ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF ELECTRICAL, ELECTRONIC, AND INFORMATION ENGINEERING "GUGLIELMO MARCONI" - DEI

SECOND CYCLE DEGREE

ELECTRONICS FOR INTELLIGENT SYSTEMS, BIG DATA, AND INTERNET OF THINGS

INTEGRATION OF AI WITH KNOWLEDGE GRAPHS FOR SMART AGRICULTURE APPLICATIONS

Supervisor

Prof. Luca Roffia

Co-Supervisor

Gregorio Monari

Defended by

Bita Alaee Majelan

GRADUATION SESSION OCTOBER 2025

ACADEMIC YEAR 2024/2025

Abstract

This thesis explores the integration of large language models (LLMs) with semantic web technologies to enhance weather-aware decision-making in smart agriculture. The study focuses on bridging natural language input with structured environmental data by converting user queries into SPARQL using a GPT-based LLM. Historical weather data, including daily minimum, maximum, and average air temperatures, was collected from two publicly available weather APIs, demonstrating the role of semantic technologies in achieving data interoperability. These datasets were semantically modeled using the SOSA ontology and stored in a SPARQL Event Processing Architecture (SEPA) knowledge graph, enabling real-time querying over RDF triples.

To evaluate the effectiveness of this approach, a prototype system was developed to process user questions in natural language and return weather observations by executing the corresponding SPARQL queries. The system was tested using a one-year dataset from both APIs for the De Bilt region (Netherlands), allowing for validation of query results and analysis of vocabulary alignment and model consistency. The findings demonstrate the potential of combining LLMs with semantic graph infrastructures to improve accessibility, usability, and interpretability of environmental data in agricultural applications.

Acknowledgement

I would like to extend my heartfelt gratitude to my supervisor, Professor Luca Roffia, for his unwavering support and insightful feedback throughout my thesis and internship journey. His steadfast commitment to academic excellence and meticulous attention to detail have profoundly shaped the development of this thesis internship.

I would also like to acknowledge my teammates at VAIMEE during this internship for their invaluable assistance and the engaging discussions that fueled my inspiration throughout my academic endeavors. Their encouragement has been a fundamental aspect of my research experience.

Furthermore, I am deeply appreciative of my family and friends, who provided both meaningful distractions when necessary and the motivation required to persevere through challenging times.

Table of Contents

Abstract i
Acknowledgement ii
List of Figures iv
List of Tables v
1 Introduction 1
2 Technologies and Standards 2
2.1 RDF (Resource Description Framework) 2
2.2 SPARQL 1.1 (SPARQL Protocol and RDF Query Language) 2
2.2.1 Query 2
2.2.2 Update 2
2.2.3 Federated Query 2
2.3 SEPA (SPARQL Event Processing Architecture) 3
2.4 Agorà: The Agritech Oracle 4
2.5 SOSA Ontology 4
2.6 JSAP (JSON SPARQL Application Profile) 5
2.7 SEPY Python Client 5
2.8 LLM-Based SPARQL Generation using OpenAI GPT API 5
2.9 Data APIs: KNMI and OpenWeather 6
3 System Design and Implementation 7
3.1 Overall System Architecture 7
3.2 Data sources 8
3.3 AI Module 11
3.4 Knowledge Graph Module 12
4 Results 14
4.1 Experimental setup 14
4.2 Test scenarios 14
4.3 Discussion 16
5 Conclusion and Future Work 17
References 18

List of Figures

Figure 2.1- Generic architecture for federated SPARQL query processing 3
Figure 2.2- Overview of SEPA architecture and publish/subscribe flows 4
Figure 3.1- System architecture integrating weather data and LLM-based query generation 7
Figure 3.2- Comparison of temperatures retrieved from KNMI and OpenWeather 9
Figure 3.3- Map of KNMI weather stations across the Netherlands 9
Figure 3.4- KNMI station metadata 10
Figure 3.5- Core structure of the SOSA ontology 13
Figure 4.1- Output Returned by GPT-4 from OpenAI Response API 14
Figure 4.2- Output Returned by GPT-4o from OpenAI Response API 15

List of Tables

Table 4.1- Summary of Temperature Data Sources 14
Table 4.2- Comparison of SPARQL Generation by Different GPT Models 15

1 Introduction

Precision agriculture is increasingly relying on data-driven decision support systems to optimize resource utilization, enhance crop yields, and adapt to dynamic environmental conditions. One of the most critical components of these systems is accurate, real-time weather information, which directly influences key agricultural decisions, such as irrigation scheduling, pest management, and crop performance. However, much of this data remains either unstructured or fragmented across different platforms and is inaccessible to end-users without specialized technical expertise.

To bridge this gap, semantic web technologies offer a solution by enabling machine-readable, interoperable environmental data that can be modeled, queried, and understood across different systems. One such example is Agorà, an innovative AgriTech solution developed by VAIMEE. Agorà acts as a Digital Twin designed to predict the water needs of fields worldwide, leveraging semantic interoperability to ensure better decision-making and reduce resource waste, specifically in water management. Agorà utilizes data from various sources, including weather stations, soil types, and crop-specific information, to provide real-time irrigation predictions for farmers [1].

In parallel, the rapid advancements in large language models (LLMs), particularly GPT-4 and other transformer-based models, present new opportunities to interface with structured knowledge bases and support natural language queries. These models bridge the gap between technical data and user-friendly interfaces, enabling non-technical users to interact with complex datasets more intuitively. By combining these two technologies, semantic web data models and AI-driven natural language interfaces, this thesis investigates an approach to weather-aware decision-making in smart agriculture.

This work focuses on integrating historical weather data from the KNMI EDR API [2] and OpenWeather History API [3] with Agorà's framework to enable easy querying of weather observations using natural language. The data, including daily minimum, maximum, and mean air temperatures for the De Bilt region, is semantically modeled using the SOSA ontology and stored in a SPARQL Event Processing Architecture (SEPA) endpoint. A GPT-based module was developed to automatically translate user questions into SPARQL queries, which are then executed against the SEPA knowledge graph to retrieve the relevant weather data.

The feasibility of this approach was evaluated by examining the syntactic correctness of the queries generated by the LLM, ensuring they aligned with the SOSA and KNMI vocabularies. Additionally, the system's ability to return accurate and actionable weather data was assessed. The experimental results demonstrate the potential of combining LLMs and semantic data models for agricultural decision support, although challenges remain in query consistency and vocabulary alignment. This work lays the foundation for integrating smart agriculture tools with more advanced AI models, aiming to enhance the accessibility of complex weather data for farmers and agronomists.

2 Technologies and Standards

This chapter summarizes the main technologies and tools used, with a focus on the features relevant to the work carried out.

