COMP 474/6741: INTELLIGENT SYSTEMS

WINTER 2024

Intelligent System Project

Instructor

Dr. René Witte

Submitted By

Saurabh Sharma (40226298) [Software Engineering]

Anubhav Mahajan (40267770) [MACS]

Team name: AK_G_01

GitHub URL:

https://github.com/saurabhs679/COMP-6741-Project

We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality

Signature Anubhav 1 ID Number:40267770 Date-15-04-2024

Signature Saurabh Sharma 2 — ID Number: 40226298 Date-15-04-2024

Vocabulary

The RDF schema outlines the structure for representing university-related data, containing courses, lectures, students, topics, and universities. Each class is defined, along with their respective properties for example course class with properties such as course name, student ID, lecture content, and topic name. Object properties establish relationships between entities, indicating which lectures are part of a course and which topics are covered in a course or lecture. The schema provides a structured framework for organizing and querying university data.

The knowledge base schema is represented using RDF (Resource Description Framework), with specific classes and properties developed to describe the domain's entities and relationships. Here is a summary of the modeling decisions:

Standard Vocabularies:

Standard vocabularies such as rdf, rdfs, and their associated properties (rdf:type, rdfs:label, rdfs:domain, rdfs:range) are utilized for defining classes, properties, and their characteristics

Classes and Properties:

Classes - ex:Course, ex:Lecture, ex:Student, ex:Topic, and ex:University. These classes provide a structured framework for categorizing and organizing information.

Properties - Properties explain an entity's attributes and relationships. Each property is connected with a

domain and a range, which define its applicability and possible values. ex:CourseName, ex:CourseNumber, and ex:Credits are characteristics that record precise facts about courses, whereas ex:IDNumber, ex:firstName, and ex:lastName describe student-related information.

Vocabulary Extensions:

We also developed a few vocabularies extension such as ex:topic_in_course, ex:topic_in_lecture to define the relationship between different classes like first one shows relationship between course and topic . Similarly, ex:topic_in_lecture links individual lectures to the topics they address. These extensions enhance the schema relation, enabling more accurate querying and displays more comprehensive representation of the knowledge base.

This design structure lays the groundwork for efficient organization, retrieval, and integration of information.

Knowledge Base Construction

Our dataset consists of five distinct datasets, each based on specific classes and their corresponding properties within our knowledge base schema. The datasets contain structured information relevant to courses, lectures, students, topics, and universities.

Courses Dataset: Contains information about various courses offered, including course name, number, credits, description, subject, outline, and webpage link.

Lectures Dataset: Provides details about individual lectures, such as lecture name, number, content, and webpage link.

Students Dataset: Includes information about enrolled students, such as student ID, first name, last name, email, and completed courses.

Topics Dataset: Contains data regarding topics covered within lectures or courses, along with topic names and DBpedia links for additional context.

Universities Dataset: Provides information about universities, including university name and DBpedia link.

We are using python script to parse and read all the datasets and converting it into a structured format compatible with the Turtle serialization.We are doing this for all 5 datasets merging them together with main vocubalary to populate the knowledge graph.

To run the tools and create the knowledge base using the provided Python notebook, follow these steps:

Prepare CSV Files:

Ensure that you have the necessary CSV files containing the data for students, courses, universities, lectures, and topics. Each CSV file should follow a specific format with columns representing different attributes such as student ID, course name, lecture content, etc.

Open the Python Notebook:

Open the Python notebook file containing the code for populating each aspect of the knowledge base using the rdflib library.

Execute Cells:

Execute each cell in the notebook using a Python interpreter or a Jupyter notebook environment. Each cell corresponds to a section of code responsible for populating a specific aspect of the knowledge base (e.g., student data, course data, etc.).

Verify Output:

After executing each cell, verify that the output is generated successfully without any errors. The output will typically be in the form of RDF triples serialized in Turtle format, printed to the notebook's output cell.

Load RDF Triples:

Once you have generated RDF triples for each aspect of the knowledge base, load them into a triple store or graph database for storage and querying. You can use a triple store such as Apache Jena.

Execute SPARQL Queries:

Within the notebook, execute SPARQL queries directly against the populated RDF graph. Use the rdflib.plugins.sparql.prepareQuery function to prepare the query, and then execute it using the RDF graph object. For example, you can execute a SPARQL query to retrieve competencies related to a specific course as follows:

```
from rdflib.plugins.sparql import prepareQuery
query = prepareQuery(f"""
    PREFIX ex: <http://example.org/>
```

```
    SELECT ?competency

    WHERE {{

      ?course ex:completedCourse "COMP 6481"
;               ex:competentIn ?competency .

    }}

""", initNs={"ex": ex})

# Execute the query and print the
competencies results = g1.query(query)
for row in results:
print(row.competency)
```

By following these steps and executing the provided Python notebook, you can effectively populate the knowledge base from the CSV files and perform querying and analysis tasks.

Graph Queries

1. **List all courses offered by [university]:**

    - This query retrieves the names and numbers of all courses offered by a specific university.
    - It uses the ex:CourseName and ex:CourseNumber properties to fetch the course details.
    - The ?course variable represents each course in the graph that matches the criteria.
    - It selects the course name and course number using the SELECT clause.
    - The ?course variable is bound to instances of the ex:Course class.

Output-

Course Name: PPS

Course Number: 6481

Course Name: Intelligent Systems Course Number:
6741

2. **In which courses is [topic] discussed?**

    - This query finds all courses where a specific topic is discussed.
    - It looks for instances of the topic with the given name using ex:TopicName.
    - It then retrieves the names of courses where the topic is discussed using the ex:topic_in_course property.
    - The ?courseName variable represents the names of courses where the topic is discussed.

Output-

PPS

3. **Which [topics] are covered in [course] during [lecture number]?**

    - This query retrieves the names of topics covered in a specific course during a particular lecture.
    - It selects topics associated with the given lecture number using the ex:lectureNumber property.
    - The ?topicName variable represents the names of topics covered in the specified lecture of the course.

Output-

Algorithm efficiency

4. **List all [courses] offered by [university] within the [subject] (e.g., "COMP", "SOEN"):**

    - This query fetches the names and numbers of courses offered by a university within a specific subject area.

- It filters courses based on the subject using the ex:Subject property.
- The ?university variable represents the university, and the ?courseName and ?courseNumber variables represent course details.

