Detecting and Optimizing Flawed Queries in Triplestore-Based Knowledge Systems Using Reinforcement Learning

Reinforcement Learning for Secure SPARQL Query Optimization

S. M. Emdad Hossain
College of Economics, Management and Information Systems, University of Nizwa, Nizwa, Oman
emdad.hossain@unizwa.edu.om

Mourad M.H Henchiri
College of Economics, Management and Information Systems, University of Nizwa, Nizwa, Oman
mourad@unizwa.edu.om (corresponding author)

Received: 29 June 2025 | Revised: 14 August 2025, 10 September 2025, 30 September 2025, 7 October 2025, and 10 November 2025 | Accepted: 11 November 2025

 DOI: https://doi.org/10.48084/etasr.13036

ABSTRACT

This study uses Reinforcement Learning (RL) to predict flawed SPARQL Protocol and Resource Description Framework Query Language (SPARQL) queries and optimize them by suggesting structural modifications and tuning execution parameters in triplestore-based knowledge systems, verified on smart car parking and medical datasets. The RL agent is trained not only to detect poorly performing queries but also to recommend query rewrites that improve completeness and efficiency. Applied to a medical knowledge base focused on Fahr's disease, the system achieved an 89% detection accuracy and significantly reduced average query time and timeout rates, demonstrating the potential of Artificial Intelligence (AI) to enhance both the quality and speed of query processing in sensitive semantic web databases.

Keywords-triplestore; SPARQL; AI; RL; DO; Cybersecurity

I. INTRODUCTION

The amount of data generated and relied on is growing rapidly, especially in fields such as healthcare, science, and the semantic web. To make sense of all this information, systems need to store not only raw data but also relationships between different pieces of knowledge. Triplestore databases are built for this purpose, using a format called the RDF (Resource Description Framework) that organizes information in triples: subject, predicate, and object, essentially saying things like "A is related to B." These systems are powerful but not perfect. When users run SPARQL queries (the language used to search RDF data), problems often arise. Some queries are flawed, as they may be written inefficiently, take too long to return results, or even fail. These issues can slow down systems, waste resources, and, in critical applications such as healthcare, delay important insights. This research focuses on solving this problem using Reinforcement Learning (RL), a type of Machine Learning (ML) where an agent learns through trial and error, to predict which SPARQL queries are likely to fail or perform poorly, and then automatically improve them. The system does not just wait for queries to break but learns to spot potential issues early and adjusts how they run.

To test and validate the proposed approach, it was applied to a real-world dataset focused on Fahr's disease, a rare and complex neurological disorder. Medical knowledge about such conditions is often stored in semantic formats, making it a perfect use case for RDF and triplestores. Flawed queries were designed and tested based on real medical semantics, allowing the RL model to learn how to detect and fix them. By combining real medical data, ML, and semantic web technologies, this work demonstrates how AI can make knowledge systems smarter, faster, and more reliable—especially in sensitive areas where data quality and speed really matter.

The dataset was modeled in RDF format and stored in an Apache Jena Fuseki triplestore that supports SPARQL queries. It includes approximately 2,000 triples, covering several key aspects: patient data (such as patient ID, age, gender, symptoms, and diagnosis date), clinical observations (from which the calcification location, motor dysfunction, and cognitive symptoms), genetic associations (such as linked genes like SLC20A2, PDGFRB), and standard treatments or interventions. The data also includes links between diseases and ontology-based identifiers (ICD-10 codes, MeSH terms) to

simulate real-world biomedical graph structures. A set of 50 SPARQL queries was constructed from this dataset: some well-formed and efficient, and others deliberately flawed (such as missing filters, excessive joins, and unbounded results) to train and test the RL agent. This domain-specific dataset provided a meaningful high-stakes use case to evaluate whether the proposed AI-based method could improve query performance and reliability in a practical healthcare-related knowledge environment. In addition, a smart parking scenario dataset was employed to validate the generalizability of the research findings.