2.1 RDF (Resource Description Framework)

RDF is a standard data model developed by the W3C for encoding, exchanging, and interlinking structured information on the web. In RDF, data is expressed in the form of triples, each consisting of a *subject*, *predicate*, and *object*. These triples represent directed relationships, allowing data to be stored and queried as a graph [4].

Each node in the RDF graph may represent an entity (identified by a URI), a literal value (such as a number, date, or string), or a blank node. The graph structure enables semantic interoperability across heterogeneous systems, supporting integration of distributed datasets. In this thesis, RDF serves as the foundational data model for representing meteorological observations retrieved from KNMI and OpenWeather sources, which are annotated using the SOSA ontology.

2.2 SPARQL 1.1 (SPARQL Protocol and RDF Query Language)

SPARQL is the query language used for retrieving and manipulating RDF data. SPARQL 1.1 supports querying multiple RDF datasets simultaneously, as well as performing updates to RDF graphs via dedicated operations. The protocol defines a standardized transport mechanism based on HTTP, enabling interaction with remote RDF stores [5].

2.2.1 Query

SPARQL queries typically follow a SELECT pattern, where variables are used to extract matching triples from RDF datasets. Queries can include complex filters, joins, and aggregations. In this thesis, SPARQL is used to retrieve temperature data stored in a SEPA triple store, with queries either written manually or generated automatically by a GPT-based LLM [6].

2.2.2 Update

SPARQL 1.1 Update is an extension of SPARQL with commands for managing RDF data, offering a comprehensive set of operations for updating RDF graphs [7], which includes:

- `INSERT DATA`: Explicitly adds new triples.
- `DELETE DATA`: Removes existing triples.
- `DELETE WHERE`: Eliminates triples based on pattern matching.
- `MODIFY`: Simultaneous insertion and deletion of triples using a WHERE clause.
- `CLEAR, DROP, COPY, and ADD`: Enable management of entire graphs.

These update operations are essential for incorporating new weather observations into the SEPA backend, as illustrated in the JSAP update templates discussed in Section 3.2.

2.2.3 Federated Query

SPARQL 1.1 introduces the capability for federated queries through the SERVICE keyword, enabling a query to access data from multiple remote SPARQL endpoints in a single execution. This feature facilitates the integration of diverse RDF datasets spread across the web, eliminating the need for centralized storage. A common application of this functionality is the combination of weather data from various APIs or knowledge graphs by distributing segments of the query to their corresponding

endpoints. Although the current implementation in this thesis does not directly employ federated querying, the architecture is designed to support this method. Future enhancements, particularly those involving the incorporation of external agricultural or geospatial datasets, could utilize this capability to enhance the system's semantic reasoning and overall coverage [8].

Figure 2.1 Illustrates a generic architecture for federated query processing systems, highlighting the flow of SPARQL queries from decomposition to execution across multiple data sources [9].

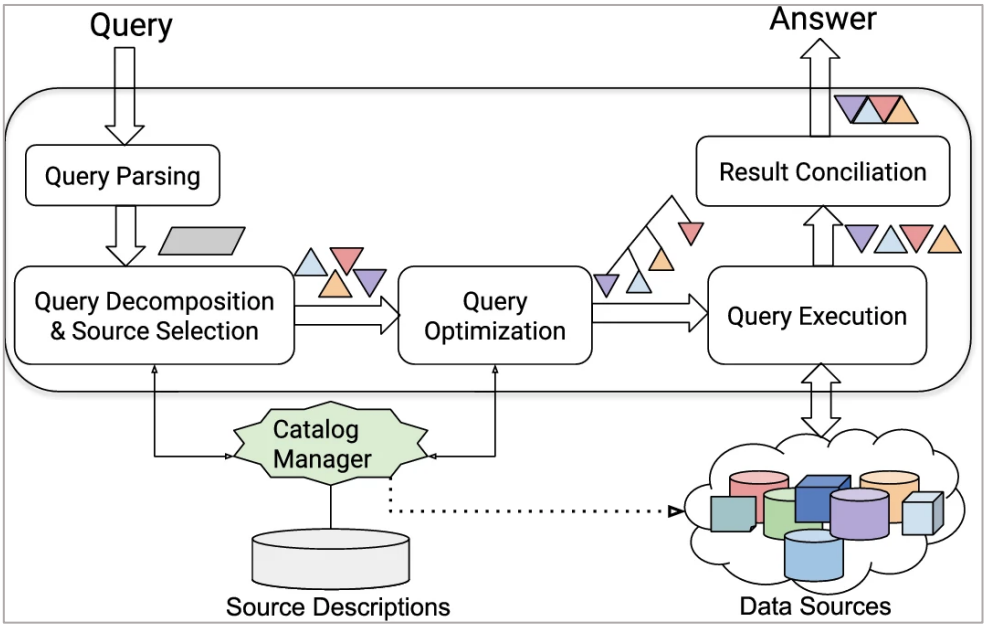

Figure 2.1- Generic architecture for federated SPARQL query processing

2.3 SEPA (SPARQL Event Processing Architecture)

SEPA is designed to support the development of dynamic linked data applications, where real-time updates and synchronization between distributed clients are essential. It extends the classical Web of Data by enabling not only queries and updates via SPARQL 1.1, but also subscriptions to data changes using a `publish`/`subscribe` paradigm.

The core of SEPA is the SEPA broker, which implements an extension of the SPARQL 1.1 protocol called the SPARQL Secure Event (SE) Protocol. This extension introduces two additional primitives, `Subscribe` and `Notify`, enabling clients to react in real time to changes in RDF data. A SEPA client can subscribe to a SPARQL query. Whenever the result set of that query changes (due to updates from other clients), a notification is automatically pushed to the subscriber.

This event-driven architecture facilitates the creation of reactive, loosely coupled systems on top of RDF graphs. SEPA also leverages JSON SPARQL Application Profiles (JSAPs) for managing configuration, authentication, and semantic interactions in a modular and extensible way.

Figure 2.2 Illustrates the interaction between SEPA agents (producers, consumers, and aggregators), the broker, and the underlying RDF datasets [10].