Output-

Course Name: PPS

Course Number: 6481

Course Name: Intelligent Systems Course Number:
6741

5. **What [materials] (slides, readings) are recommended for [topic] in [course] [number]?**

- This query fetches the study material for the topic.
- It uses ex:TopicName and ex:topic_in_lecture to traverse the graph to fetch details to filter and output is found in ex:Content

Output-

Slides: aa.pdf Worksheets:
Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

6. **How many credits is [course] [number] worth?**

- This query retrieves the number of credits associated with a specific course number.
- It filters courses based on the course number using the ex:CourseNumber property.
- The ?credits variable represents the number of credits associated with the course

Output-

4

7. **For [course] [number], what additional resources (links to web pages) are available?**
    - This query retrieves the webpages associated with a specific course number.
    - It focuses on extracting additional resources specified by the predicate ex:WebpageLink

Output-

https://www.concordia.ca/academics/graduate/calendar/current/gina-cody-school -of-engineering-and-computer-science-courses/computer-science-and-software-en gineering-master-s-and-phd-courses.html

8. **Detail the content (slides, worksheets, readings) available for [lecture number] in [course] [number].**
    - This query retrieves the content associated with a specific course number for the given lecture
    - It uses ex:CourseNumber, ex:lectureNumber, and ex:Content to specify relationships in the RDF graph and give output

Output-

Slides: slides03.pdf

Worksheets: worksheet03.pdf

Readings: [Yu14, Chapter 4] (RDFS), [Yu14, Chapter 7] (FOAF)

Slides: Recursion.pdf Worksheets:
Readings: https://www.geeksforgeeks.org/introduction-to-recursion-data-struct ure-and-algorithm-tutorials/

9. **What reading materials are recommended for studying [topic] in [course]?**
    - This query retrieves reading materials recommended for studying a specific topic in a course.
    - It filters content containing "Readings" by using predicates like ex:CourseName, ex:topic_in_lecture, and ex:Content to specify relationships in the RDF graph.

Output-

Slides: aa.pdf Worksheets:

Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

10. What competencies [topics] does a student gain after completing [course] [number]?

- This query identifies the competencies (topics) acquired by a student upon completing a specific course.
- It utilizes predicates such as ex:completedCourse and ex:competentIn to specify relationships in the RDF graph and retrieve the desired competency information.

Output-

Run Time Errors Hash
table

11. What grades did [student] achieve in [course] [number]?

- This query retrieves the grades achieved by a specific student in a particular course.
- It extracts the first name, last name, and grade of the student using predicates such as ex:firstName, ex:lastName, ex:completedCourse, and ex:grade from the RDF graph

Output-

Name: Anubhav Mahajan, Grade: A

Name: Saurabh Sharma, Grade: B

12. Which [students] have completed [course] [number]?

- This query identifies which students have completed a specific course.
- It retrieves the first name and last name of the students who completed the course identified by the course number "COMP 6481", using predicates such as ex:firstName, ex:lastName, and ex:completedCourse from the RDF graph.

Output-

Name: Anubhav Mahajan

Name: Saurabh Sharma

13. Print a transcript for a [student], listing all the course taken with their grades.

- This query generates a transcript for a specific student, listing all the courses taken along with their respective grades.
- It retrieves the completed courses and corresponding grades for the student named "Anubhav" using predicates such as rdf:type, ex:firstName, ex:completedCourse, and ex:grade from the RDF graph.

Output-

Transcript for Anubhav: Completed Course:
COMP 6481
Grade: A

Triplestore and SPARQL Endpoint Setup

We are using Apache Jena Fuseki to set up a Triplestore and SPARQL Endpoint. We are downloading the Apache Jena Fuseki distribution package and extracting it. We also need to make sure java is installed in our system and its environment path is configured. We can open it from command prompt using fusekiserver command and it runs on localhost 3030. We create a new dataset and Upload RDF data into the newly created dataset using the web interface. To access the SPARQL endpoint provided by Fuseki to query the loaded data, construct and execute SPARQL queries using the web interface.

Knowledge Base Topic Population.

- **Document Processing:**

Document processing is the initial step in the creation of topic triples. Various documents, including PDFs and DOCX files, are collected from diverse sources such as academic papers, textbooks, and online resources. The text extraction process involves utilizing libraries like pdfplumber and python-docx to extract textual content from these documents. Subsequently, preprocessing techniques like tokenization and named entity recognition (NER) using the spaCy library are applied to identify relevant entities, particularly topics or concepts, within the text.

- **Linking to Linked Open Data**

After Document processing, we leverage Named Entity Recognition (NER) to processed text files to identify entities such as persons, organizations, locations, etc. In addition to basic entity identification, a specific focus is placed on filtering out entities pertaining to artificial intelligence (AI) and machine learning (ML) using a predefined set of keywords. This step ensures that only relevant topics are targeted for further processing.

Once topics are identified through NER, we link them to entities in LOD repositories such as DBpedia using the DBpedia Spotlight API. This process involves matching identified topics with their corresponding entities in LOD based on semantic similarity or other criteria. The DBpedia Spotlight API returns linked entities along with additional information such as descriptions and categories, enriching the understanding of topics and providing valuable context for the knowledge base. By sending the identified entities to the API, the code retrieves linked entities from the DBpedia knowledge graph along with additional metadata such as URIs, surface forms, and types.

To ensure the quality and relevance of linked entities, we are applying filtering techniques. reconciling conflicts in cases of multiple linked entities for the same topic and applying domain-specific filtering to include only relevant topics in the knowledge base.

```
File: Text Mining_worksheet3.txt
Entity: NLP & Text Mining
URI: http://dbpedia.org/resource/Natural_language_processing
Surface Form: NLP
Types:

File: Text Mining_worksheet3.txt
Entity: NLP & Text Mining
URI: http://dbpedia.org/resource/Text_mining
Surface Form: Text Mining
Types:
```

- **Generating RDF Triples:**

Once topics are linked to LOD entities, we use python script to parse through saved text file containg the LOD information to generate a turtle file. Then we RDF triples in the form of subject-predicate-object statements from this turtle file. Each triple represents a relationship between entities, such as hierarchical relationships, associations, or properties. Additionally, we include additional properties such as provenance information (e.g., lecture, tutorial, worksheet) and course mappings to provide context for each topic.in the knowledge base. By incorporating these additional properties into our RDF triples, we create a robust knowledge base.

Afterwords we add it to the original Knowledge graph vocabulary.

Turte Format

```
ex:Steve_Jobs a ex:Topic ;
    ex:TopicName "Steve Jobs" ;
    ex:dbpediaLink <http://dbpedia.org/resource/Steve_Jobs> ;
    ex:provenance "worksheet" ;
    ex:topic_in_course ex:COMP_6741 ;
    ex:topic_in_lecture ex:Intelligent_Agents_worksheet3 .

ex:TensorFlow a ex:Topic ;
    ex:TopicName "TensorFlow" ;
    ex:dbpediaLink <http://dbpedia.org/resource/TensorFlow> ;
    ex:provenance "lecture" ;
    ex:topic_in_course ex:COMP_6741 ;
    ex:topic_in_lecture ex:Text_Mining .