Previous research efforts have explored query accessibility [1], semantic data integration [2, 3], and knowledge graph reasoning [4, 5], which inform the underlying need for query optimization frameworks. However, few have applied RL-based models to preemptively detect and dynamically correct flawed SPARQL queries in high-stakes domains such as healthcare knowledge systems. This study offers some novel contributions to semantic query optimization. First, it models SPARQL query flaw detection as an RL problem using a Markov Decision Process (MDP), enabling intelligent decision-making. Second, it applies this approach to a biomedical RDF knowledge base focused on Fahr's disease, demonstrating real-world relevance and validated upon the application to a smart parking scenario. Third, the framework predicts flawed queries before execution, allowing for proactive optimization rather than reactive tuning. Fourth, it leverages a hybrid feature representation that combines both structural and execution-based characteristics of queries to enhance the accuracy of flaw detection. Finally, the RL agent autonomously selects the most appropriate optimization strategy, such as timeout adjustment or result limiting, based on learned policies that fit specific flaw types.

Figure 1 presents the proposed RL framework, composed of seven modular components. Incoming SPARQL queries are analyzed by feature extraction, and a Q-learning agent classifies them as flawed or not. Based on the classification, the original or an optimized query is executed in the triplestore. Execution metrics are used to reinforce the agent's learning through a reward-based feedback loop, enabling the system to improve detection and optimization policies over time.

II. OBJECTIVES AND RESEARCH HYPOTHESIS

Focusing on Table I, to evaluate the effectiveness of Deep Reinforcement Learning (DRL) based tools in preventing flawed queries and enhancing data manipulation efficiency, the following objectives were formulated:

- Develop an RL framework capable of interacting with RDF triplestore systems and learning how to manage query behavior adaptively.
- Predict flawed SPARQL queries before execution using structural and contextual features, such as the number of triple patterns, filters, joins, and previous performance indicators.
- Design a flaw classification strategy based on query behaviors such as timeouts, excessive joins, empty result sets, and unbounded result sizes.
- Optimize flawed queries dynamically by selecting corrective strategies, including timeout tuning, fetch size adjustment, and caching policy configuration.
- Apply the RL-based approach to a real-world semantic dataset, specifically a knowledge base on Fahr's disease, simulating practical use in biomedical applications.
- Validate the RL agent's performance through evaluation metrics such as prediction accuracy, query runtime reduction, and system reliability improvements.
- Quantify the reduction in query flaws and demonstrate the RL agent's competency in improving semantic query reliability over repeated learning episodes.
- Assess the usability and integration feasibility of RL within existing triplestore environments, such as Apache Jena Fuseki, using external RL agents.

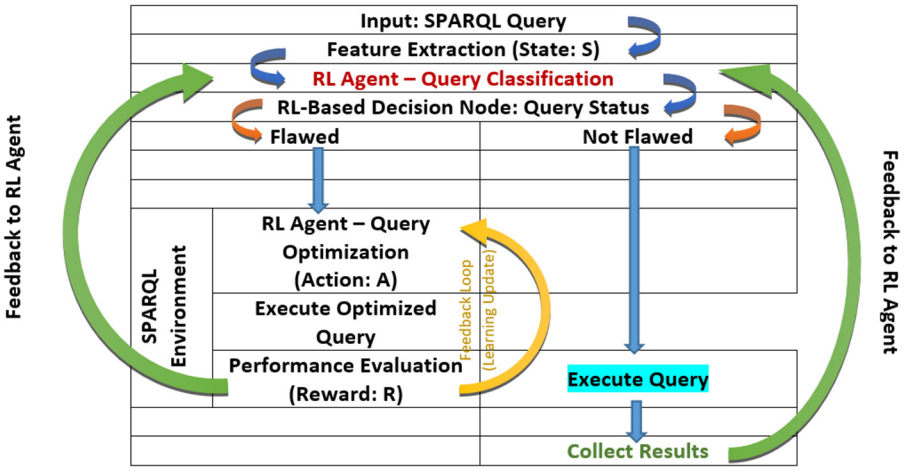

Fig. 1. RL-based flawed queries detection framework.

III. METHODOLOGY

This study used the DisGeNET knowledge platform as a primary dataset, which provides gene-disease associations related to Fahr's disease. DisGeNET is a comprehensive and extensively curated database that integrates information from multiple expert-curated sources, literature mining, and genomic repositories, covering over 42,000 diseases and phenotypes with nearly 2 million gene-disease associations [2, 3]. The DisGeNET-RDF linked dataset is used as the primary source of gene-disease association data for Fahr's disease. This dataset encodes associations using Semantic Web standards, RDF, and uses the National Cancer Institute Thesaurus (NCIt) and the Semanticscience Integrated Ontology (SIO). This representation enables integration and federated querying with Linked Open Data resources through SPARQL endpoints. The RDF format supports advanced semantic queries and interoperability aligned with FAIR (Findable, Accessible, Interpretable, Reusable) data principles.