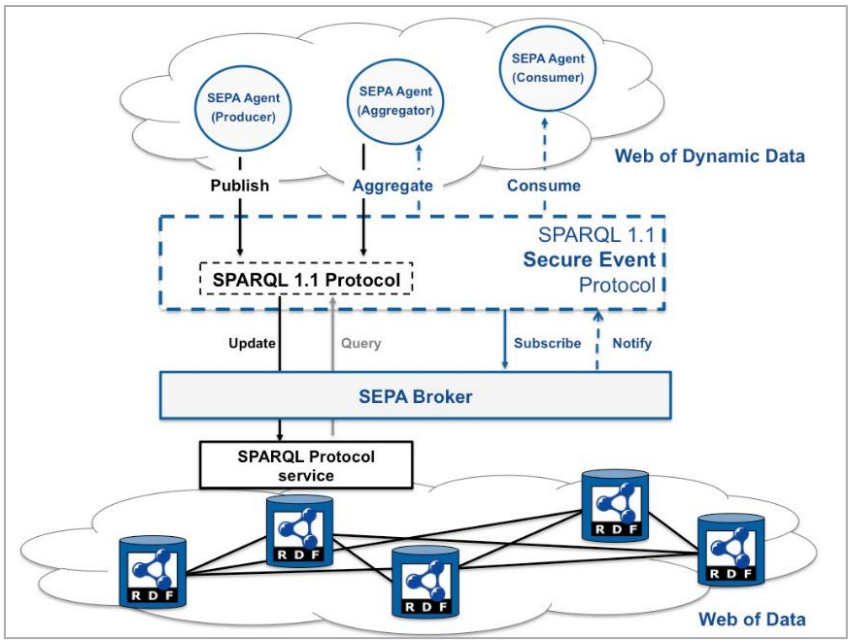

Figure 2.2 - Overview of SEPA architecture and publish/subscribe flows

2.4 Agorà: The Agritech Oracle

Agorà is an innovative AgriTech solution created by VAIMEE. This Digital Twin is designed to accurately forecast the water requirements of fields, crops, and various soil types worldwide, utilizing semantic interoperability to minimize water waste. Users can receive irrigation recommendations for the following day tailored to a specific unit, which is defined as a combination of soil type, crop type, and meteorological station. Agorà utilizes the CRITERIA model to produce its forecasts. This one-dimensional agro-hydrological model simulates soil water movement, crop growth, root water uptake, and irrigation water needs [1][11].

2.5 SOSA Ontology

The Sensor, Observation, Sample, and Actuator (SOSA) ontology is a lightweight, modular ontology for modeling observations, sensing devices, procedures, and the resulting data. It is designed for use cases where interoperability and ease of integration are essential, such as the Internet of Things, environmental monitoring, and smart agriculture. SOSA provides a core set of classes (e.g., `sosa:Observation`, `sosa:Sensor`, `sosa:FeatureOfInterest`, `sosa:Result`) and properties (e.g., `sosa:hasResult`, `sosa:observedProperty`, `sosa:resultTime`) that make it suitable for describing observational data in RDF [12].

In this thesis, SOSA was adopted as the primary ontology for representing weather observations, enabling semantic alignment across different data sources. The ontology's compatibility with the W3C PROV-O model also facilitates the traceability and provenance tracking of observations, which is crucial in critical domains such as precision farming. A visual summary of SOSA's structure is provided later in Section 3.4 to contextualize its use in modeling temperature values.

2.6 JSAP (JSON SPARQL Application Profile)

JSON SPARQL Application Profile (JSAP) is a configuration format developed within the SEPA ecosystem that describes all SPARQL endpoints and their associated operations in a single, machine-readable JSON file. Each JSAP file specifies named SPARQL queries and updates, namespace prefixes, forced bindings, and graph URIs, allowing client applications to interact with RDF graphs in a modular and reusable way.

In this thesis, JSAP files were used to insert weather observations into the SEPA triple store (`INSERT_OBSERVATION`) and to retrieve specific records based on query conditions (`GET_OBSERVATIONS_FOR_STATION`). JSAP enabled clear separation between semantic logic and API implementation, simplifying integration with Python scripts and the SEPY library. A representative JSAP configuration is later discussed in Section 3.4, where its structure and practical use are demonstrated [13].

2.7 SEPY Python Client

To interact programmatically with SEPA, this project uses SEPY, a Python client library developed for loading JSAP files and executing SPARQL queries, updates, and subscriptions. The `SAPObject` class parses the JSAP and manages SPARQL templates, while the `SEPA` class establishes connections to the query and update endpoints defined in the configuration.

This library simplifies integration between Python scripts and the SEPA broker, allowing automation of data insertion, live monitoring, and query execution. It was used in both the data ingestion phase (e.g., uploading temperature observations) and in the LLM query interface, where GPT-generated SPARQL queries are executed dynamically [14].

2.8 LLM-Based SPARQL Generation using OpenAI GPT API

Large language models (LLMs), such as GPT-4o, have showcased impressive abilities in comprehending and generating natural language. Trained on extensive text corpora, these models can execute complex reasoning tasks, produce contextually relevant responses, and even translate human inquiries into structured query languages. In the realm of semantic web technologies, LLMs provide an innovative interface for engaging with RDF-based knowledge graphs through natural language. This thesis leverages these capabilities to bridge the gap between human questions and SPARQL queries by integrating OpenAI's GPT API into the data pipeline.

The OpenAI GPT API provides programmatic access to large language models that can understand natural language input and generate structured output [15]. In this project, the API is utilized to convert user questions (for example, "*What was the maximum temperature in De Bilt on June 6, 2025?*") into corresponding SPARQL queries using controlled prompting.

The prompt includes context, prefix definitions for the SOSA and KNMI vocabularies, and specific task instructions. The model used is GPT-4o, selected for its ideal balance between speed and syntax-awareness.

This integration enables natural language interfaces to interact with RDF knowledge graphs, acting as a semantic layer that simplifies SPARQL complexity for end-users. The generated queries are validated and subsequently executed via SEPY against the SEPA knowledge graph.

2.9 Data APIs: KNMI and OpenWeather

Meteorological data for this study were retrieved from two publicly available APIs. The KNMI EDR API provides historical weather observations in JSON format, offering high-resolution data for the Netherlands meteorological stations. In parallel, the OpenWeather History API supplies globally available weather records with flexible date range queries and simplified JSON formatting.

These sources were used to collect daily mean (TG), minimum (TN), and maximum (TX) air temperature values for the De Bilt region. The retrieved data was parsed and semantically annotated by mapping temperature observations to relevant concepts in the SOSA ontology. Each observation was then inserted into a SEPA (SPARQL Event Processing Architecture) knowledge graph as RDF triples using the SEPY client and JSAP configuration.

A comparative analysis of the datasets, including differences in temporal resolution, value discrepancies, and data availability, is presented in subsequent chapters.

3 System Design and Implementation

3.1 Overall System Architecture

The system architecture proposed in this thesis draws conceptual inspiration from the Agorà platform. In Agorà, weather information serves as a critical input to a broader reasoning component known as the criteria model, which evaluates user-defined objectives such as yield optimization or resource efficiency.