```

RDF triples

```
<http://example.org/Information_Extraction> <http://example.org/TopicName> "Information Extraction" .
<http://example.org/Washington> <http://example.org/dbpediaLink> <http://dbpedia.org/resource/Washington_(state)> .
<http://example.org/hash> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Topic> .
<http://example.org/kNN> <http://example.org/provenance> "lecture" .
<http://example.org/Steve_Jobs> <http://example.org/provenance> "worksheet" .
```

Total Number of Triples	737
Number of Distinct Topics	118
Number of Topic Instances/Course	COMP6741-76 COMP6481-42

Chatbot Design

The chatbot design for our system revolves around using the Rasa framework to implement a natural language conversation interface. This section will delve into the various components of the chatbot, including intents, stories, entities, and actions. Additionally, it will cover the integration process with the Fuseki server for executing SPARQL queries based on user input. Moreover, the method for translating SPARQL query results into natural language answers, emphasizing the use of templates or LLM integration, will be discussed. Finally, concrete examples from each query type will be provided to showcase the chatbot's workflow.

The chatbot is designed to recognize various intents from user inputs, which are predefined categories representing the user's intention or goal. Intents include common greetings, farewells, affirmations, and specific queries related to courses, topics, materials, competencies, grades, students, and transcripts, among others. Stories outline the possible conversational paths that the chatbot can take based on a sequence of user inputs and the corresponding actions to be executed.

Entities are used to extract specific pieces of information from user inputs, such as person names, course codes, lecture numbers, topics, subjects, universities, course numbers, completed courses, and student names. These entities help the chatbot understand the context of the user's query and provide relevant responses.

Actions represent the behaviors or responses of the chatbot to user inputs. These actions can range from uttering predefined messages to executing custom logic, such as querying a knowledge base or external database, like the Fuseki server, to retrieve information. Examples of actions include providing course lists, fetching materials, listing competencies, showing grades, and generating student transcripts.

Integration with Fuseki Server

The integration process with the Fuseki server involves executing SPARQL queries to retrieve relevant information from the RDF knowledge base. Python's requests library is utilized to send HTTP requests to the Fuseki server's SPARQL endpoint, passing the query as a parameter. Upon receiving the response, the chatbot processes the results and generates appropriate natural language responses to convey the information to the user.

```
response = requests.post('http://localhost:3030/roboprof/query', data={'query': query})

results = response.json()
```

Handling OOV Words and Error Cases

To handle out-of-vocabulary (OOV) words and error cases, our chatbot employs several strategies, including the use of a fallback classifier, synonyms, and Python's if-else conditions for displaying appropriate error messages.

Fallback Classifier:

The fallback classifier serves as a safety net for handling user inputs that do not match any predefined intents. When the classifier detects an OOV word or an unrecognized intent, it triggers the fallback action, prompting the user to rephrase the query or provide additional information. This ensures that the chatbot can gracefully handle unexpected inputs and guide the user towards a successful interaction.

```
Your input ->  ewffOBUo doalguwpu; hjbdow;jda
I'm sorry, I didn't understand that. Can you please rephrase your question?
```

Synonyms:

To enhance the chatbot's understanding of user inputs, we incorporate synonyms for key terms and phrases within our training data. Synonyms expand the vocabulary of the chatbot, allowing it to recognize a broader range of expressions for the same concept. For example, synonyms for course codes, topic names, and other domain-specific terms enable the chatbot to accurately interpret user queries, even if the exact wording varies. By leveraging synonyms, we improve the chatbot's robustness and adaptability to diverse user inputs.

Error Handling with Python's If-Else Conditions:

In addition to the fallback classifier and synonyms, our chatbot utilizes Python's if-else conditions to handle error cases during query execution or processing. If an error occurs, such as a failed SPARQL query or connectivity issues with the Fuseki server, the chatbot evaluates the error condition using if-else statements. Based on the type of error encountered, the chatbot generates an appropriate error message to inform the user about the issue and guide them towards a resolution. These error messages are designed to be informative and user-friendly, helping users understand the nature of the problem and providing guidance on how to proceed.

By combining these strategies—fallback classifier, synonyms, and error handling with if-else conditions—our chatbot ensures robust and resilient performance, effectively managing OOV words and error cases to deliver a seamless conversational experience for users.

Translating SPARQL Query Results

For converting sparql queris results to natural language, Initially, we tried hugging face, but it was giving us result only in german language. So later, we employ Python code to convert the structured results obtained from SPARQL queries into human-readable, natural language responses. This process involves iterating over the query results, extracting relevant information, and formatting it into a coherent message for the user.

We initialize an empty string variable named response_message to store the final natural language response. This string will eventually contain the formatted information about the courses offered by the university.

Iterating Over Query Results:

We iterate over the query results obtained from the SPARQL query. Each result corresponds to a course offered by the university and contains information such as the course name and course number.

Extracting Course Information:

Within each iteration, we extract the course name and course number from the result. These values are retrieved from the results dictionary using appropriate keys, such as 'courseName' and 'courseNumber'.

Formatting Response Message:

For each course retrieved from the query results, we concatenate its name and number to the response_message string. We format the information in a user-friendly manner, typically as "Course Name : Course Number", and append it to the existing content of response_message.

Sending Natural Language Response:

Once all courses have been processed, we use the dispatcher.utter_message() function to send the constructed response_message to the user. This function is responsible for delivering the natural language response within the conversational interface.

Return Statement:

Finally, we return an empty list [] to signal the completion of the action execution.

By following this approach, we ensure that the raw data obtained from the SPARQL query is transformed into a meaningful and comprehensible format that can be easily understood by the user. This translation process enhances the user experience by providing relevant information in a human-friendly manner, facilitating effective communication between the user and the chatbot.

Example Workflow

As an example, workflow, consider the user query "What materials are recommended for Algorithm efficiency in PPS?":

Initial User Input: User asks about recommended materials for a specific topic.
Intent Recognition and Entity Extraction: Chatbot recognizes the intent "ListMaterialByTopicAndCourse" and extracts the topic "Algorithm efficiency" and course "PPS" from the user query.
Generated SPARQL Query: Chatbot constructs a SPARQL query to retrieve materials for the specified topic and course from the knowledge base.
Query Output: Chatbot executes the SPARQL query against the Fuseki server and receives a list of recommended materials.
Translation of Query Results: Chatbot translates the SPARQL query results into a fluent natural language response, listing the recommended materials for Algorithm efficiency in PPS.
Final Response: Chatbot provides the user with the translated response, showcasing the recommended materials for the specified topic and course.
This example illustrates how the chatbot interacts with the user, processes the query, retrieves information from the knowledge base, and communicates the results effectively in natural language. Similar workflows are implemented for other query types, covering a wide range of user inquiries related to courses, topics, materials, competencies, grades, students, and transcripts.

