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 3.1

Initial Report on the Machine Learning for SPARQL Optimization in Triplestores

Dissemination Level	Public
Due Date of Deliverable	February 28, 28/02/2026
Actual Submission Date	February 27, 27/02/2026
Work Package	WP3, Machine Learning for SPARQL Query Optimization in Triplestores
Deliverable	D3.1
Type	Report
Approval Status	Final
Version	1.0
Number of Pages	9

Abstract: This deliverable (D3.1, UPB, M16) reports the initial progress of Work Package 3 of the Eurostars SPARQL-ML project, which focuses on machine learning-based SPARQL query optimization in triplestores using deep reinforcement learning (DRL). We implemented an end-to-end optimization pipeline built around the Tentris triplestore, including a custom Gymnasium environment, a Maskable PPO agent with action masking, and a FastAPI-based communication layer. For query representation (T3.1), we developed Transformer-based embeddings combined with structural query features. These embeddings are generated on the fly and consumed directly by the DRL agent. This work completes Milestone MS5 (M16) by enabling embeddings for real-world SPARQL queries extracted from LSQ v2. The system supports reproducible experimentation across multiple datasets and employs a warm-cache runtime protocol to ensure stable reward signals.

History

Version	Date	Reason	Revised by
0.1	29/01/2026	Initial template	Mirko Spasić
0.2	29/01/2026	Initial template and structure	Muhammad Sohail Nisar
0.3	11/02/2026	Content completion, MS5 milestone clarification, dataset/workload alignment	Muhammad Sohail Nisar
1.0	25/02/2026	Virtuoso specifics, polishing the content, adding references	Mirko Spasić, Milos Jovanovik

Author List

Organization	Name	Contact Information
University of Paderborn	Muhammad Sohail Nisar	msohail@mail.uni-paderborn.de
OpenLink Software	Mirko Spasić	mspasic@openlinksw.com
OpenLink Software	Milos Jovanovik	mjovanovik@openlinksw.com

Contents

1 Introduction

This deliverable (D3.1, UPB, M16) reports the initial progress on "Machine Learning for SPARQL Optimization in Triplestores" within the Eurostars SPARQL-ML project. The central goal is to improve SPARQL query execution performance by learning to generate more efficient query plans using machine learning techniques, with particular emphasis on deep reinforcement learning (DRL) for join-order optimization.

As proposed in the project plan, our approach consists of two tightly integrated tracks:

(i) **SPARQL query representation** (T3.1), where queries are transformed into numerical embeddings that capture both structural and semantic characteristics, and

(ii) **DRL-based optimization** (T3.2), where a reinforcement learning agent observes these query representations and learns to produce query plans that minimize execution time on a target triplestore engine.

At M16, UPB has implemented a complete end-to-end training pipeline built around the Tentris triplestore [2]. This pipeline includes: query feature extraction through an API bridge, a custom Gymnasium environment [3] for reinforcement learning training, a Maskable PPO agent, and multiple embedding mechanisms (including a Transformer-based variable encoder). A similar pipeline will be built around the Virtuoso triplestore [8, 4]. For this system, the query feature extraction phase has already been implemented by the OpenLink team, while the relevant information is passed to the API bridge in JSON format.

In addition, UPB established a data engineering workflow to curate, filter, and balance query workloads and to support consistent **train/validation/test splits across datasets**. The current datasets used for training and evaluation are **LinkedGeoData** [9], **Wikidata** [11], **SWDF** [6], and **DBpedia** [1, 5]. The remainder of this report is structured according to the promised tasks and details the current progress, identified challenges, and planned next steps.

2 SPARQL Query Representation and Workload Preparation (T3.1)

This section covers the work corresponding to the planned task T3.1: SPARQL query embeddings / query representation, and the dataset preparation required to train and evaluate the proposed approach.

2.1 Query representation goals and delivered features (T3.1)

Goal and promise (T3.1): The proposal commits to representing SPARQL queries numerically via embeddings, leveraging both query structure (e.g., joins, BGPs, projected variables) and semantic information. The expected output is a representation that supports downstream optimization tasks and can be used by DRL (T3.2).

