Eurostars Project

SPARQL-ML: Machine Learning for SPARQL Query Optimization over Centralized and Distributed RDF Knowledge Graphs

Project Number: 5736 **Start Date of Project:** 2024/11/01 **Duration:** 36 months

Deliverable 1.2
Conceptual Architecture

Dissemination Level	Public
Due Date of Deliverable	April 30, 30/04/2025
Actual Submission Date	April 30, 30/04/2025
Work Package	WP1, Requirements Elicitation & Conceptual Architecture
Deliverable	D1.2
Type	Report
Approval Status	Final
Version	1.0
Number of Pages	7

Abstract: The SPARQL-ML project aims to improve SPARQL query optimization for centralized and distributed RDF Knowledge Graphs using machine learning techniques. WP1, led by OpenLink, brings together project partners to identify and define the functional requirements for this goal and to develop a suitable technical architecture tailored to meet these established requirements. This deliverable, D1.2, focuses on the development of the conceptual architecture which will serve as the basis for the other work packages. Having the technical requirements specified in D1.1 as a starting point, the partners produced the system architecture for the project, which contains and describes the components and services of the system, their roles and interconnection, along with an overall system information flow, which details the main expected behaviors.

eurostars™

History

Version	Date	Reason	Revised by
0.1	18/02/2025	Initial Template & Structure	Milos Jovanovik
0.2	21/02/2025	Detailed Architecture	Milos Jovanovik
0.3	03/03/2025	Additional Content and Conclusion	Muhammad Sohail Nisar
0.4	09/04/2025	Additional Content	Edgard Marx
1.0	30/04/2025	Finalizing	Milos Jovanovik, Mirko Spasić, Hugh Williams

Author List

Organization	Name	Contact Information
OpenLink Software	Milos Jovanovik	mjovanovik@openlinksw.com
OpenLink Software	Mirko Spasić	mspasic@openlinksw.com
OpenLink Software	Hugh Williams	hwilliams@openlinksw.com
eccenca GmbH	Edgard Marx	edgard.marx@eccenca.com
University of Paderborn	Muhammad Sohail Nisar	msohail@mail.uni-paderborn.de

Contents

1 Introduction

As modern applications increasingly depend on large datasets that exceed the capabilities of single-server environments, distributed architectures have become essential for efficient RDF Knowledge Graph management and SPARQL query optimization. The SPARQL-ML project builds on these foundations by incorporating machine learning techniques to enhance query performance in both centralized and distributed settings. To ensure seamless integration of these techniques, a well-defined system architecture is necessary, outlining the interaction between components and services responsible for data processing and optimization.

This deliverable, D1.2, builds upon the technical requirements established in D1.1 [3] by defining the conceptual architecture that underpins the entire project. The system architecture developed by the project partners specifies the components and services required for machine learning-driven SPARQL query optimization, detailing their roles, interactions, and overall data flow. This architectural framework serves as the foundation for subsequent work packages, providing a structured approach to implementing and evaluating the proposed methods.

2 SPARQL-ML Architecture

In recent years, the emergence of large RDF Knowledge Graphs [2], such as the 15 billion triples in the Wikidata Knowledge Graph [6] used by Google's search engine and the 210 billion triples in UniProt [1], has presented a considerable challenge in efficiently querying these vast repositories. Traditional query optimization methods have not fully explored the potential of modern deep learning techniques for scalable execution of SPARQL queries over massive and distributed RDF Knowledge Graphs. The current landscape highlights a growing necessity to explore contemporary AI and machine learning-based solutions to enhance SPARQL query processing over both centralized and distributed RDF Knowledge Graphs. As data volumes continue to increase and become more widely distributed, organizations face challenges in optimizing the use of their data assets.

Existing distributed solutions often cater to centralized storage or static data distribution, resulting in suboptimal query performance. Consequently, there is a pressing demand for advanced data distribution strategies and federated query engines capable of handling substantial amounts of data with high efficiency. Our objective is to elevate data management to a new standard, capitalizing on high-performance, scalability, and efficient handling of semantic data to meet the evolving demands of the market.