Questions and Sparql Queries

1. List all courses offered by [university]

   Initial User Input-

   List all courses offered by the university

   Intent recognition and entity extraction

   - intent: ListCoursesIntent

    - action: action_list_courses

   SparQl query

```
query = """
    PREFIX ex: <http://example.org/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?courseName ?courseNumber
    WHERE {
      ?course a ex:Course ;
              ex:CourseName ?courseName ;
              ex:CourseNumber ?courseNumber .
    }
    """
```

Output

/roboprof/sparql   JSON   Turtle

```
PREFIX ex: <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?courseName ?courseNumber
WHERE {
  ?course a ex:Course ;
          ex:CourseName ?courseName ;
          ex:CourseNumber ?courseNumber .
}
```

Table   Response   1981 results in 0.069 seconds   Simple view   Ellipse   Filter query results   Page size: 50

	courseName	courseNumber
1	PPS	6481
2	Intelligent Systems	6741
3	Traduction littéraire du français à l'anglais	301
4	Traduction littéraire de l'anglais au français	304
5	Traduction littéraire avancée : du français à l'anglais	401
6	Traduction littéraire avancée : de l'anglais au français	402
7	Social Economy & Sustainable Futures	450
8	Introduction to Quantum Theory	333

Natural language responses.

```
Your input ->  List all courses offered by the university
Courses offered by the university:
PPS : 6481
Intelligent Systems : 6741
Traduction littéraire du français à l’anglais : 301
Traduction littéraire de l’anglais au français : 304
Traduction littéraire avancée : du français à l’anglais : 401
Traduction littéraire avancée : de l’anglais au français : 402
Social Economy & Sustainable Futures : 450
Introduction to Quantum Theory : 333
Self-Managed Learning : 225
Fieldwork Practice : 435
```

2. In which courses is [topic] discussed?

Initial User Input-

In which courses is Hash table discussed?

Intent recognition and entity extraction

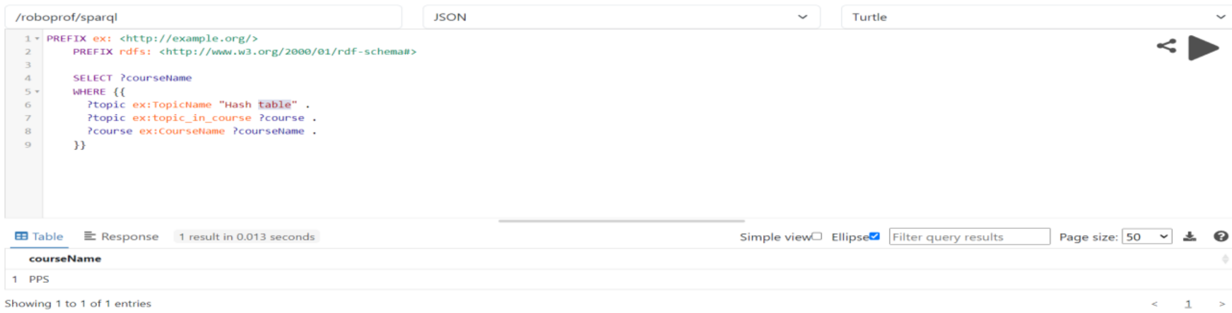
- intent: CoursesForTopicIntent
- action: action_courses_for_topic

SparQl query

```
query = """
    PREFIX ex: <http://example.org/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?courseName
    WHERE {
        ?topic ex:TopicName "%s" .
        ?topic ex:topic_in_course ?course .
        ?course ex:CourseName ?courseName .
    }
    """ % topic
```

Output



Natural language responses.

```
Your input ->  In which courses is Hash table discussed?
Course covering topic 'Hash table': PPS
```

3. Which [topics] are covered in [course] during [lecture number]?

Initial User Input-

Which topic are covered in Intelligent Systems during 6?

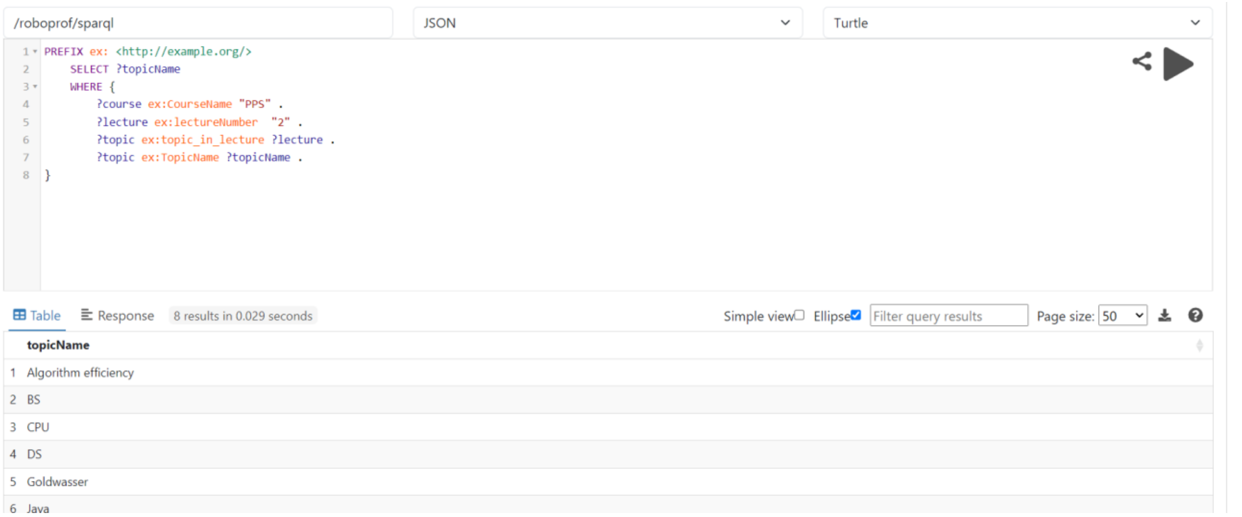
Intent recognition and entity extraction

- intent: TopicsCoveredInCourseIntent
- action: action_topics_covered_in_course

SparQl query

```
query = """
    PREFIX ex: <http://example.org/>
    SELECT ?topicName
    WHERE {
        ?course ex:CourseName "%s" .
        ?lecture ex:lectureNumber "%s" .
        ?topic ex:topic_in_lecture ?lecture .
        ?topic ex:TopicName ?topicName .
    }
    """ % (course, lecture_number)
```

Output



Natural language responses.

```
Your input ->  Which topic are covered in PPS during 2?
Topics covered in course 'PPS' at lecture 2: Algorithm efficiency, BS, CPU, DS, Goldwasser, Java, Wiley, Windows
```

4. List all [courses] offered by [university] within the [subject] (e.g., \COMP", \SOEN").

Initial User Input-

List all courses offered by concordia within the COMP?