Current status: We implemented a feature-centric query representation pipeline based on the information that Tentris can extract reliably at runtime. The current representation includes:

- **Variable-level attributes:** variable names, degrees, and per-variable cardinalities.
- **Query-level attributes:** number of triple patterns, distinct flag, total query cardinality, and graph density.
- **Graph structure:** an adjacency matrix capturing variable co-occurrence and join connectivity.
- **Role masks:** projection variables, join variables, and non-join variables.

This representation is passed from Tentris/Virtuoso to the API and then to the DRL environment as the observation. It directly supports the DRL agent's decision making process for join-order selection [7].

Milestone achievement (MS5). The work completed in Task T3.1 (Query Representation) has led to the achievement of **Project Milestone MS5 (M16) – Embeddings for SPARQL Queries**. In particular, we implemented and operationalized an embedding pipeline that generates numerical representations of SPARQL queries and supplies them directly as observations to the DRL optimizer in Task T3.2. In our implementation, embeddings are computed *on-the-fly* during training and evaluation and consumed immediately by the DRL agent. This design avoids a separate offline embedding storage step while still fulfilling the milestone objective of providing query embeddings to the DRL optimizer.

The query workloads are extracted from **LSQ v2** [10] (real-world queries originating from public endpoint query logs, including DBpedia), and are prepared into fixed train/validation/test splits to support reproducible experimentation.

2.2 Transformer-based embeddings (implemented)

Transformer-based embeddings (implemented): In addition to classical engineered features, we implemented a Transformer-based variable embedding module (`TransformerVariableEmbedding`) that produces *context-aware, order-invariant* variable vectors.

Key properties:

- **Semantic encoding:** variable names are encoded using a frozen Transformer encoder (e.g., DistilBERT) to obtain dense vectors.
- **Structural context:** neighbor pooling via the adjacency matrix and global pooling over all variables inject query context.
- **Role + scalar integration:** role indicators (`proj/join/nonjoin`) and scalar statistics (degree, log-cardinality, density, number of variables, number of triple patterns) are injected via fixed random projections.
- **Order invariance:** pooling operations are commutative (mean pooling), making embeddings invariant to arbitrary variable ordering (given aligned features).

Embeddings are computed *on-the-fly* as part of the environment observation and consumed immediately by the DRL agent.

2.3 Query workload selection and preparation (datasets and splits)

Query workload selection and preparation: The proposal references LSQ and feature-driven selection. In the current stage, we have established a robust pipeline to curate training workloads from benchmark/log-style query sets, focusing on properties most relevant to runtime optimization. In practice, the query workloads used in our experiments are extracted from **LSQ v2**, which provides real-world SPARQL queries collected from public endpoint query logs, including DBpedia. These queries are filtered, normalized, and organized into fixed **train/validation/test** splits stored in JSONL format to ensure reproducibility and controlled evaluation.

The target RDF knowledge graphs against which these query workloads are executed are referred to as *datasets* in this report. The datasets currently used and prepared in our experimental pipeline are:

- **LinkedGeoData (LGD)**
- **DBpedia**
- **Wikidata**
- **SWDF**

In this report, we use the term *dataset* to denote the RDF knowledge graph loaded into the Tentris triplestore, while *query workloads* refer to real-world SPARQL queries extracted from LSQ v2 and executed against the corresponding target datasets during training and evaluation.

Splits and balancing (all four datasets): For each dataset, we created **train/validation/test splits** in JSONL format and applied workload conditioning to avoid domination by trivial cases. In particular, we track join-related metadata (e.g., `joinVertexCount`) and use it to:

- filter or down-weight low-join queries where appropriate,
- construct balanced subsets (by join bucket and template where available),
- ensure evaluation uses fixed held-out workloads for comparability across checkpoints.

We observed that naive workloads contain many easy queries (e.g., low join complexity) that can dominate learning signals and reduce training efficiency. Therefore, we have created scripts and procedures to remove or reduce **join=1** queries when possible, balance join distributions, and maintain consistent manifests across datasets.