The architecture for SPARQL-ML is shown in Figure 1. Starting from the bottom, we will first construct Knowledge Graphs (KGs) using the Resource Description Framework format (RDF), a W3C standard for the representation of knowledge on the Web [4]. The source data will be data which is currently missing formal semantics and exists in structured, semi-structured or unstructured formats. We then distribute the resulting RDF KGs among different data storage solutions, i.e., triple stores with public SPARQL endpoints, thus allowing users to query them by means of SPARQL, the W3C standard for querying RDF data [5]. The triplestores will also host the data from the project use-cases which are already represented as RDF KGs. The triplestores will be able to exchange the data dynamically, to exploit data locality for maximizing and balancing the amount of computation in a single storage solution. The triplestores will use machine learning (ML) techniques particularly deep reinforcement learning (DRL) in order to optimize the incoming SPARQL queries, for improved performance. The user can query the distributed triplestores directly or via the SPARQL federation engine, which makes use of the federation index which is calculated beforehand, based on the profile of each triplestore.

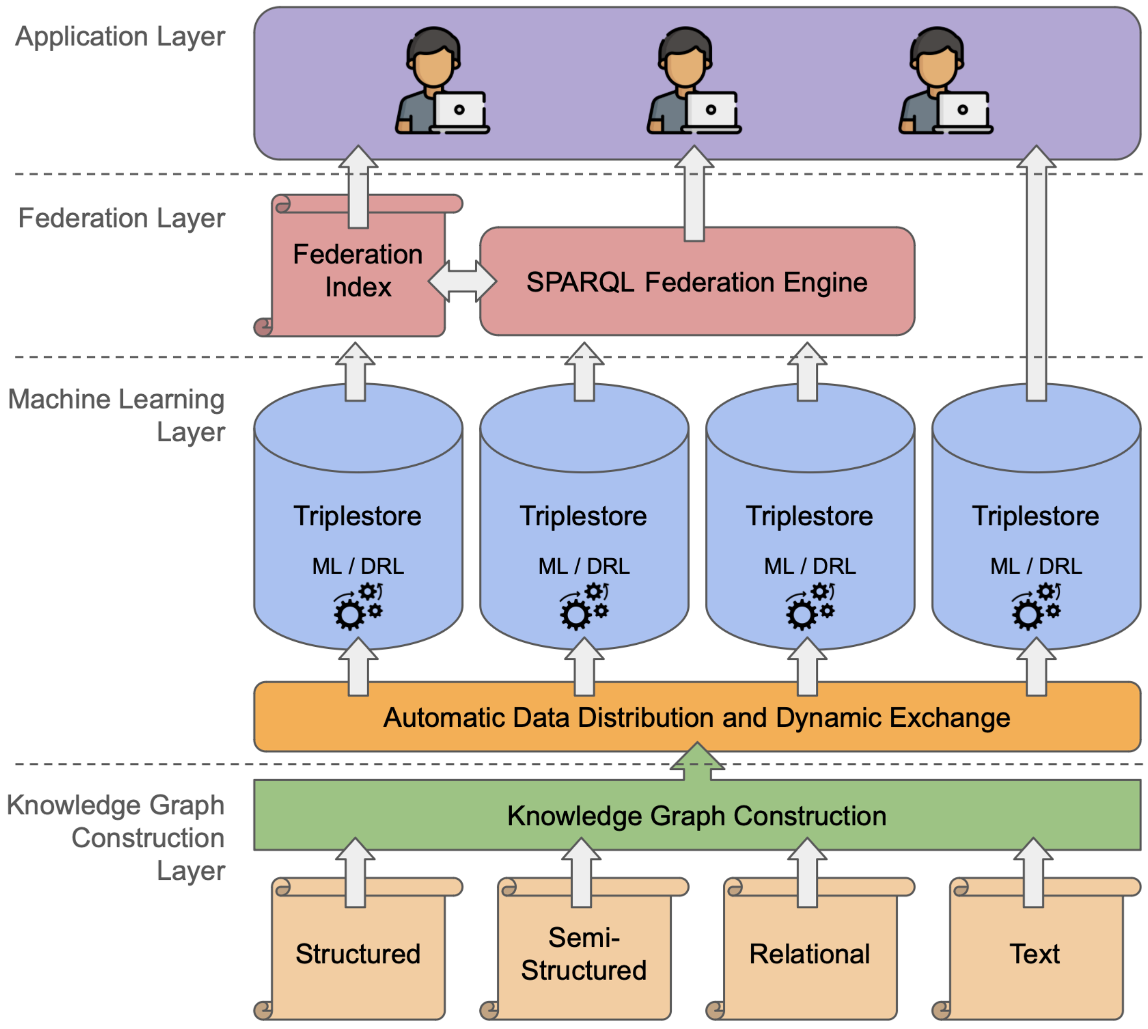

Figure 1: SPARQL-ML Architecture

2.1 Knowledge Graph Construction Layer

In this architectural layer, we will transform all business-relevant data into rich, robust Knowledge Graphs (KGs) using the Resource Description Framework (RDF), which is the internationally recognized W3C standard for representing knowledge on the Web. This process not only addresses structured, semi-structured, and unstructured data but also leverages the eccenca Data Integration workflow pipeline. This pipeline plays a crucial role in orchestrating the extraction, transformation, and loading (ETL) of diverse data sources, ensuring that data is meticulously standardized and seamlessly integrated throughout the conversion process.

Due to the complexity of orchestrating data ingestion into the triplestore, Corporate Memory data integration will be used for placing the relevant benchmark dataset into the dedicated triplestore. This hosting arrangement facilitates efficient management and retrieval, allowing users to perform sophisticated queries via SPARQL—the standardized query language for RDF data. The transformation and storage phases are enabled by a tightly coupled integration between the Corporate Memory (CMEM) data integration platform, the eccenca Data Integration workflow pipeline, and Tentris operating as a state-of-the-art triplestore solution.

This comprehensive integration framework guarantees that all data types are processed uniformly, converting raw input from various origins into a unified, queryable knowledge base. The eccenca Data Integration workflow pipeline, in particular, provides an enhanced level of automation and consistency throughout the data transformation lifecycle, thereby improving overall data quality and enabling more insightful analytics aligned with the project's diverse business objectives.