In the present work, the criteria model has been intentionally excluded to simplify the system design and focus on the technical integration challenges associated with external weather APIs, semantic knowledge graphs, and large language models. The central objective is to demonstrate the feasibility of linking natural language user interfaces with RDF-based environmental datasets through LLM-generated SPARQL queries. This foundational architecture can be extended in future iterations by incorporating domain reasoning layers, such as Agorà's criteria model, to support fully automated decision-making.

Figure 3.1 Illustrates the overall system architecture. The pipeline begins with the retrieval of daily minimum (TN), maximum (TX), and mean (TG) air temperature values from two publicly available sources: the KNMI EDR API and the OpenWeather History API. Data acquisition is handled through Python scripts that manage endpoint authentication, parameterized HTTP requests, and JSON parsing.

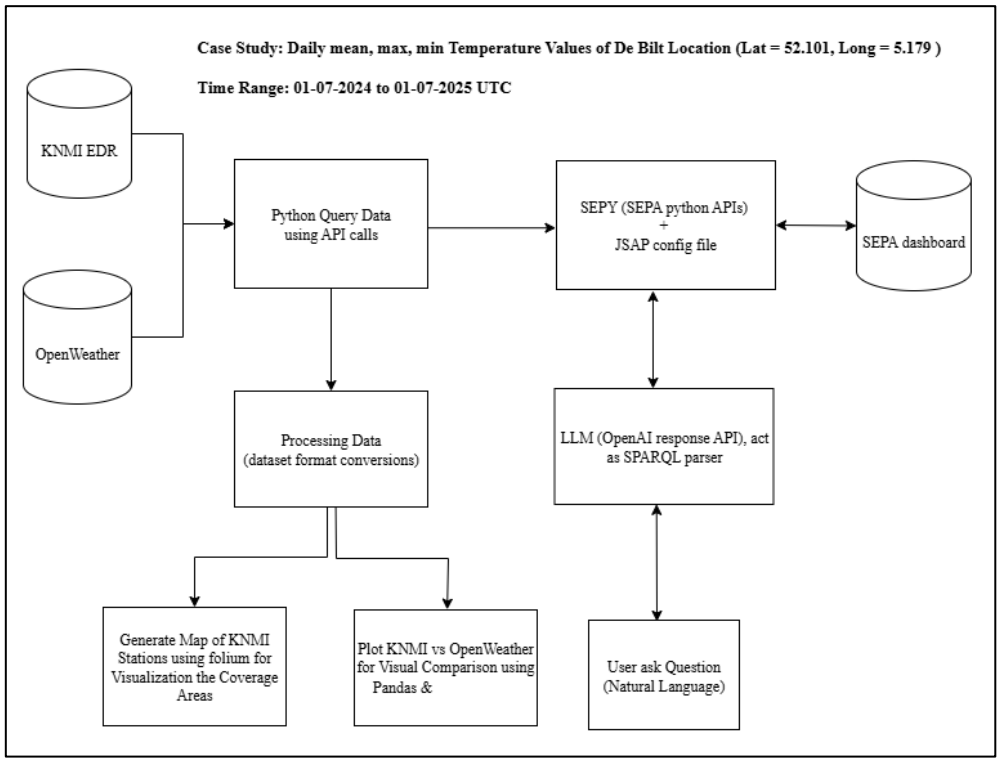

Figure 3.1- System architecture integrating weather data, semantic modeling, and LLM-based query generation

The retrieved datasets are then cleaned and harmonized into a unified format by aligning timestamp conventions and standardizing temperature units. Once processed, the data is semantically annotated using the SOSA ontology and stored in a SPARQL Event Processing Architecture (SEPA) triple store via the SEPY client API and a structured JSAP configuration file.

On the user-facing side, the system enables natural language interaction, where free-text queries are processed by an LLM (specifically, GPT-4o). The model translates these inputs into SPARQL queries that are syntactically valid and semantically aligned with the SOSA and KNMI vocabularies. These queries are then executed against the SEPA endpoint to retrieve the corresponding weather observations from the RDF graph.

Finally, the system provides two visualization components: a Folium-based map for displaying the spatial distribution of KNMI stations across the Netherlands, and Pandas and Matplotlib plots for comparing historical weather data retrieved from KNMI and OpenWeather. This architecture illustrates how AI-powered natural language interfaces and semantic web technologies can be jointly deployed to facilitate intuitive access to structured meteorological data, forming a foundational layer for potential extensions of the Agorà platform in real-world smart agriculture use cases.

3.2 Data sources

Meteorological data used in this thesis were retrieved from two public providers: the KNMI (Royal Netherlands Meteorological Institute) and OpenWeather. Both organizations offer multiple datasets and APIs, each with different data formats, time resolutions, and access methods. To ensure consistency and reliability, several candidate sources were explored and compared before selecting the most suitable raw data endpoints for the scope of this work.

For KNMI, data was accessed through two different mechanisms. The first was the KNMI EDR API, which provides observational data in JSON format. This API enables querying historical weather values from a gridded dataset [16], including daily minimum (TN), maximum (TX), and mean (TG) air temperatures, for specific coordinates. The second source was the KNMI OpenData API, which exposes complete multi-station historical datasets in NetCDF format. The latter was processed using the Python xarray library, which allows for structured parsing and extraction of relevant parameters, such as station names, geographic coordinates, and temperature observations. The result was a tabular dataset visualized using Pandas and Matplotlib, enabling both numerical inspection and graphical comparison.

In parallel, the OpenWeather History API was queried using a different structure, relying on global grid-based weather archives and customizable time intervals. The API returns data in simplified JSON format, making it easier to integrate but often less localized than station-based data. To enable fair comparison, both the KNMI and OpenWeather datasets were filtered and aligned by timestamp and location (De Bilt), and the retrieved values were plotted over a one-year observation period.

Figure 3.2 Shows the comparison between KNMI and OpenWeather daily minimum and maximum temperatures for the De Bilt region over a 12-month interval. The discrepancies observed between the two sources underscore the importance of data selection and standardization in building robust smart agriculture pipelines.

