SDM Lab-2 Project

By

Kaiwen Yuan
Nishant Sushmakar
GitHub Repo Link

Universitat Politècnica de Catalunya

May 2025

Contents

1 Part B.1

1.1 Modelling TBOX

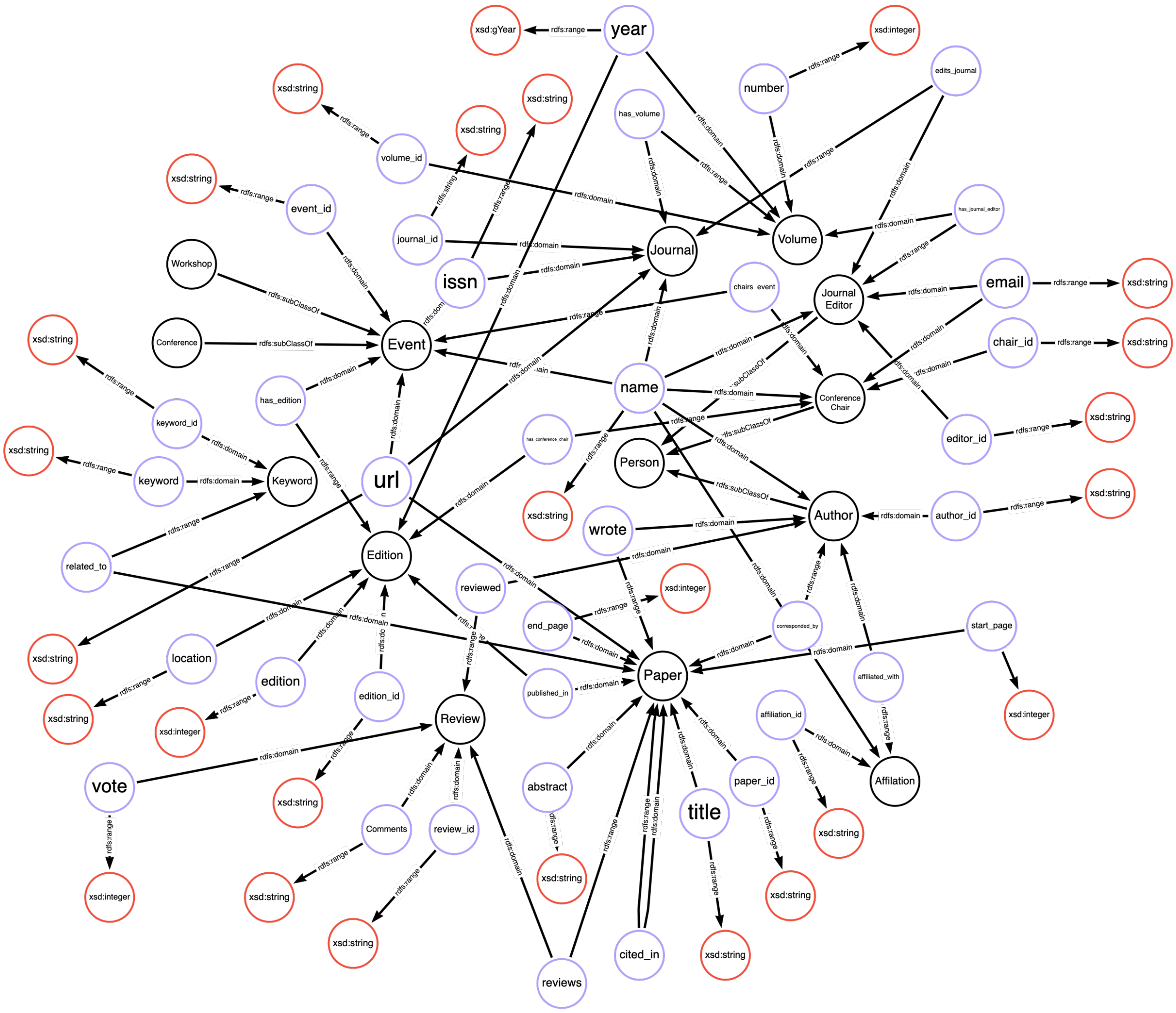

Figure 1: Graphical Representation of TBOX

Figure 1 presents the complete TBox representation of the research landscape. Classes (`rdf:type` values) are depicted in black, representing `rdfs:Class`, while properties (`rdf:Property`) are shown in purple. To avoid excessive complexity in the visual representation, explicit references have not been included in the graph. The complete graph is provided as part of the submission for those who wish to review the full, detailed structure.