2.4 Join complexity tracking, sampling policy, and the "star join" challenge

Join complexity tracking and sampling policy: A practical challenge is that some query collections still contain many low-join queries even after filtering. Executing them wastes training time because episodes become trivial and provide limited learning signal for join-order optimization. To mitigate this, we implemented join-aware sampling and filtering mechanisms:

- **Metadata-based sampling:** when `joinVertexCount` is available, query sampling can be weighted (e.g., 50% join=2, 50% join ≥ 3) to ensure sufficient exposure to non-trivial join graphs.
- **On-load filtering:** minimum join thresholds can be applied while reading JSONL to avoid spending time on low-join instances.

Key structural mismatch with real-world queries: star joins vs. Tentris WCOJ.

A major observation from real-world query logs is that **many queries are dominated by star joins**, i.e., one variable appears in many triple patterns (TPs). In a classical binary-join optimizer, such star joins still produce a large combinatorial join-order space. However, **Tentris relies on a Worst-Case Optimal Join (WCOJ) execution strategy**. Under WCOJ, a star-shaped basic graph pattern often collapses into what is effectively **a single join variable driving most of the join structure**. This leads to an important challenge for RL:

- the **action space becomes extremely small** (often essentially selecting *one* join variable),
- the environment may yield **short episodes** with limited sequential decision making,

- learning signals for "join ordering" may become weaker or require reframing (e.g., learning decisions beyond variable choice).

This mismatch between typical real-world star patterns and Tentris's WCOJ behavior is currently one of the main conceptual and engineering challenges. It motivates future work on extending the controllable planning decisions (e.g., beyond a single variable choice) and on carefully selecting workloads where multiple meaningful join decisions exist.

3 DRL-Based Optimization Pipeline (T3.2)

This section covers the work corresponding to the promised task T3.2: Deep Reinforcement Learning for SPARQL optimization, including the development of a custom environment, the communication interface (bridge), reward design, and the current status of training and evaluation.

3.1 Custom RL environment and Tentris–API–Agent integration (T3.2)

Custom environment and system integration (T3.2): We implemented a dedicated Gymnasium environment around Tentris to enable closed-loop DRL training.

The end-to-end control flow is:

1. The environment selects a query and initiates its execution against Tentris.
2. Tentris extracts query-specific features and sends them to the API.
3. The API forwards these features to the environment via a queue manager.
4. The DRL agent generates a join-order plan (action) based on the observation.
5. The API returns the proposed plan to Tentris for execution.
6. Tentris executes the plan and posts the measured runtime back to the API.
7. The API forwards runtime feedback to the environment to compute reward and proceed.

Key components delivered:

- **FastAPI bridge** provides endpoints for receiving query features and receiving runtime feedback.
- **QueueManager** ensures isolation of plans, features and runtime data across episodes using per-`exec_id` buffering to prevent cross-talk and stale message leakage.
- **TrainQueryManager (driver)** handles query dispatch, logging/skipping of problematic queries, join-aware sampling, and (optionally) manages Tentris process lifecycle.
- **RL algorithm:** Maskable PPO (sb3-contrib) with action masking to guarantee valid join decisions.

3.2 Reward design, warm-cache runtime, and evaluation pipeline

Reward definition and evaluation pipeline: In the proposal, the reward was originally derived from query runtime (negative runtime). In practice, we refined this approach to improve learning stability and reduce noise:

- **Warm-cache reward protocol (explicit):** for each training episode, the same query is executed **twice using the same query plan**. The first execution warms caches (including engine internals, page cache, and operator state). The **runtime of the second execution** is then used as the reward signal (negative runtime). This approach reduces the variance due to the effects of cold-start and better isolates the impact of the agent's planning decisions.
- **Runtime feedback channel:** Tentris returns runtime together with status metadata (`ok/timeout/error`), which is forwarded to the environment for reward computation.

Training vs evaluation: We implemented an evaluation pipeline that periodically pauses training to measure the agent on a fixed evaluation workload. This enables:

- tracking whether improvements generalize beyond the training stream,
- detecting regressions and instability,
- selecting checkpoints based on evaluation performance.

Current observation (M16): Early runs confirm that the end-to-end pipeline works correctly. However, training remains at an early stage and is affected by the "star join / WCOJ" effect described above:

- episodes can be short (often length ≈ 1) when queries yield only one meaningful join decision,
- reward variance remains high because it depends on runtime measurements,
- evaluation currently provides a sanity check and checkpointing signal, but does not yet support publishable improvement claims.