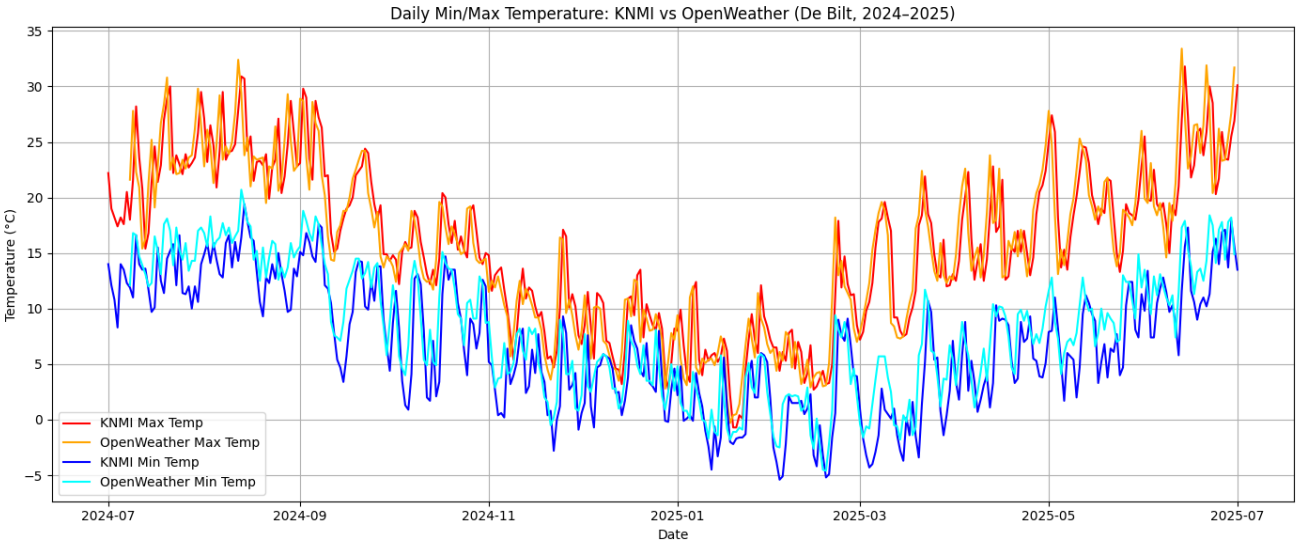

Figure 3.2- Comparison of daily minimum and maximum air temperatures retrieved from KNMI and OpenWeather for the De Bilt region over 1 year.

In addition to the temperature comparison, a geographic analysis of KNMI station coverage was conducted. The NetCDF file retrieved from the KNMI OpenData platform includes metadata for all 51 automatic weather stations, of which 37 are land-based and 14 are located at sea. A Folium-based interactive map was created to visualize station distribution across the Netherlands. This allowed qualitative inspection of spatial coverage, helping identify areas that may lack nearby observations. The goal was not to quantify accuracy by distance, but rather to gain an intuitive understanding of potential coverage gaps.

Figure 3.3- Map of KNMI weather stations across the Netherlands (Green points represent Land stations and red points are Sea stations).

To semantically integrate the processed weather data into the system, observations were annotated using the SOSA ontology and converted into RDF triples. These triples were inserted into the SEPA broker using the SEPY Python client, which requires a configuration file in JSAP format. The JSAP defines all queryable SPARQL endpoints by specifying `QUERY` and `UPDATE` templates in JSON. Understanding how to write and customize JSAP files was an essential prerequisite for making SEPA interactions modular and reusable.

A simplified snippet of the JSAP file used in this project is shown below, highlighting the structure of an `INSERT_OBSERVATION` update template:

```
{ "INSERT_OBSERVATION": {
    "sparql": "PREFIX sosa: <http://www.w3.org/ns/sosa/> PREFIX knmi: <http://example.org/knmi#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> INSERT { GRAPH
<http://example.org/knmi/observations> { ?observation a sosa:Observation ;
sosa:hasFeatureOfInterest ?foi ; sosa:observedProperty ?property ; sosa:hasResult ?value ;
sosa:resultTime ?timestamp ; sosa:madeBySensor ?sensor . } } WHERE {}",
    "forcedBindings": {
      "observation": { "type": "uri", "value": "knmi:OBSERVATION_ID" },
      "foi":         { "type": "uri", "value": "knmi:StationDeBilt" },
      "property":    { "type": "uri", "value": "knmi:TG" },
      "value":       { "type": "literal", "value": "18.5", "datatype": "xsd:float" },
      "timestamp":   { "type": "literal", "value": "2024-07-01T00:00:00Z", "datatype":
"xsd:dateTime" },
      "sensor":      { "type": "uri", "value": "knmi:TEMP_SENSOR" }
    }
  }
}
```

This configuration ensures that new observations can be programmatically inserted into the knowledge graph in a consistent and SPARQL-compliant manner. Working with JSAP required an understanding of `SPARQL INSERT` and `SELECT` queries, triple patterns, and RDF datatypes, all of which were incrementally learned and applied throughout the project via iterative development tasks.

Figure 3.4 Provides a snippet of the dataset after parsing it into a structured table using xarray and pandas, which facilitated the spatial analysis and mapping tasks.

stations	name	WMO	WSI	type	time	lat	lon	height	iso_dataset	product	projection
0	D15-FA-1	06201	0-20000-0-06201	Platform/AWS	2009-02-10	54.3256	2.9358	42.70	b''	b''	b''
1	P11-B	06203	0-20000-0-06203	Platform/AWS	1990-01-01	52.3600	3.3417	41.84	b''	b''	b''
2	K14-FA-1C	06204	0-20000-0-06204	Platform/AWS	1976-12-25	53.2694	3.6278	41.80	b''	b''	b''
3	A12-CPP	06205	0-20000-0-06205	Platform/AWS	2009-02-10	55.3992	3.8103	48.35	b''	b''	b''
4	L9-FF-1	06207	0-20000-0-06207	Platform/AWS	2006-11-27	53.6144	4.9603	45.31	b''	b''	b''
...	...	...	...	...	...	...	...	...	...	...	...
67	Arcen	06391	0-20000-0-06391	AWS	2006-11-09	51.4972	6.1961	19.50	b''	b''	b''
68	Horst	06392	0-528-0-06392	AWS	2024-11-20	51.4869	6.0561	21.88	b''	b''	b''
69	Saba Airport	78871	0-20000-0-78871	AWS	2016-01-01	17.6461	-63.2208	31.00	b''	b''	b''
70	Sint Eustatius Airport	78873	0-20000-0-78873	AWS	2016-01-01	17.4956	-62.9828	36.02	b''	b''	b''
71	Bonaire Airport	78990	0-20000-0-78990	AWS/Aerodrome	2013-01-01	12.1300	-68.2758	2.11	b''	b''	b''

[72 rows x 11 columns]

Figure 3.4- KNMI station metadata extracted from NetCDF and displayed as a Pandas DataFrame

3.3 AI Module

To enable natural language interaction with the weather data stored in SEPA, the system integrates an AI module powered by the OpenAI GPT API. The goal of this module is to translate human-readable questions into syntactically correct SPARQL queries that are compatible with the RDF vocabulary used in the knowledge graph (i.e., SOSA and KNMI namespaces).