Ontology Classes Overview

The ontology defines 14 main classes to model the research landscape. These are grouped hierarchically by role and function:

- **Person** (Base class): Represents any individual involved in research.

- **Author**
  Properties: `author_id`, `name`, `affiliated_with`
- **JournalEditor**
  Properties: `editor_id`, `name`, `email`
- **ConferenceChair**
  Properties: `chair_id`, `name`, `email`

- **Paper**
  Properties: `paper_id`, `title`, `abstract`, `url`, `start_page`, `end_page`
- **Journal**
  Properties: `journal_id`, `name`, `issn`, `url`
- **Volume**
  Properties: `volume_id`, `number`, `year`
- **Keyword**
  Properties: `keyword_id`, `keyword`
- **Affiliation**
  Properties: `affiliation_id`, `name`
- **Review**
  Properties: `review_id`, `comments`, `vote`
- **Edition**
  Properties: `edition_id`, `edition`, `year`, `location`
- **Event** (Base class): Represents academic events.
  - **Conference**
  - **Workshop**

  Properties: `event_id`, `name`, `issn`, `url`

Object Properties

Object properties define semantic relationships between instances of classes. They are grouped by functional domains:

- **Author-related**
  - `wrote`: `Author` → `Paper`
  - `affiliated_with`: `Author` → `Affiliation`
  - `reviewed`: `Author` → `Review`

- **Paper-related**
  - corresponded_by: Paper → Author
  - cited_in: Paper → Paper
  - related_to: Paper → Keyword
  - published_in: Paper → Edition
- **Event-related**
  - has_edition: Event → Edition
  - has_conference_chair: Edition → ConferenceChair
- **Journal-related**
  - has_volume: Journal → Volume
  - has_journal_editor: Volume → JournalEditor
  - edits_journal: JournalEditor → Journal
- **Review-related**
  - reviews: Review → Paper

Data Properties

These properties capture literal attributes of class instances, typed using xsd datatypes:

- **Identifier Properties** (xsd:string)
  Used to uniquely identify instances.
  Examples: paper_id, author_id, journal_id
- **Name Properties** (xsd:string)
  Used to denote the name of an entity.
  Examples: name (used in Author, Journal, Event)
- **URL Properties** (xsd:string)
  Used to store links or references.
  Examples: url (used in Paper, Journal, Event)
- **Numeric Properties** (xsd:integer)
  Represent numeric values.
  Examples: start_page, end_page, edition, number, vote
- **Date Properties** (xsd:gYear)
  Represent years.
  Example: year

- **Text Properties** (`xsd:string`)
  Represent longer or descriptive textual content.
  Examples: `abstract`, `comments`, `keyword`

1.2 Ontology Construction using RDFLib

The ontology was programmatically constructed using Python's `rdflib` library. The following steps outline the creation and annotation of classes, object properties, and data properties in the ontology:

Class Definitions

Classes are defined and enriched with human-readable metadata:

- Each class is added to the RDF graph as an instance of `rdfs:Class`.
- A human-readable label is attached using `rdfs:label`.
- A description or comment is added using `rdfs:comment`.

```
for class_name, label, comment in classes:
    g.add((RESEARCH[class_name], RDF.type, RDFS.Class))
    g.add((RESEARCH[class_name], RDFS.label, Literal(label)))
    g.add((RESEARCH[class_name], RDFS.comment, Literal(comment)))
```

Subclass Relationships

Subclass hierarchies are established using `rdfs:subClassOf`, connecting specialized classes to their parent classes. For example:

- `Author`, `JournalEditor`, and `ConferenceChair` are subclasses of `Person`.
- `Conference` and `Workshop` are subclasses of `Event`.

```
g.add((RESEARCH.Author, RDFS.subClassOf, RESEARCH.Person))
g.add((RESEARCH.JournalEditor, RDFS.subClassOf, RESEARCH.Person))
g.add((RESEARCH.ConferenceChair, RDFS.subClassOf, RESEARCH.Person))
g.add((RESEARCH.Conference, RDFS.subClassOf, RESEARCH.Event))
g.add((RESEARCH.Workshop, RDFS.subClassOf, RESEARCH.Event))
```

Object Properties

Object properties are used to define relationships between instances of different classes:

