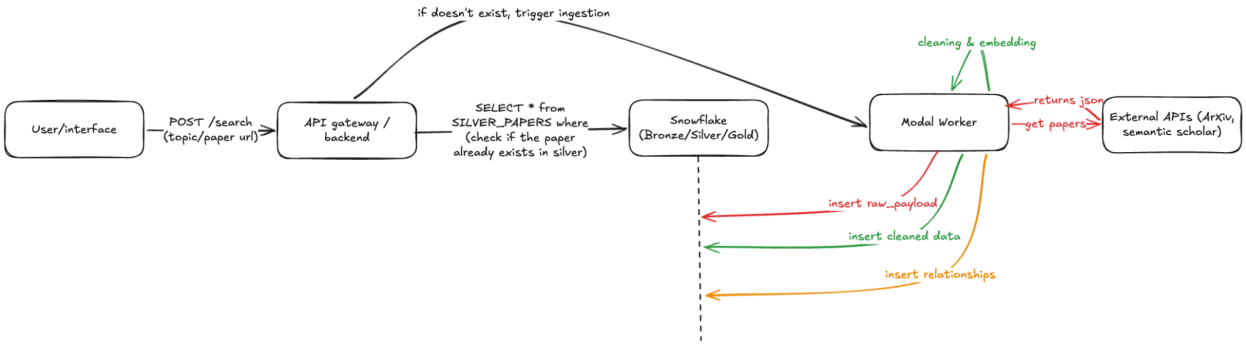

***Figure 2**: Sequence diagram for use case 1. Note, for use case 1 Modal Worker gets the specific paper by url. For use cases 2 and 3, it searches by keyword and fetches the top k results, where k is a parameter we set.*

Use case 1: user pastes a link to an article.
They want to know the related articles and related field.

- First, check to see if this has been searched before. If so, retrieve from the data warehouse. If not:
- Search through arxiv api
- Extract keywords and info from that paper. Citations, and also the abstract / conclusion for semantic search.
- Clean out the results, store them
- Output the map

Use case 2: user types a general topic ("transformers")

- First, check if they already exists. (it's in our database of common keywords / nodes). If not:
- Arxiv api: top 20 results
- For each of those papers, it does this process:
    - Extract keywords and info from that paper. Citations, and also the abstract / conclusion for semantic search.
    - Clean out the results, store them
- Output the map

Use case 3: starting maps / keywords ("Physics" → "nuclear physics", "quantum mechanics", "theoretical physics")

- Similar to use case 2, except using some prechosen starting maps/keywords instead of user search
- This prepopulate our database with some entries to give an idea of what our application is like

Code Snippet

Here is a snippet from main.py, which shows a sample run of use case 2

```python
from config import app
from ingestion import ingest_from_arxiv
from transformation import main as transform_main
from embedding_worker import run_embedding_batch, backfill_similar_ids
from graph_worker import build_knowledge_graph

@app.local_entrypoint()
def pipeline(
    query: str = "transformers",
    max_results: int = 50,
    threshold: int = 50,
    k: int = 10
):
    """Full pipeline orchestrator"""
    print("Step 1: Ingesting papers from arXiv...")
    ingest_from_arxiv.remote(query=query, max_results=max_results)

    print("Step 2: Transforming Bronze → Silver...")
    transform_main.remote()

    print("Step 3: Generating embeddings...")
    run_embedding_batch.remote(
        limit=max_results,
        populate_similar=True,
        min_corpus_size_for_neighbors=threshold,
        k=k
    )

    print("Step 4: Building knowledge graph...")
    build_knowledge_graph.remote()

    print("✓ Pipeline complete!")
```

Next Steps

Paper Understanding

Once we have the map, the next step is paper understanding and summarization. In this writeup, we focused on implementing the basic knowledge graph creation feature. As a next step, we can expand our schema to also store not just the embedding of the raw abstract and conclusion, but also extract key points of the paper using the full pdf in conjunction with the abstract and conclusion, and store these key points in our silver layer as well. Some points of interest to investigate for this topic could be how to extract key ideas and points from the paper, how to identify what its main research question is, the next steps or remaining gaps from the paper. Having good logic for extracting the key insights from the paper could also help in expanding the graph edges as well.