The selected model for this task is GPT-4o, accessed through the chat.completions endpoint of the OpenAI Python SDK. A simple two-message prompt is used: the system message defines the assistant's role as a SPARQL expert (with access to SOSA ontology), and the user message contains a weather-related question, such as:

```
from openai import OpenAI

client = OpenAI(api_key="sk-...")

response = client.chat.completions.create(
          model="gpt-4o",
          messages=[
 {"role": "system", "content": "You are a SPARQL expert. You generate SPARQL queries from natural
language questions."},
 {"role": "user", "content": "How much was the maximum temperature today in De Bilt?"} ],

          temperature=0.0 )

print(response.choices[0].message.content)
```

The `temperature=0.0` parameter disables randomness in the LLM's output, ensuring that the model always returns the most probable response. This deterministic behavior is critical for generating consistent and reproducible SPARQL queries, especially during testing and validation. By contrast, using higher temperature values (e.g., 0.7 or 1.0) introduces variability, which can lead to hallucinated predicates, incorrect vocabulary usage, or syntactically invalid queries, making the system unreliable for structured data interactions. The resulting SPARQL string is printed and passed directly to the SEPY client for execution against the SEPA endpoint.

This module was iteratively evaluated with multiple models (GPT-4, GPT-4o, GPT-5, and GPT-5-thinking), and various prompt structures were explored. Outputs were manually reviewed for correctness in terms of syntax, vocabulary alignment, and semantic validity. The goal was not to train or fine-tune any LLM, but to assess feasibility and robustness when using LLM in direct integration without fine-tuning with structured weather data.

```
response = client.chat.completions.create(
          model="gpt-4o",
          messages=[
{"role": "system", "content": (
          "You are a SPARQL-generating assistant that answers natural language weather questions"
              "by generating SPARQL queries using the SOSA ontology and KNMI dataset structure.\n"
                 "Use these prefixes in all queries:\n"
                 "PREFIX sosa: <http://www.w3.org/ns/sosa/>\n"
                 "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n"
                 "PREFIX knmi: <http://example.org/knmi/>" ) },
{"role": "user", "content": "What was the maximum temperature in De Bilt on 16 September 2025?"
}],
          temperature=0.0 )

print(response.choices[0].message.content)
```

Overall, the AI module demonstrates that, even without fine-tuning, current large language models can produce executable and meaningful SPARQL queries in the context of weather data stored as RDF. These results open opportunities for future work in prompt optimization, vocabulary constraint injection, or reinforcement-style feedback loops to improve query accuracy and reduce hallucinations [17].

3.4 Knowledge Graph Module

To semantically structure the retrieved weather data, this thesis adopts a knowledge graph (KG) approach based on the Resource Description Framework (RDF). A knowledge graph represents information as subject–predicate–object triples, enabling the creation of machine-readable, interoperable data models. In this work, each weather observation was described using concepts from the SOSA (Sensor, Observation, Sample, and Actuator) ontology, which is designed for representing sensor-based data in the environmental and Internet of Things domains.

The annotated RDF data was inserted into a SEPA triple store using the SEPY Python client. SEPA enables both querying and subscribing to RDF data over time, supporting real-time updates to the graph. To interact with SEPA, a structured configuration for the JSAP file was defined. This file specifies `SPARQL INSERT` and `SELECT` operations along with the necessary bindings and namespaces for interacting with the knowledge base.

Before integrating the pipeline, early development experiments involved creating a small RDF dataset in Turtle (.ttl) format and loading it into Protégé for validation [18]. This allowed a preliminary understanding of how ontologies like SOSA are structured and how semantic models are navigated. These foundational steps ensured that the final system could semantically represent and store weather observations in a SPARQL-compliant manner, serving as the backbone for LLM-powered query execution in later stages.

To formally represent the framework of environmental observations, the SOSA ontology was utilized as the base vocabulary. SOSA offers a lightweight and adaptable structure for detailing sensor-driven observations, encompassing the property being observed, the sensor itself, the outcome, and the feature of interest. To become familiar with the SOSA classes and properties, initial testing was performed using Protégé, where a simple RDF graph was manually constructed in Turtle syntax and visualized. The following Figure 3.5, outlines the fundamental structure of the SOSA ontology, showcasing its key relationships with the PROV-O (Provenance Ontology) model. This diagram, sourced from Janowicz et al. [12], informed the development of the RDF triples and ensured semantic accuracy in aligning the dataset schema with the ontology's framework.

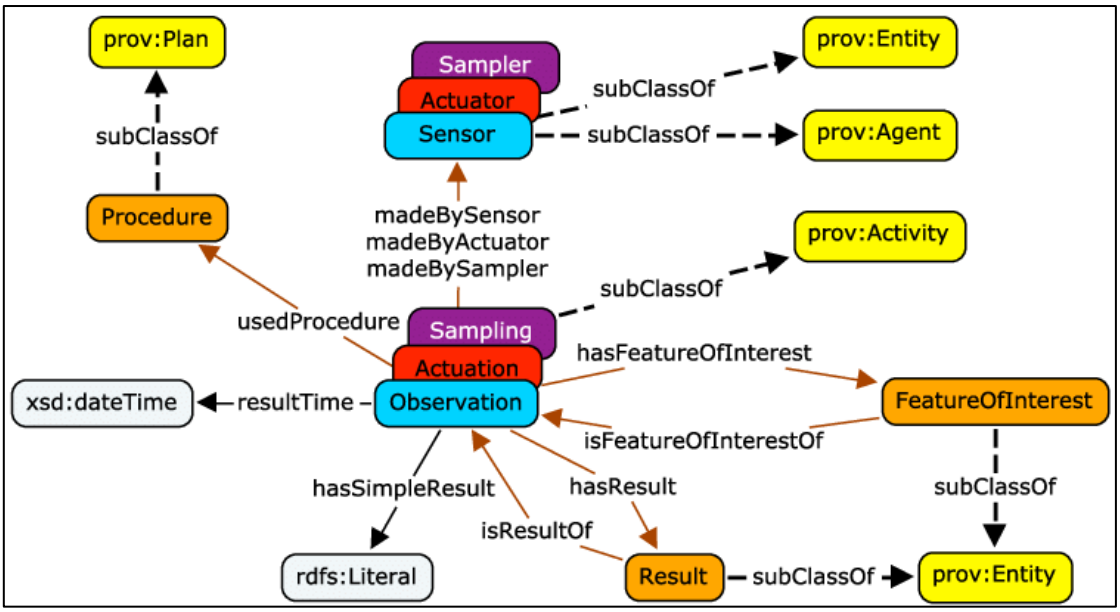

Figure 3.5- Core structure of the SOSA ontology [12].