- Each property is declared as an instance of `rdf:Property`.
- The domain and range of the property are set using `rdfs:domain` and `rdfs:range`.
- Each property includes a label and a descriptive comment.

```
for prop_name, domain, range, label, comment in object_properties:
    prop = RESEARCH[prop_name]
    g.add((prop, RDF.type, RDF.Property))
    g.add((prop, RDFS.domain, RESEARCH[domain]))
    g.add((prop, RDFS.range, RESEARCH[range]))
    g.add((prop, RDFS.label, Literal(label)))
    g.add((prop, RDFS.comment, Literal(comment)))
```

Data Properties

Data properties are used to assign literal values (e.g., strings, integers) to instances:

- Similar to object properties, they are typed as `rdf:Property`.
- The domain is set to a class, and the range is set to an XML Schema datatype (e.g., `xsd:string`, `xsd:integer`).
- Labels and comments provide human-readable documentation.

```
for prop_name, domain, range_type, label, comment in data_properties:
    prop = RESEARCH[prop_name]
    g.add((prop, RDF.type, RDF.Property))
    g.add((prop, RDFS.domain, RESEARCH[domain]))
    g.add((prop, RDFS.range, range_type))
    g.add((prop, RDFS.label, Literal(label)))
    g.add((prop, RDFS.comment, Literal(comment)))
```

2 Part B.2

2.1 Modelling ABOX

You can access the file on Google Drive here: Google Drive Link

The methodology encompasses several key phases:

Data Integration and Preprocessing The transformation begins with loading and validating CSV files containing research publication data. The initial dataset consists of 22 CSV files derived from Lab1, which provides the foundational research publication data including papers, authors, journals, conferences, affiliations, keywords, citations, and review information. However, to fully comply with our TBox ontology definition, additional data was required for classes such as `JournalEditor`, `ConferenceChair`, and specific relationship mappings not present in the original dataset.

To address these gaps, we developed a data generation script (`generate_missing_data.py`) that creates 9 additional CSV files. The generation process employs strategic sampling and synthetic data creation:

- **Journal Editor Generation**: 20% of existing authors are randomly selected and assigned as journal editors, with synthetic email addresses generated using domain patterns typical of academic institutions.

- **Conference Chair Generation**: 15% of remaining authors (excluding those already selected as editors) are designated as conference chairs to avoid role overlap.
- **Relationship Mapping**: Editor-journal, chair-event, volume-editor, and edition-chair relationships are established through controlled random assignment, ensuring realistic distribution patterns (e.g., each editor manages 1-3 journals, each chair oversees 1-2 events).
- **Review Data Restructuring**: The original author-paper review relationships are transformed into a three-entity model (Author-Review-Paper) to align with the TBox definition of Review as an independent class rather than a relationship property.

The combined dataset of 31 CSV files (22 original + 9 generated) provides comprehensive coverage of all classes and relationships defined in our TBox ontology, ensuring semantic completeness while maintaining data integrity and realistic distribution patterns.

Entity URI Generation A consistent URI scheme is implemented to ensure unique identification of all entities. Each entity type follows the pattern `http://example.org/resource/{type}/{identifier}`, where the identifier is derived from the original CSV data's primary keys. This approach maintains referential integrity across different data sources while ensuring global uniqueness.

```
# Define namespaces
RESEARCH = Namespace("http://example.org/research#")
RESOURCE = Namespace("http://example.org/resource/")

# Create resource URI function
def create_uri(resource_type, identifier):
    return RESOURCE[f"{resource_type}/{str(identifier)}"]

# Example usage
paper_uri = create_uri("paper", "12345")
# Results in: http://example.org/resource/paper/12345
author_uri = create_uri("author", "auth_001")
# Results in: http://example.org/resource/author/auth_001
```

Type Assignment Strategy The implementation employs a hierarchical type assignment strategy that leverages the class inheritance structure defined in the TBox. For instance, `Author`, `JournalEditor`, and `ConferenceChair` instances are assigned both their specific type and the parent `Person` type. This dual typing enables both specific and general queries while supporting inference-based reasoning.

```
def add_authors():
    authors_df = load_csv("author.csv")
    for _, row in authors_df.iterrows():
        author_uri = create_uri("author", row['authorId'])
        # Add both Person and Author types (hierarchical typing)
        g.add((author_uri, RDF.type, RESEARCH.Person))
        g.add((author_uri, RDF.type, RESEARCH.Author))
        g.add((author_uri, RESEARCH.author_id, Literal(row['authorId'])))

        if pd.notna(row['name']):
            g.add((author_uri, RESEARCH.name, Literal(row['name'])))
```

```python
def add_journal_editors():
    editors_df = load_csv("journal_editor.csv", generated=True)
    for _, row in editors_df.iterrows():
        editor_uri = create_uri("editor", row['editorId'])
        # Add both Person and JournalEditor types
        g.add((editor_uri, RDF.type, RESEARCH.Person))
        g.add((editor_uri, RDF.type, RESEARCH.JournalEditor))
        g.add((editor_uri, RESEARCH.editor_id, Literal(row['editorId'])))

        if pd.notna(row['name']):
            g.add((editor_uri, RESEARCH.name, Literal(row['name'])))
```