Expanding Knowledge Graph Edges

Currently, we have two types of edges between papers: citation, and semantic similarity. Citation is one of the most direct methods of assessing the relationship between papers, and one of the standards implemented by most competitors in the space such as Semantic Scholar. However, in order to have a more holistic understanding of a field or the related papers, just citations is often insufficient and does not offer a wide enough view, potentially missing many other crucial relationships.

In order to expand this, we also use embeddings to incorporate edges that quantify the semantic similarity between pairs of papers. The embeddings are based on the abstract and the conclusion of each paper. As a next step, we can improve the logic in this section further. Instead of solely relying on embeddings as a “similarity” metric, how might we identify more concrete positive/negative directions? For example, we could use the embeddings as an initial sieve to filter out a subset of “related paper” candidates, then use a LLM to rerank and identify if paper 1 supports or contradicts the main point of paper 2.

Rate Limits

One practical challenge we have encountered is managing the strict rate limits associated with the ArXiv and Semantic Scholar APIs, which currently bottleneck our ingestion speed and system responsiveness. To build a more resilient pipeline, as a next step we can try implementing an asynchronous queuing system within our Modal compute layer. This could smooth out traffic and ensure we stay within permitted request thresholds. We also plan to look into integrating a better caching layer that checks the Snowflake Silver table before every external call to minimize redundant

network overhead and prevent unnecessary API consumption, and some more robust retry logic. Overall, we need to figure out a better way to address the rate limit issue, either with a better / more complete offline loading schema such that the calls to ArXiv / semantic scholar by the user is a much less frequent occurrence than it currently is.

Part 4 - Infrastructure as Code Implementation

For our Infrastructure as Code implementation, we have created two modules representing our development and production environments within Snowflake. The two environments have separate database resources, which ensures that if one environment (ex. dev) loses data, the other one isn't affected. This setup is simple, maintainable, and can be easily adjusted if future needs require larger databases or additional warehouses in production.

We have chosen to work exclusively with Snowflake, as it provides a straightforward and reliable solution for storing, querying, and managing research paper data. By focusing on Snowflake alone, we avoid unnecessary complexity from other cloud platforms, such as AWS, while still enabling future expansion if additional tools or storage capabilities are required.

Our implementation emphasizes modularity and clarity, using Terraform modules for both environments. This ensures that infrastructure is reusable, well-organized, and aligned with the earlier parts of the project, including data ingestion and processing pipelines. Each module encapsulates the necessary resources, like databases, schemas, and warehouses, allowing us to maintain consistency between environments while supporting environment-specific configurations.

Additionally, while scripts were considered for some tasks, we prioritized Terraform modules to ensure that all infrastructure changes are version-controlled, auditable, and reproducible, meeting the marking criteria for code clarity, organization, and completeness. This approach also allows for seamless scaling or modification of resources as project requirements evolve, while maintaining alignment across development and production environments.

Part 5 - Disaster Recovery Demonstration

The recovery demonstration is provided here: https://drive.google.com/file/d/1Nbxug4lKQqZI_cBctVgP5VktJiYjlc12/view?usp=sharing

This just showcases how our production database is able to recover while the developer database is still running. Provides an exhaustive list, as we show at each step (pre-delete, deletion, creation) what both databases look like.

Our infrastructure is configured carefully, as mentioned in part 4, so that we have a different schema for each tier or type of data, and the configuration of terraform reflect that. All compute resources, including `MINDMAP_PROD_WH` warehouse, are defined with auto-suspend and auto-resume settings for cost efficiency. All environment-specific values, like account credentials, are managed via Terraform variables (snowflake_account, snowflake_user, snowflake_password). This allows reproducible deployments in DEV or PROD.

Access control is controlled through Snowflake roles. Terraform connects with SYSADMIN, which allows us to create databases, schemas, tables, and warehouses. Credentials are never hard-coded; instead, we use Terraform variables and Modal secrets for safe storage. This ensures that only authorized applications and users can access production resources