4 Results

4.1 Experimental setup

The implementation phase involved retrieving and processing one year of temperature observations from two sources: KNMI and OpenWeather APIs.

Table 4.1- Summary of Temperature Data Sources

Feature	KNMI EDR API	OpenWeather History API
Source Type	Station-based (Netherlands)	Global grid-based model
Temporal Coverage	Full historical coverage	Limited to certain years
Granularity	Daily min, max, mean (TG, TN, TX)	Daily min, max, mean
Data Format	OGC Coverage JSON	Simplified JSON
Geo Precision	Exact station location	Approximate geocoordinates
Accessibility	Open Data Portal (NetCDF, EDR)	Commercial API

Initial comparisons revealed that KNMI provided more precise and readily available data, as it is directly derived from physical weather stations nationwide. In contrast, OpenWeather delivers global coverage with a more generalized, grid-based approach, which may sacrifice local accuracy. This observation aligns with OpenWeather's own report on accuracy and quality of weather data, which states that their outputs are based on a combination of modeled and observational sources [19].

To visualize the discrepancies between the two APIs, the retrieved values were aligned by timestamp and plotted using Pandas and Matplotlib. Figure 3.2 (previous section) illustrates potential differences in temperature values throughout the year.

4.2 Test scenarios

This section describes how the system's behavior changed in response to modifications made to the prompt used in the OpenAI GPT-4 model, as discussed in Section 3.3.

In the initial test, a general prompt was provided to the LLM without any SPARQL or ontology context. The result, shown in Figure 4.1 was a natural language explanation, warning that SPARQL cannot retrieve real-time data directly from weather APIs. The model correctly explained that SPARQL is used for querying RDF-based knowledge graphs, not for direct API access.

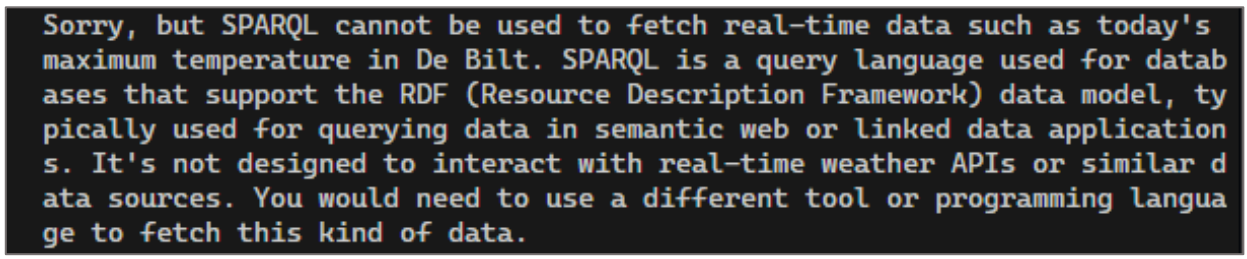

Figure 4.1- Output Returned by GPT-4 from OpenAI Response API

Subsequent tests with the same prompt and different models (e.g., GPT-5, GPT-5 Thinking) yielded similar generic responses, often indicating that more context or specific data access tools were required to answer the question accurately.

To improve the output, the prompt was refined by explicitly stating the desired behavior (i.e., generating a SPARQL query using SOSA and KNMI vocabulary) and by including relevant prefixes in the system role message. This change produced significantly better results: instead of returning an answer, the model generated a structured and syntactically valid SPARQL query. An example of this improved output is shown in Figure 4.2.

To find the maximum temperature in De Bilt on 16 September 2025, you can use the following SPARQL query:

```sparql
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX knmi: <http://example.org/knmi/>

SELECT ?maxTemperature
WHERE {
  ?observation sosa:hasFeatureOfInterest knmi:DeBilt ;
               sosa:observedProperty knmi:maximumTemperature ;
               sosa:resultTime "2025-09-16"^^xsd:date ;
               sosa:hasSimpleResult ?maxTemperature .
}
```

This query selects the maximum temperature observed in De Bilt on the specified date.

Figure 4.2- Output Returned by GPT-4o from OpenAI Response API

The results, summarized in Table 4.2, indicate a clear improvement in SPARQL generation accuracy when moving from earlier models to GPT-5 Thinking, especially when SOSA and KNMI vocabularies were embedded into the system prompt.

The output still required careful evaluation to ensure semantic correctness and alignment with the underlying ontology. Still, it represented a successful step toward connecting natural language interfaces with machine-interpretable environmental data.

Table 4.2- Comparison of SPARQL Generation by Different GPT Models

Model	Prompt	Output	Hallucination	SPARQL Validity	Comments
GPT-4	Basic prompt	Textual explanation	Yes	Invalid	Misinterpreted query intent, suggested general-purpose tools
GPT-4o	SOSA+ enhanced prompt	Partial SPARQL	Occasional	Partially valid	Improved syntax, but needed domain prefixes to avoid errors
GPT-5	SOSA + Prefixes	Mostly correct SPARQL	Rare	Mostly valid	Good structure, minor vocabulary mismatches
GPT-5 Thinking	Refined contextual	Mostly correct SPARQL	Rare	Mostly valid	Correct SOSA structure and accurate bindings, but high latency

4.3 Discussion

The research findings outlined in this thesis support the technical viability of integrating natural language interfaces with RDF-based weather knowledge graphs, particularly in the context of smart agriculture. By leveraging GPT-4's capabilities to convert user questions into SPARQL queries and linking them to a SEPA knowledge graph populated with carefully organized temperature data, this prototype establishes a valuable and relevant connection between AI-driven language models and semantic web technologies.

The comparative examination of data from KNMI and OpenWeather highlights the crucial importance of data provenance and reliability in informing agricultural decision-making processes. Although both APIs showed generally consistent trends, the KNMI data, derived from official weather stations throughout the Netherlands, displayed less variability and greater temporal consistency compared to OpenWeather data, which aggregates information from global models and various external sources. This observation aligns with OpenWeather's own findings, which acknowledge that accuracy can vary significantly across different locations and time frames. Such differences can have significant implications in real-world applications, such as irrigation planning, where precise moisture level thresholds determine whether to initiate or postpone irrigation actions.

From the perspective of artificial intelligence, initial assessments indicated that language models like GPT-4 faced difficulties in producing syntactically correct and semantically appropriate SPARQL queries without specific prompt engineering. Initial results often contained hallucinatory components or used unsuitable terminology. However, by enhancing the system's prompts with targeted domains, such as the SOSA ontology and the KNMI vocabulary, and by offering accurate SPARQL prefixes and structural guidelines, the quality of the generated queries improved significantly. This observation highlights the sensitivity of LLM performance to the design of prompts and their contextual grounding, a crucial consideration for future integrations of LLMs with knowledge graphs.