Data Property Mapping Literal values from CSV columns are mapped to appropriate data properties with proper XSD datatype assignment. String values are typed as `xsd:string`, numeric values as `xsd:integer`, and year values as `xsd:gYear`. Special handling is implemented for complex data such as page ranges, which are parsed and split into separate start and end page properties.

```python
def add_papers():
    papers_df = load_csv("paper.csv")
    # Load page information from relationship files
    vol_pages_df = load_csv("paper_publishedIn_volume.csv")
    vol_pages_dict = dict(zip(vol_pages_df['paperId'], vol_pages_df['pages']))

    for _, row in papers_df.iterrows():
        paper_uri = create_uri("paper", row['paperId'])
        g.add((paper_uri, RDF.type, RESEARCH.Paper))

        # String properties
        g.add((paper_uri, RESEARCH.paper_id, Literal(row['paperId'])))
        if pd.notna(row['title']):
            g.add((paper_uri, RESEARCH.title, Literal(row['title'])))
        if pd.notna(row['abstract']):
            g.add((paper_uri, RESEARCH.abstract, Literal(row['abstract'])))

        # Complex data handling: page ranges
        if row['paperId'] in vol_pages_dict:
            pages = vol_pages_dict[row['paperId']]
            try:
                if '-' in pages:
                    start_page, end_page = pages.split('-')
                    # Integer properties with explicit typing
                    g.add((paper_uri, RESEARCH.start_page, Literal(int(start_page))))
                    g.add((paper_uri, RESEARCH.end_page, Literal(int(end_page))))
            except:
                pass  # Handle invalid page format

def add_editions():
    editions_df = load_csv("edition.csv")
    for _, row in editions_df.iterrows():
        edition_uri = create_uri("edition", row['editionId'])
        g.add((edition_uri, RDF.type, RESEARCH.Edition))

        # Integer property
        if pd.notna(row['edition']):
            g.add((edition_uri, RESEARCH.edition, Literal(int(float(row['edition'])))))

        # Year property with XSD datatype
        if pd.notna(row['year']):
            g.add((edition_uri, RESEARCH.year,
                   Literal(int(row['year']), datatype=XSD.gYear)))
```

Relationship Establishment Object properties are established by creating triples that link entity URIs based on foreign key relationships present in the CSV data. The system handles one-to-many and many-to-many relationships by processing separate relationship files that contain paired identifiers.

```
# Many-to-many relationship: Author wrote Paper
def add_author_wrote_paper():
    relations_df = load_csv("author_wrote_paper.csv")
    for _, row in relations_df.iterrows():
        author_uri = create_uri("author", row['authorId'])
        paper_uri = create_uri("paper", row['paperId'])
        g.add((author_uri, RESEARCH.wrote, paper_uri))

# One-to-many relationship: Paper corresponded by Author
def add_paper_corresponded_by_author():
    relations_df = load_csv("paper_correspondedBy_author.csv")
    for _, row in relations_df.iterrows():
        paper_uri = create_uri("paper", row['paperId'])
        author_uri = create_uri("author", row['authorId'])
        g.add((paper_uri, RESEARCH.corresponded_by, author_uri))

# Complex relationship with Review restructuring
def add_reviews():
    reviews_df = load_csv("review_relations.csv")
    for _, row in reviews_df.iterrows():
        # Create unique review identifier
        review_id = f"rev_{row['authorId']}_{row['paperId']}"
        review_uri = create_uri("review", review_id)

        g.add((review_uri, RDF.type, RESEARCH.Review))
        g.add((review_uri, RESEARCH.review_id, Literal(review_id)))

        if pd.notna(row['comments']):
            g.add((review_uri, RESEARCH.comments, Literal(row['comments'])))

        # Establish three-way relationships: Author-Review-Paper
        author_uri = create_uri("author", row['authorId'])
        paper_uri = create_uri("paper", row['paperId'])

        g.add((author_uri, RESEARCH.reviewed, review_uri))
        g.add((review_uri, RESEARCH.reviews, paper_uri))
```