Remaining steps for conference-grade validation

To achieve a VLDB-style paper submission, final validation requires a controlled test phase:

- run the trained agent on a held-out test set,
- compare against strong baselines (e.g., Tentris default optimizer, heuristic plans),
- report runtime distributions (median, p25/p75), win-rate, and perform significance testing (paired tests) under the same warm-cache protocol.

4 Conclusion

At month M16, UPB has delivered a a fully functional end-to-end ML-driven SPARQL optimization pipeline centered on Tentris. This progress includes the completion of **Milestone MS5 (M16) – Embeddings for SPARQL Queries** via the implemented query representation and embedding pipeline described in Section 2.1–2.2. The project has implemented:

(i) query representations based on structural features and a Transformer-based variable embedding module, and

(ii) a DRL training loop using a custom Gymnasium environment, Maskable PPO with action masking, and a FastAPI/Queue-based communication bridge between Tentris and the agent. Reward is derived from negative runtime using a **two-run warm-cache protocol**: the query is executed twice with the same plan and the second runtime is used as reward. The current datasets in active use are LinkedGeoData, Wikidata, SWDF, and DBpedia, and **train/validation/test splits have been created and balanced for all four datasets**.

A key challenge identified at this stage is the structural mismatch between real-world workloads, which are dominated by star joins, and Tentris's WCOJ execution behavior. This often collapses many cases into effectively a single join-variable decision, resulting in a very small action space. The main remaining work toward the Eurostars objectives and a VLDB-quality scientific evaluation is to:

(i) Enhance join-structure-aware dataset preparation and join counting,

(ii) Stabilize training on workloads with multiple meaningful join decisions,

(iii) Broaden the controllable optimization decisions beyond the "single join variable" setting where needed, and

(iv) Perform comprehensive test-time benchmarking against Tentris baselines under controlled warm-cache experimental conditions.

References

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, page 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] Alexander Bigerl, Felix Conrads, Charlotte Behning, Mohamed Ahmed Sherif, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. Tentris – a tensor-based triple store. In Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web – ISWC 2020*, pages 56–73, Cham, 2020. Springer International Publishing.

[3] Farama Foundation. Gymnasium Documentation, 2026.

[4] Milos Jovanovik and Mirko Spasić. Benchmarking Virtuoso 8 at the Mighty Storage Challenge 2018: Challenge Results. In Davide Buscaldi, Aldo Gangemi, and Diego Reforgiato Recupero, editors, *Semantic Web Challenges - 5th SemWebEval Challenge at ESWC 2018, Heraklion, Greece, June 3-7, 2018, Revised Selected Papers*, volume 927 of *Communications in Computer and Information Science*, pages 24–35. Springer, 2018.

[5] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 6, 01 2014.

[6] Knud Möller, Tom Heath, Siegfried Handschuh, and John Domingue. Recipes for semantic web dog food—the eswc and iswc metadata projects. In *International Semantic Web Conference*, pages 832–841. Springer, 2007.

[7] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai, editors, *The Semantic Web: Trends and Challenges*, volume 8465 of *Lecture Notes in Computer Science*, pages 176–191. Springer International Publishing, 2014.

[8] Mirko Spasić and Milos Jovanovik. MOCHA 2017 as a Challenge for Virtuoso. In Mauro Dragoni, Monika Solanki, and Eva Blomqvist, editors, *Semantic Web Challenges - 4th SemWebEval Challenge at ESWC 2017, Portoroz, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 769 of *Communications in Computer and Information Science*, pages 21–32. Springer, 2017.

[9] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. LinkedGeoData: A Core for a Web of Spatial Open Data. *Semantic Web Journal*, 3:333–354, 01 2012.

[10] Claus Stadler, Muhammad Saleem, Qaiser Mehmood, Carlos Buil-Aranda, Michel Dumontier, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. Lsq 2.0: A linked dataset of sparql query logs. *Semantic Web*, 14(1):1–23, 2023.

[11] Wikidata contributors. Wikidata, 2024. Accessed: 24.02.2026.