While the current prototype does not fully represent Agorà's comprehensive decision-making logic, its modular architecture enables potential future improvements. Prospective enhancements could involve adding geospatial reasoning abilities through GeoSPARQL, context-aware prompting tailored to specific agricultural situations, or developing real-time fallback systems that intelligently select the most reliable weather API based on coverage or confidence levels.

Furthermore, implementing feedback loops and correction mechanisms for erroneous queries, possibly through reinforcement learning methods or by analyzing user interaction logs, could play a crucial role in reducing hallucinations and enhancing the overall resilience and accuracy of the system. This comprehensive approach to enhancing LLM and knowledge graph integration is essential for advancing the tools available to modern agricultural practitioners.

5 Conclusion and Future Work

This thesis has demonstrated the feasibility of combining large language models (LLMs) with semantic web technologies to enable natural language interaction with weather-related knowledge graphs for smart agriculture applications. By linking GPT-generated SPARQL queries to a SEPA semantic backend populated with structured temperature data from KNMI and OpenWeather, the system facilitates intuitive environmental queries, marking an important step toward intelligent decision-support systems in precision farming.

A key area for future research is improving the robustness and precision of SPARQL query generation by the LLM. In this study, GPT-4o was employed to translate user questions into SPARQL, guided by carefully crafted prompts and a vocabulary aligned with the SOSA. While many outputs were syntactically correct and semantically relevant, several failure modes were observed, such as incomplete triple patterns, invalid variable bindings, and hallucinated predicates that are not present in the RDF schema. In smart agriculture, such inaccuracies are especially problematic, as even minor errors may lead to incorrect irrigation scheduling or misinterpretation of environmental data.

To address these challenges, future implementations could incorporate feedback mechanisms to detect and respond to query failures, either through runtime execution feedback or structural validation. This feedback could inform prompt redesign, improve training dataset curation, or, where applicable, support fine-tuning of the underlying language model. Over time, this would establish a reinforcement-style learning loop, in which invalid queries help refine system behavior. Although fine-tuning was beyond the scope of this thesis, such mechanisms could significantly reduce hallucinations and improve semantic alignment. Incorporating Retrieval-Augmented Generation (RAG) techniques may further increase control over knowledge usage and improve query consistency [20][21].

Another valuable extension would be to enhance the system's ability to dynamically select data sources based on geographic input. While this thesis focused on KNMI and OpenWeather data for the De Bilt region, many APIs lack uniform global coverage. Future versions could analyze API availability and reliability for a given latitude–longitude pair and automatically select the most appropriate provider. This would improve the system's generalizability and effectiveness in diverse agricultural contexts.

Finally, incorporating GeoSPARQL, the W3C standard for geospatial RDF querying, would significantly expand the system's reasoning capabilities [22]. GeoSPARQL support would enable spatial filters such as identifying the nearest weather stations or visualizing gaps in sensor coverage and enhance the integration of geospatial data, including satellite observations and IoT-based sensor networks.

References

1. Agorà, https://vaimee.com/agora-2/
2. KNMI-Royal Netherlands Meteorological Institute, https://dataplatform.knmi.nl/
3. OpenWeather API, https://openweathermap.org/api
4. Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. 2014. url: https://www.w3.org/TR/rdf11-concepts/.
5. Lee Feigenbaum et al. SPARQL 1.1 Protocol, W3C Recommendation 21 March 2013. 2013. url: http://www.w3.org/TR/sparql11-protocol/.
6. Steve Harris e Andy Seaborne. SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. 2013. url: http://www.w3.org/TR/sparql11-query/.
7. Paula Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 Update, W3C Recommendation 21 March 2013. 2013. url: http://www.w3.org/TR/sparql11 update/.
8. https://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/
9. Endris, K.M., Vidal, ME., Graux, D. (2020). Chapter 5 Federated Query Processing. In: Janev, V., Graux, D., Jabeen, H., Sallinger, E. (eds) Knowledge Graphs and Big Data Processing. Lecture Notes in Computer Science(), vol 12072. Springer, Cham. https://doi.org/10.1007/978-3-030-53199-7_5
10. Roffia, L., Azzoni, P., Aguzzi, C., Viola, F., Antoniazzi, F., & Salmon Cinotti, T. (2018). Dynamic Linked Data: A SPARQL Event Processing Architecture. Future Internet, 10(4), 36. https://doi.org/10.3390/fi10040036
11. CRITERIA-1D, https://github.com/ARPA-SIMC/CRITERIA1D
12. Janowicz, Krzysztof & Haller, Armin & Cox, Simon & Phuoc, Danh & Lefrançois, Maxime. (2018). SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators. 10.48550/arXiv.1805.09979.
13. JSON SPARQL Application Profile (JSAP), https://vaimee.org/TR/jsap.html
14. SEPA Python3 APIs, https://github.com/arces-wot/SEPA-python3-APIs
15. OpenAI. (2023). OpenAI API documentation. https://platform.openai.com/docs/
16. Subedi, Samikshya & Kechchour, Ayoub & Kantar, Michael & Sharma, Vasudha & Runck, Bryan. (2025). Can gridded real-time weather data match direct ground observations for irrigation decision-support? . Agrosystems, Geosciences & Environment. 8. 10.1002/agg2.70100.
17. Emonet, V., Bolleman, J., Duvaud, S., de Farias, T. M., & Sima, A. C. (2024). Llm-based sparql query generation from natural language over federated knowledge graphs. arXiv preprint arXiv:2410.06062.
18. Protégé, Ontology Editor, https://protege.stanford.edu/
19. Accuracy and quality of weather data at OpenWeather, https://openweather.co.uk/accuracy-and-quality
20. Yang, S. et al. (2025). ShizishanGPT: An Agricultural Large Language Model Integrating Tools and Resources. In: Barhamgi, M., Wang, H., Wang, X. (eds) Web Information Systems Engineering – WISE 2024. WISE 2024. Lecture Notes in Computer Science, vol 15439. Springer, Singapore. https://doi.org/10.1007/978-981-96-0573-6_21

21. Samuel, Dinesh Jackson & Skarga-Bandurova, Inna & Sikolia, David & Awais, Muhammad. (2025). AgroLLM: Connecting Farmers and Agricultural Practices through Large Language Models for Enhanced Knowledge Transfer and Practical Application. 10.48550/arXiv.2503.04788.
22. GeoSPARQL, https://www.ogc.org/standards/geosparql/
23. CoverageJSON, https://www.ogc.org/standards/coveragejson/
24. Semantic Sensor Network Ontology, https://www.w3.org/TR/vocab-ssn/