2.2 Machine Learning Layer

In this layer we will use our existing solutions for automatic data distribution between storage nodes (in a clustered version of the data storage solution) and for dynamic data exchange between the nodes, based on usage load information. This dynamic exchange of data can be a deletion, an insertion or the insertion of a replicated chunk of data from another storage solution. The decision of dynamic exchange will be mostly based on the monitoring of the storage solutions, in particular on the federated queries that were issued to the distributed storage solution.

Each triplestore will implement an ML-based solution for SPARQL query optimization of the incoming queries. More specifically, we aim to apply deep reinforcement learning (DRL) techniques to optimize the SPARQL query processing in triplestores, motivated by the improved performance of using DRL in optimizing relational database queries. The main goal of this component will be to select an optimal join-order in the query plan that leads to significant performance improvement in terms of query execution time. The proposed optimization methods will be implemented in Tentris, a well-known state-of-the-art triplestore for SPARQL query processing, freely available from the DICE research group at University of Paderborn. In addition, OpenLink will also adopt the proposed methods in their Virtuoso RDF Quad Store. We will use both real-world and synthetic SPARQL benchmarks to evaluate the proposed optimization algorithms to be used in this layer.

2.3 Federation Layer

In this layer, we will use our SPARQL query federation engine developed in a previous research project (3DFed). The join ordering of the plans it generates is based on an estimation of the cardinality of the triple patterns and the joins contained in the input query. The decision between the join implementation to use (bind or symmetric hash join) is based on the join cost estimation function it implements. In addition, it will implement means to keep its index up-to-date based on regular intervals as well as at fixed times.

2.4 Application Layer

In this layer, we will provide user friendly interfaces for querying the distributed data hosted by different data storage solutions.

3 Architecture for Deep Reinforcement Learning for SPARQL

In this section we present the architecture for SPARQL query optimization using Deep Reinforcement Learning (DRL). Figure 2 depicts the DRL architecture which will be applied in our use-case as follows.

3.1 Agent

Deep Reinforcement Learning Model: This component represents the DRL agent trained to optimize SPARQL queries by making decisions based on the state it observes.