Intent recognition and entity extraction

- intent: ListCousesInSubject
- action: action_list_courses_in_subject

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?courseName ?courseNumber
    WHERE {{
      ?course a ex:Course ;
              ex:CourseName ?courseName ;
              ex:CourseNumber ?courseNumber ;
              ex:Subject "{subject}" .
      ?university a ex:University ;
                  ex:UniversityName "Concordia University" .
    }}
    """
```

Output

/roboprof/sparql | JSON | Turtle

```
PREFIX ex: <http://example.org/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?courseName ?courseNumber
    WHERE {
      ?course a ex:Course ;
              ex:CourseName ?courseName ;
              ex:CourseNumber ?courseNumber ;
              ex:Subject "COMP" .
      ?university a ex:University ;
                  ex:UniversityName "Concordia University" .
    }
```

Table | Response | 2 results in 0.019 seconds | Simple view | Ellipse | Filter query results | Page size: 50

	courseName	courseNumber
1	PPS	6481
2	Intelligent Systems	6741

Natural language responses.

```
Your input ->  List all courses offered by concordia within the COMP?
Courses offered by the university within the subject:
PPS : 6481
Intelligent Systems : 6741
```

5. What [materials] (slides, readings) are recommended for [topic] in [course] [number]?

Initial User Input-

What readings should I focus on for Java in PPS?

Intent recognition and entity extraction

- intent: ListMaterialByTopicAndCourse
- action: action_list_materials

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    SELECT ?content
    WHERE {{
        ?topic ex:TopicName "{topic}" .
        ?topic ex:topic_in_lecture ?lecture .
        ?lecture ex:Content ?content .
    }}
    """
```

Output

/roboprof/sparql | JSON | Turtle

```
PREFIX ex: <http://example.org/>
    SELECT ?content
    WHERE {
        ?topic ex:TopicName "Java" .
        ?topic ex:topic_in_lecture ?lecture .
        ?lecture ex:Content ?content .
    }
```

Table | Response | 7 results in 0.016 seconds | Simple view | Ellipse | Filter query results | Page size: 50

	content
1	Slides: aa.pdf Worksheets: Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Natural language responses.

```
Your input ->  What readings should I focus on for Java in PPS?
Recommended materials for Java in PPS: Slides: aa.pdf
Worksheets:
Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
```

6. How many credits is [course] [number] worth?

Initial User Input-

How many credits is 6741 worth?

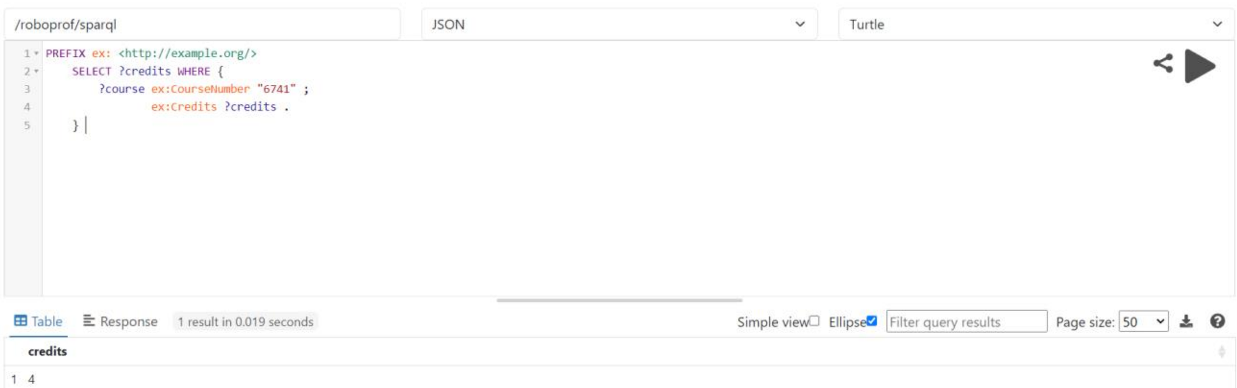
Intent recognition and entity extraction

- intent: ListCourseCredits
- action: action_list_course_credits

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    SELECT ?credits WHERE {{
        ?course ex:CourseNumber "{number_entity}" ;
                ex:Credits ?credits .
    }}
    """
```

Output



Natural language responses.

```
Your input ->  How many credits is 6741 worth?

The course 6741 is worth 4 credits.
```

7. For [course] [number], what additional resources (links to web pages) are available?

Initial User Input-

For 6481, what additional resources are available?

Intent recognition and entity extraction

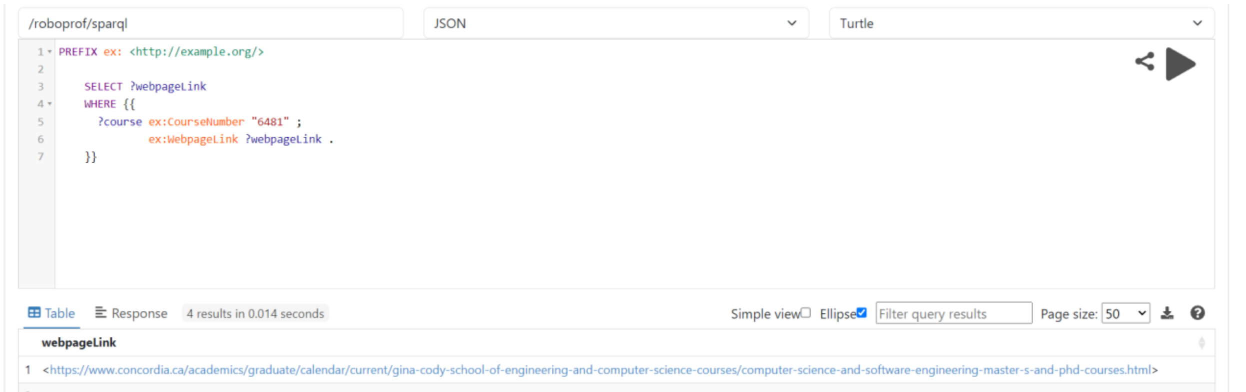
- intent: ListAdditionalResources
- action: action_list_additional_resources

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>

    SELECT ?webpageLink
    WHERE {{
      ?course ex:CourseNumber "{course_number}" ;
              ex:WebpageLink ?webpageLink .
    }}
    """
```

Output



Natural language responses.

```
The course 6741 is worth 4 credits.
Your input ->  For 6481, what additional resources are available?
Additional resources for course 6481 are available at the following link: https://www.concordia.ca/academics/graduate/calendar/current/gina-cody-school-of-engineering-and-computer-science-courses/computer-scienc
e-and-software-engineering-master-s-and-phd-courses.html
```