2.2 Inference Regime Entailment

Our knowledge graph implementation leverages RDFS-optimized inference in GraphDB to derive implicit knowledge from explicitly asserted facts. The system utilizes several key RDFS inference rules to enrich the knowledge graph:

Subclass Inference (rdfs9) When an instance is declared as a member of a subclass, it is automatically inferred to be a member of all superclasses. In our implementation:

- All `Author` instances are automatically classified as `Person` instances
- All `JournalEditor` instances are automatically classified as `Person` instances
- All `ConferenceChair` instances are automatically classified as `Person` instances
- All `Conference` instances are automatically classified as `Event` instances

- All `Workshop` instances are automatically classified as `Event` instances

Domain and Range Inference (rdfs2, rdfs3) Property domain and range constraints automatically assign types to resources:

- Resources appearing as subjects of `research:wrote` are inferred to be `Author` instances
- Resources appearing as objects of `research:wrote` are inferred to be `Paper` instances
- Resources appearing as subjects of `research:edits_journal` are inferred to be `JournalEditor` instances

2.3 Knowledge Graph Statistics and Summary

The resulting knowledge graph represents a comprehensive model of the research publication domain. Table 1 presents detailed statistics about the instances and relationships within our ABOX.

Table 1: Knowledge Graph Statistics Summary

Metric	Count
Class Instances	
Papers	15,199
Persons (Total)	104,778
- Authors	77,614
- Journal Editors	15,522
- Conference Chairs	11,642
Journals	2,667
Events (Total)	519
- Conferences	504
- Workshops	15
Editions	969
Volumes	7,711
Keywords	6,683
Affiliations	952
Reviews	35,633
Property Usage	
Author-Paper (wrote)	94,403
Paper Citations (cited_in)	397,702
Paper-Keyword (related_to)	30,947
Paper Correspondence (corresponded_by)	15,166
Publication Relationships (published_in)	13,012
Review Relationships (reviewed/reviews)	35,633
Editorial Relationships (edits_journal)	31,067
Chair Relationships (chairs_event)	17,434
Author-Affiliation (affiliated_with)	1,144
Journal-Volume (has_volume)	7,711
Event-Edition (has_edition)	969
Overall Statistics	
Total RDF Triples	1,437,855
Total Classes	14
Total Properties	25

3 Part B.3

3.1 Subject Matter Expertise of the Reviewers

Implementation Code

```
PREFIX research: <http://example.org/research#>
PREFIX rdfs : < http :// www . w3 . org /2000/01/ rdf-schema# >
PREFIX xsd : < http :// www . w3 . org /2001/ XMLSchema# >
PREFIX rdf : < http :// www . w3 . org /1999/02/22-rdf-syntax-ns# >
SELECT ?reviewerName (GROUP_CONCAT(DISTINCT ?expertise; SEPARATOR=", ") AS ?expertiseAreas)
WHERE {
  ?reviewer research:reviewed ?review ;
              research:name ?reviewerName .
  ?review research:reviews ?paper .
  ?paper research:related_to/research:keyword ?expertise .
}
GROUP BY ?reviewer ?reviewerName
```

Output

reviewerName	expertiseAreas
Yu-Le Yong	degron, cognitive test, keap1, neurocognitive, health, genome-wide association study, genetic admixture, genetic architecture, genetic genealogy
Tomasz Nazim	degron, benzonitrile, keap1, annulene
Yunfeng Hu	foam cell, disease, molecularly imprinted polymer, molecular imprinting, chromatographic, pathogenesis, molecular, synthesis, alanine
M. Wadelius	foam cell, disease, pathogenesis, alanine
B. Brus	flavoprotein, flavin-containing monooxygenase, hydroxylamine, nitroso, nitrone
Y. Abubakar	flavoprotein, flavin-containing monooxygenase, hydroxylamine, nitroso, nitrone, complex matrix, molecularly imprinted polymer, nanomaterials, sample preparation, carbon fibers
O. Lakhneko	flavoprotein, grass carp, flavin-containing monooxygenase, hydroxylamine, nitroso, nitrone, comparative genomics, sequence assembly, genome size
C. Barsamian	orchidaceae, two-dimensional chromatography, phalaenopsis, dendrobium, genomic, comparative genomics, genome size
Tao Wang	orchidaceae, phalaenopsis, dendrobium, bistability, intensity, exponential decay, genomic, comparative genomics, exponent, metastability, genome size
Tapabrata (Rohan) Chakraborty	onboarding, transpose, matrix representation, sentiment analysis, matrix (chemical analysis), representation, scope (computer science), disinformation, edge device
F. Henriques	onboarding, scope (computer science), big data, data analysis, homomorphic encryption, edge device
Tanya K. Ronson	fertility clinic, menarche, disease, polygenic risk score, thematic analysis, hyperlipidemia
U. Ben-David	fertility clinic, menarche, data sharing, distributed data store, thematic analysis
Boyu Zhang	restricted boltzmann machine, feature (linguistics), boltzmann machine, initialization
L. Coelho	restricted boltzmann machine, feature (linguistics), boltzmann machine, initialization
S. Fuhrmann	manihot esculenta