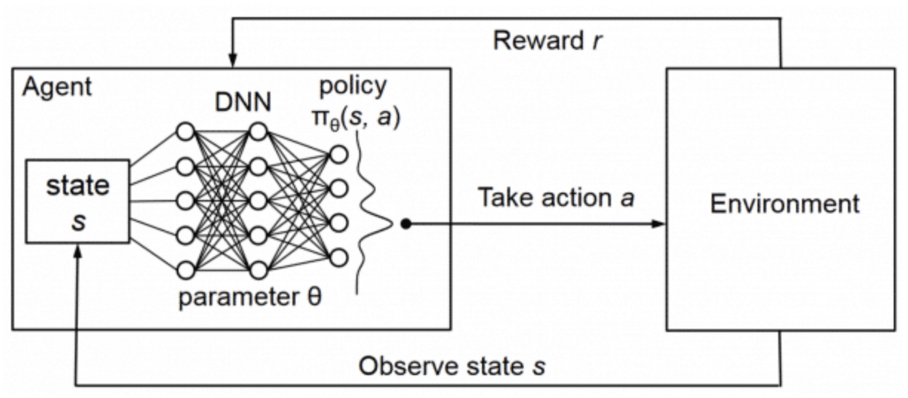

Figure 2: The DRL-based SPARQL Query Optimization

3.2 State

Current Query Configuration: This is the input to the agent that represents the current unoptimized query plan with a set of joins. Basically, the set of states are the joins that need to be performed in the current SPARQL query to get the complete results.

3.3 Action

Optimization Decisions: These are the potential actions the agent can take to optimize the SPARQL query plan, like the selection of a join from the remaining set to be executed for the query under execution. It also includes the execution of the selected join from the given SPARQL query.

3.4 Environment

Triplestore Database System: The component where the SPARQL query gets executed and which evaluates the actions taken by the agent, returning a performance measure like Tentris or Virtuoso, etc.

3.5 Reward

Performance Feedback: The feedback value provided to the agent, often related to the efficiency of the executed action in terms of reduced execution time. The negative of execution time value act as a reward for the DRL agent.

3.6 Deep Neural Network (DNN)

Q-value Function Approximation: The neural network within the agent tasked with learning the Q-value function, predicting the expected rewards for actions taken in various states.

3.7 Policy

Action Selection Strategy: The Deep Q-Network (DQN) policy directs the agent to select the action with the highest predicted Q-value, indicating the action's potential for earning the highest cumulative reward over time.

This structured flow encapsulates the iterative learning process the DRL agent undergoes, continuously refining its policy to achieve optimal SPARQL query performance.

4 Conclusion

This deliverable outlines the conceptual architecture of SPARQL-ML, focusing on integrating machine learning techniques to optimize SPARQL query execution over centralized and federated RDF Knowledge Graphs. The proposed architecture incorporates multiple layers, including knowledge graph construction, deep reinforcement learning-based query optimization, and a federation engine for distributed query processing.

By leveraging Deep Reinforcement Learning, the system aims to enhance query execution efficiency by dynamically selecting optimal query plans. The architecture also includes a robust data distribution mechanism, ensuring efficient query execution across multiple triplestores while minimizing network overhead. The federation engine extends these capabilities to distributed RDF stores, enabling seamless and scalable SPARQL query execution.

This work will focus on implementing and evaluating the proposed architecture using real-world and synthetic benchmarks. The outcomes of this project will contribute to advancing SPARQL query processing by introducing ML-driven optimizations, improving performance, and making large-scale RDF data more accessible for complex queries.

References

[1] UniProt Consortium. UniProt: A Worldwide Hub of Protein Knowledge. *Nucleic Acids Research*, 47(D1):D506–D515, 2019.

[2] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge Graphs. *ACM Computing Surveys*, 54(4):1–37, 2021.

[3] Milos Jovanovik, Mirko Spasić, Edgard Marx, and Muhammad Sohail Nisar. Deliverable D1.1: Requirement Specification, 2025. SPARQL-ML Project Deliverable.

[4] Frank Manola, Eric Miller, Brian McBride, et al. RDF Primer. *W3C Recommendation*, 10(1-107):6, 2004.

[5] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.

[6] Denny Vrandečić and Markus Krötzsch. Wikidata: a Free Collaborative Knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.