8. Detail the content (slides, worksheets, readings) available for [lecture number] in [course] [number].

Initial User Input-

Detail the content available for lecture 3 in course 6481

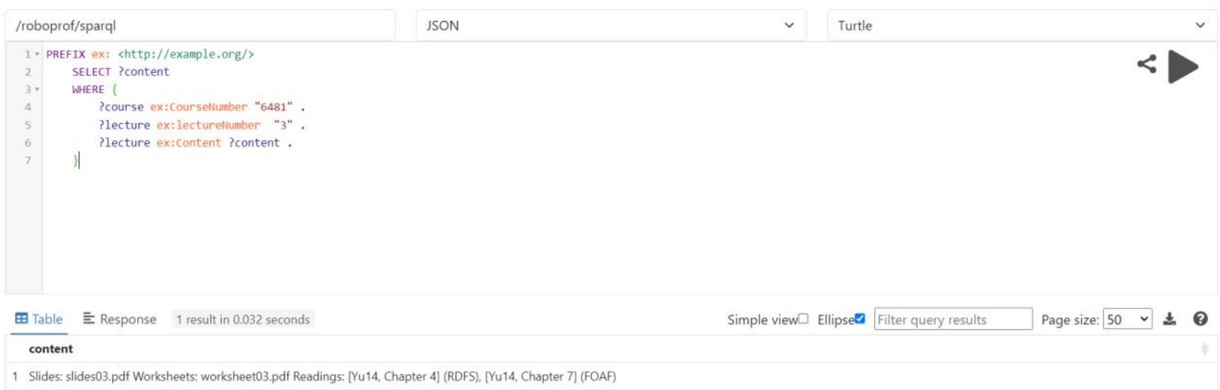
Intent recognition and entity extraction

- intent: ListContentDetails
- action: action_list_content_details

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    SELECT ?content
    WHERE {{
        ?course ex:CourseNumber "{course_number}" .
        ?lecture ex:lectureNumber  "{lecture_number}" .
        ?lecture ex:Content ?content .
    }}
    """
```

Output



Natural language responses.

```
Your input ->  Detail the content available for lecture 3 in course 6481
Content available for lecture 3 in course 6481:
Slides: slides03.pdf
Worksheets: worksheet03.pdf
Readings: [Yu14, Chapter 4] (RDFS),  [Yu14, Chapter 7] (FOAF)
```

9. What reading materials are recommended for studying [topic] in [course]?

Initial User Input-

What reading materials are recommended for studying Hash table in PPS?

Intent recognition and entity extraction

- intent: ListReadingMaterials
- action: action_reading_materials

SparQl query

```
PREFIX ex: <http://example.org/>
SELECT ?content
WHERE {{
    ?course ex:CourseName "{course}" .
    ?topic ex:topic_in_lecture ?lecture .
    ?lecture ex:Content ?content .
    FILTER(contains(?content, "Readings"))
}}
```

Output

```
PREFIX ex: <http://example.org/>
SELECT DISTINCT ?content
WHERE {
    ?course ex:CourseName "PPS" .
    ?topic ex:topic_in_lecture ?lecture .
    ?lecture ex:Content ?content .
    FILTER(contains(?content, "Readings"))
}
```

Table   Response   14 results in 0.056 seconds   Simple view   Ellipse   Filter query results   Page size: 50

	content
1	Slides: aa.pdf Worksheets: Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
2	Slides: C:\Users\anubh\OneDrive\Desktop\COMP 6481\aa.pdf Worksheets: /Users/saurabh/Documents/Courses/All terms/Fall 23/COMP 6481-PPS/Tutorial_2.pdf Readings: Textbook Data Structures and Algorithms in Jav...
3	Slides: chap7-Inheritance.pdf Worksheets: Readings:
4	Slides: C:\Users\anubh\OneDrive\Desktop\COMP 6481\chap7-Inheritance.pdf Worksheets: /Users/saurabh/Documents/Courses/All terms/Fall 23/COMP 6481-PPS/Tutorial_1.pdf Readings:
5	Slides: slides01.pdf Worksheets: worksheet01.pdf Readings: Introduction to Artificial Intelligence, https://plato.stanford.edu/entries/artificial-intelligence/, Eliza, https://en.wikipedia.org/wiki/ELIZA
6	Slides: C:\Users\anubh\OneDrive\Desktop\COMP 6741\slides01.pdf Worksheets: C:\Users\anubh\OneDrive\Desktop\COMP 6741\worksheet01.pdf Readings: Introduction to Artificial Intelligence, https://plato.stanford.ed...
7	Slides: Recursion.pdf Worksheets: Readings: https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/
8	Slides: C:\Users\anubh\OneDrive\Desktop\COMP 6481\Recursion.pdf Worksheets: /Users/saurabh/Documents/Courses/All terms/Fall 23/COMP 6481-PPS/COMP6481_TUT_05_Reg.pdf Readings: https://www.geeksforgee...
9	Slides: /Users/saurabh/Documents/Courses/All terms/Fall 23/COMP 6481-PPS/Lectures Slides/Trees/trees.pdf Worksheets: /Users/saurabh/Documents/Courses/All terms/Fall 23/COMP 6481-PPS/Tutorial_10.pdf Readings...

Natural language responses.

```
Your input ->  What reading materials are recommended for studying Hash table in PPS?
Reading materials recommended for studying Hash table in PPS:
- Slides: aa.pdf
Worksheets:
Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
- Slides: C:\Users\anubh\OneDrive\Desktop\COMP 6481\aa.pdf
Worksheets: /Users/saurabh/Documents/Courses/All terms/Fall 23/COMP 6481-PPS/Tutorial_2.pdf
Readings: Textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
- Slides: aa.pdf
```

10. What competencies [topics] does a student gain after completing [course] [number]?

Initial User Input-

- intent: ListCompetencies
- action: action_list_competencies

What competencies does a student gain after completing COMP 6481?

Intent recognition and entity extraction

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    SELECT ?competency
    WHERE {{
      ?student ex:completedCourse "{course_number}" ;
               ex:competentIn ?competency .
    }}
    """
```

Output

/roboprof/sparql | JSON | Turtle

```
PREFIX ex: <http://example.org/>

  SELECT ?competency
  WHERE {{
    ?student ex:completedCourse "COMP 6481" ;
             ex:competentIn ?competency .
  }}
```

Table | Response | 7 results in 0.014 seconds | Simple view | Ellipse | Filter query results | Page size: 50

	competency
1	Hash table
2	Run Time Errors
3	Binary Trees
4	Depth First Search (DFS)
5	Algorithm efficiency

Natural language responses.

```
Your input -> What competencies does a student gain after completing COMP 6481?
Competencies gained after completing COMP 6481:
- Hash table
- Run Time Errors
- Binary Trees
- Depth First Search (DFS)
- Algorithm efficiency
- Graphs
- Binary Trees
```