Results Interpretation

The query retrieves information about the subject matter expertise of reviewers based on the keywords associated with the papers they have reviewed for conferences and journals. These keywords provide a useful indication of each reviewer's area of expertise. This information can be leveraged in multiple ways: to match reviewers with relevant submissions more accurately, to assess the diversity and distribution of expertise across the reviewer pool, and to support the development of reviewer recommendation systems. By understanding the areas in which reviewers are most knowledgeable, the overall quality, fairness, and relevance of the review process can be significantly improved.

3.2 Dense Collaboration Network Among the Research Institutes

Implementation Code

```
PREFIX research: <http://example.org/research#>
PREFIX rdfs : < http :// www . w3 . org /2000/01/ rdf-schema# >
PREFIX xsd : < http :// www . w3 . org /2001/ XMLSchema # >
PREFIX rdf : < http :// www . w3 . org /1999/02/22-rdf-syntax-ns# >
SELECT ?affiliationName (COUNT(DISTINCT ?author) AS ?researchers)
        (GROUP_CONCAT(DISTINCT ?coAuthor; SEPARATOR="; ") AS ?collaborators)
WHERE {
  ?author research:affiliated_with ?affiliation ;
        research:wrote ?paper .
  ?coAuthor research:wrote ?paper ;
                research:affiliated_with ?affiliation .
  ?affiliation research:name ?affiliationName .
  FILTER (?author != ?coAuthor)
}
GROUP BY ?affiliation ?affiliationName
```

Output

affiliationName	researchers	collaborators
Shanghai Jiao Tong University	4	http://example.org/resource/author/38735298; http://example.org/resource/author/79494403; http://example.org/resource/author/2144908858; http://example.org/resource/author/144248374
Beijing Institute of Technology	2	http://example.org/resource/author/2298857218; http://example.org/resource/author/144843219
Adobe Research	2	http://example.org/resource/author/2462276; http://example.org/resource/author/145262461
University of Rochester	2	http://example.org/resource/author/33642939; http://example.org/resource/author/145585312
1 Department of Public Health and Primary Care, Centre for Biomedical Ethics and Law, University of Leuven, Leuven, Belgium .	2	http://example.org/resource/author/37402205; http://example.org/resource/author/14756295
University of Potsdam	2	http://example.org/resource/author/2045980201; http://example.org/resource/author/1683688
Zhejiang University	2	http://example.org/resource/author/2109584311; http://example.org/resource/author/1694815
University of Oregon	2	http://example.org/resource/author/35539899; http://example.org/resource/author/1811211
University of Oxford	2	http://example.org/resource/author/2536934; http://example.org/resource/author/2146147410
Computer Engineering and Informatics Department, University of Patras	2	http://example.org/resource/author/2156929752; http://example.org/resource/author/2156928873
University of Manchester	2	http://example.org/resource/author/145229872; http://example.org/resource/author/2165227439
Seoul National University	2	http://example.org/resource/author/3091593; http://example.org/resource/author/2575874
Heriot-Watt University	2	http://example.org/resource/author/1389544608; http://example.org/resource/author/2621022
University of the Basque Country UPV/EHU	2	http://example.org/resource/author/1453724884; http://example.org/resource/author/3242916

Results Interpretation

The query identifies institutions with dense collaboration networks among their researchers. This insight can be used to recognize research intensive institutions or ”research powerhouses.” It also helps research supporters such as funding agencies and policymakers assess and prioritize institutions that demonstrate strong internal collaboration. Additionally, the results can guide institutions in identifying gaps in their internal research networks, enabling them to promote interdisciplinary collaboration and make strategic cross-disciplinary hires to strengthen their collaborative ecosystem.