11. What grades did [student] achieve in [course] [number]?

Initial User Input-

Show me the grades of Saurabh Sharma in COMP 6481

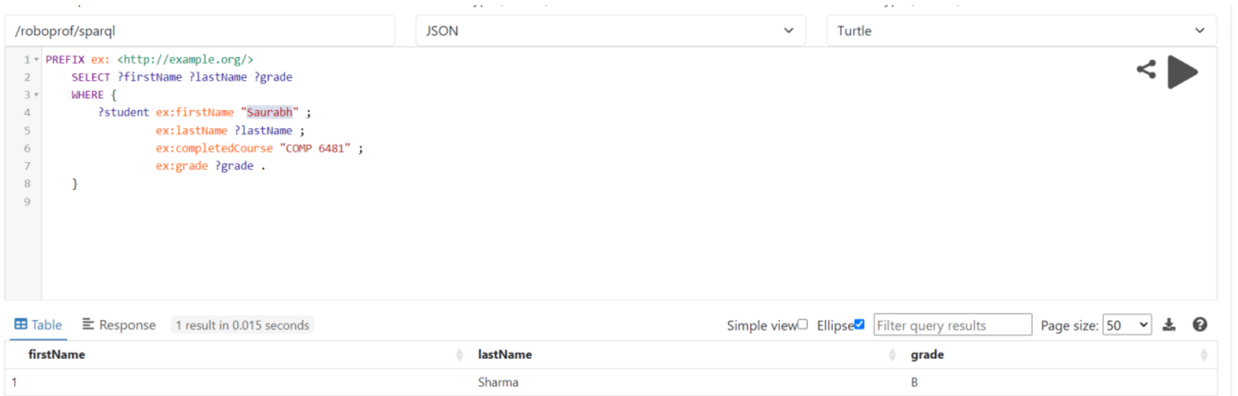
Intent recognition and entity extraction

- intent: ListGrades
- action: action_list_grades

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    SELECT ?firstName ?lastName ?grade
    WHERE {{
        ?student ex:firstName ?firstName ;
                 ex:lastName ?lastName ;
                 ex:completedCourse "{course_entity}" ;
                 ex:grade ?grade .
        FILTER(?firstName = "{first_name}")
    }}
    """
```

Output



Natural language responses.

```
Your input ->  Show me the grades of Saurabh Sharma in COMP 6481
Saurabh Sharma achieved B in COMP 6481
```

12. Which [students] have completed [course] [number]?

Initial User Input-

List students who completed COMP 6481

Intent recognition and entity extraction

- intent: ListStudentsByCompletedCourse
- action: action_list_students_by_completed_course

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>

    SELECT ?firstName ?lastName
    WHERE {{
      ?student ex:firstName ?firstName ;
               ex:lastName ?lastName ;
               ex:completedCourse "{completed_course}" .
    }}
    """
```

Output

/roboprof/sparql | JSON | Turtle

```
PREFIX ex: <http://example.org/>

    SELECT ?firstName ?lastName
    WHERE {
      ?cour ex:firstName ?firstName ;
               ex:lastName ?lastName ;
               ex:completedCourse "COMP 6481" .
    }
```

Table | Response | 8 results in 0.014 seconds | Simple view | Ellipse | Filter query results | Pa

	firstName	lastName
4	Yash	Changrani
3	Anubhav	Mahajan
1	Saurabh	Sharma

Natural language responses.

```
Your input ->  List students who completed COMP 6481
Students who have completed course COMP 6481:
Saurabh  Sharma
Saurabh Sharma
Anubhav Mahajan
Yash Changrani
Arshdeep Singh
Prashant Tilekar
Rishabh Tayal
Karan Singla
```

13. Print a transcript for a [student], listing all the course taken with their grades.

Initial User Input-

Can you provide the transcript for a Saurabh Sharma?

Intent recognition and entity extraction

- intent: ListStudentTranscript
- action: action_list_students_transcript

SparQl query

```
    query = f"""
        PREFIX ex: <http://example.org/>
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

        SELECT ?completedCourse ?grade
        WHERE {{
          ?student rdf:type ex:Student ;
                   ex:firstName "{first_name}" ;
                   ex:completedCourse ?completedCourse ;
                   ex:grade ?grade .
        }}
        """
```

Output

/roboprof/sparql | JSON | Turtle

```
PREFIX ex: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?completedCourse ?grade
WHERE {{
  ?student rdf:type ex:Student ;
           ex:firstName "Saurabh" ;
           ex:completedCourse ?completedCourse ;
           ex:grade ?grade .
}}
```

Table | Response | 1 result in 0.013 seconds | Simple view | Ellipse | Filter query results | Page size: 50

	completedCourse	grade
1	COMP 6481	B

Natural language responses.

```
Your input ->  Can you provide the transcript for a Saurabh Sharma?

Transcript for student Saurabh Sharma:
Course: COMP 6481, Grade: B
```

14. What is the <course> about?"

Initial User Input-

What is the Intelligent Systems about?

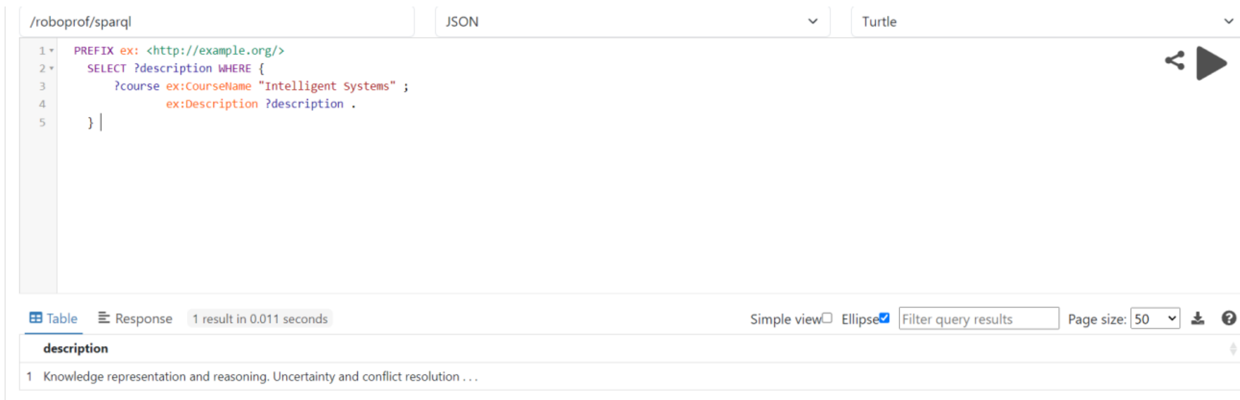
Intent recognition and entity extraction

- intent: ShowDescription
- action: action_show_description

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>
    SELECT ?description WHERE {{
        ?course ex:CourseName "{course}" ;
                ex:Description ?description .
    }}
    """
```

Output



Natural language responses.

```
Your input ->  What is the Intelligent Systems about?
The course Intelligent Systems is about: Knowledge representation and reasoning. Uncertainty and conflict resolution . . .
```

15. Which course events cover <Topic>?"

Initial User Input-

List topics for COMP_6481

Intent recognition and entity extraction

- intent: ListTopicByCourseEvent
- action: action_list_topic_by_course_event

SparQl query

```
query = f"""
    PREFIX ex: <http://example.org/>

    SELECT DISTINCT ?dbpediaLink ?topicName ?provenance ?topicInLecture
    WHERE {{
      ?topic ex:topic_in_course ex:{formatted_course_code} ;
             ex:dbpediaLink ?dbpediaLink ;
             ex:TopicName ?topicName ;
             ex:provenance ?provenance ;
             ex:topic_in_lecture ?topicInLecture .
    }}
    """
```

Output

```
PREFIX ex: <http://example.org/>

SELECT DISTINCT ?dbpediaLink ?topicName ?provenance ?topicInLecture
WHERE {
  ?topic ex:topic_in_course ex:COMP_6481 ;
         ex:dbpediaLink ?dbpediaLink ;
         ex:TopicName ?topicName ;
         ex:provenance ?provenance ;
         ex:topic_in_lecture ?topicInLecture .
}
```

Table   Response   74 results in 0.017 seconds   Simple view   Ellipse   Filter query results   Page size: 50

	dbpediaLink	topicName	provenance	topicInLecture
1	<https://dbpedia.org/page/Category:Algorithms>	Algorithm efficiency	Course outline	<http://example.org/Algorithm_analysis>
2	<https://dbpedia.org/page/Hash_table>	Hash table	Course outline	<http://example.org/Hashmap>
3	<http://dbpedia.org/resource/Recursion>	Recursion	lecture	<http://example.org/Recursion>
4	<http://dbpedia.org/resource/Recursion>	Recursion	lecture	<http://example.org/Hashmap_Tutorial>
5	<http://dbpedia.org/resource/Recursion>	Recursion	tutorial	<http://example.org/Recursion>
6	<http://dbpedia.org/resource/Recursion>	Recursion	tutorial	<http://example.org/Hashmap_Tutorial>

Natural language responses.

```
Your input ->  List topics for COMP_6481
Topic: Algorithm efficiency
DBpedia Link: https://dbpedia.org/page/Category:Algorithms
Provenance: Course outline
Lecture Topic URI: http://example.org/Algorithm_analysis
Topic: Hash table
DBpedia Link: https://dbpedia.org/page/Hash_table
Provenance: Course outline
Lecture Topic URI: http://example.org/Hashmap
Topic: Recursion
DBpedia Link: http://dbpedia.org/resource/Recursion
Provenance: lecture
Lecture Topic URI: http://example.org/Recursion
Topic: Recursion
DBpedia Link: http://dbpedia.org/resource/Recursion
Provenance: lecture
Lecture Topic URI: http://example.org/Hashmap_Tutorial
Topic: Recursion
DBpedia Link: http://dbpedia.org/resource/Recursion
Provenance: tutorial
Lecture Topic URI: http://example.org/Recursion
Topic: Recursion
DBpedia Link: http://dbpedia.org/resource/Recursion
Provenance: tutorial
Lecture Topic URI: http://example.org/Hashmap_Tutorial
Topic: Run Time Errors
DBpedia Link: https://www.wikidata.org/wiki/Q15825362
Provenance: Course outline
Lecture Topic URI: http://example.org/Exceptions
Topic: Run Time Errors
DBpedia Link: https://www.wikidata.org/wiki/Q15825362
Provenance: Course outline
Lecture Topic URI: http://example.org/Exception_Handelling
```

16. Which topics are covered in <course event>?"

Initial User Input-

Show courses with Text mining topics

Intent recognition and entity extraction

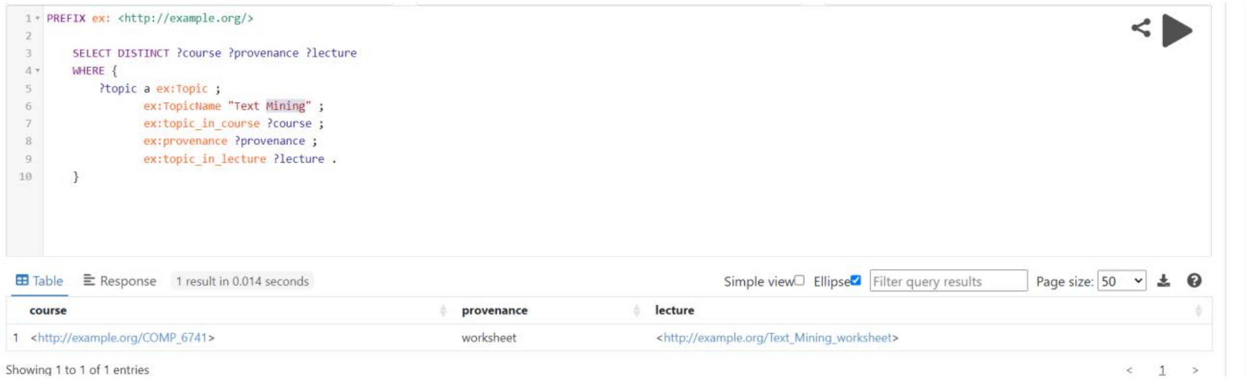
- intent: ListTopicContent
- action: action_list_topic_content

SparQl query

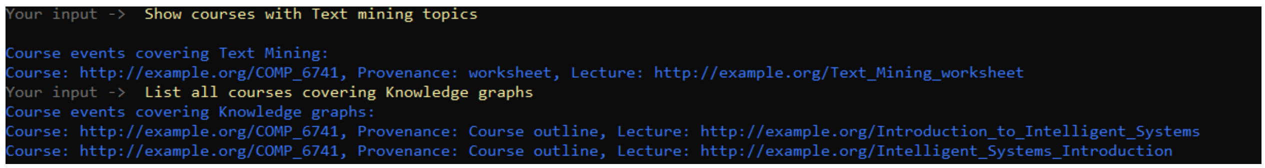
```
query = f"""
    PREFIX ex: <http://example.org/>

    SELECT DISTINCT ?course ?provenance ?lecture
    WHERE {{
        ?topic a ex:Topic ;
               ex:TopicName "{topic}" ;
               ex:topic_in_course ?course ;
               ex:provenance ?provenance ;
               ex:topic_in_lecture ?lecture .
    }}
    """
```

Output



Natural language responses.



Additional on converting sparql query result to natural language:
{SPARQL query results are translated into natural language answers using predefined templates or dynamic message generation. Templates are designed to structure the output in a human-readable format, incorporating variables or placeholders to insert specific values retrieved from the query results. Alternatively, natural language generation (NLG) techniques can be employed to dynamically generate responses based on the content of the query results, ensuring fluent and coherent communication with the user.}

Conclusion

Total Number of Generated Triples	20882

The final phase, as outlined in this report, was dedicated to lay the groundwork for our project. Through meticulous parsing of datasets and leveraging Semantic Web technologies, we successfully built a structured knowledge graph encompassing diverse educational domains.

Moving forward, our focus shifts towards refining this foundation and incorporating additional enhancements. The subsequent phase will involve further enriching the knowledge base, implementing advanced features, and developing a natural language processing interface. These endeavors aim to enhance the usability and functionality of our system, facilitating seamless access to educational resources and insights.

By integrating these components into a cohesive system, we aspire to create a robust platform. Through continuous iteration and refinement, we remain committed to pushing the boundaries of knowledge representation and discovery, empowering users with valuable insights and facilitating meaningful interactions with educational data.