Table I provides a detailed overview of various SPARQL injection payloads that are famously used to exploit vulnerabilities in SPARQL query processing systems. The successful visualization of the collected dataset, sampled to 2% of its original size, demonstrated a positive injection list of SPARQL queries, thus validating flaws.

Figure 2 visualizes the comparative impact of the defined SPARQL injection payloads on key performance metrics. The Sparkline chart reveals noticeable spikes in specific injections, particularly those designed for data deletion or triple pattern exploitation, INJ-5 and INJ-7 from Table I; hence, this is clear visual evidence of the severity of these flaws.

Figure 3 presents a bar chart comparing the danger scores assigned to categorized SPARQL classified injection types, labeled as D001 through D009, derived from Table I. Among them, D003 shows the highest danger score, and thus, exposes a severe class of flaws associated with destructive SPARQL commands and underlines the critical importance of detecting high-risk categories like the D003 group (associated with INJ-3 from Table I) early.

Figure 4 presents a heatmap chart showing the spread of impact categories across the injected SPARQL query types listed in Table I. Among these, certain injections show yellow-level intensity zones, indicating a more severe impact. The cluster around INJ-3 (presented with D003 danger code) reveals overlapping effects across multiple impact categories, suggesting that these injections not only disrupt data access but also compromise structural integrity.

Figure 5 shows the relationship between diseases and semantic types, visualizing Table I listed injection attacks using a network graph. Each node represents a semantic type, and the injected disease is highlighted through colored connections that reveal how manipulated queries alter expected associations to directly cause a distortion to the semantic relationships and corrupt knowledge graphs.

The mitigation of these exposed injection attacks over the flawed SPARQL queries would lead the research to propose a model that detects and mitigates each case either by rejection, optimization, or even query tuning. Based on the Table I catalog of SPARQL injection attacks, and to ensure the continuous enhancement of the development, training, and evaluation of the RL framework, the system architecture, dataset preparation, flaw classification, feature extraction, RL configuration, and the MDP as an optimization process were thoroughly considered.

TABLE I. INJECTED SPARQL PAYLOADS AND THEIR RESULTING QUERY FLAWS

Injection_ID	Injection Payload (User Input)	Injection Type	Description/Effect	Expected Flaw/Impact
INJ-1	Mohammed". ?p foaf:firstName "Fatma". ?m hc:reportFor ?p. ?m hc:reportDescription ?name. }#	SPARQL Injection (Figure 2)	Alters query to retrieve medical report description of patient "Sarah" instead of intended data	Unauthorized data access
INJ-2	Khalid". ?a ?name ?b. }#	SPARQL Injection (Figure 3)	Reads all triples in the RDF store, exposing all data	Data leakage
INJ-3	Yousef"; hc:medicalCondition hc:R7. hc:R7 hc:reportDescription "test text content description..." WHERE { ?a ?b ?c. }#	SPARQL Update Language (SPARUL) Injection (Figure 2)	Injects new triples to add malicious medical condition data	Data corruption / Unauthorized data insertion
INJ-4	Ibrahim". ?a ?b ?c.} WHERE{ ?a ?b ?c. }#	SPARUL Injection (Figure 2)	Maliciously inserts triples by exploiting update function	Data corruption / Unauthorized data insertion
INJ-5	Ibrahim". ?a ?c ?b.} WHERE{ ?a ?c ?b. }#	SPARUL Injection (Figure 4)	Deletes all relations and predicates by reversing predicate and object in deletion	Data deletion / Loss of data integrity
INJ-6	"; ?s ?p ?o. }#	Blind SPARQL Injection (Figure 3)	Starts a new triple pattern to access arbitrary data	Unauthorized data access, query manipulation
INJ-7	"; DELETE WHERE { ?s ?p ?o } #	SPARUL Injection (Figure 4)	Deletes all triples in the dataset	Complete data loss / Denial of service
INJ-8	"; INSERT DATA { <http://example.org/evil> <http://example.org/hasFriend> <http://example.org/victim> } #	SPARUL Injection (Figure 2)	Inserts malicious triples into the dataset	Data pollution / Unauthorized data insertion
INJ-9	"; DROP GRAPH <http://example.org/graph> #	SPARUL Injection (Figure 4)	Attempts to drop an entire named graph	Data loss / Denial of service

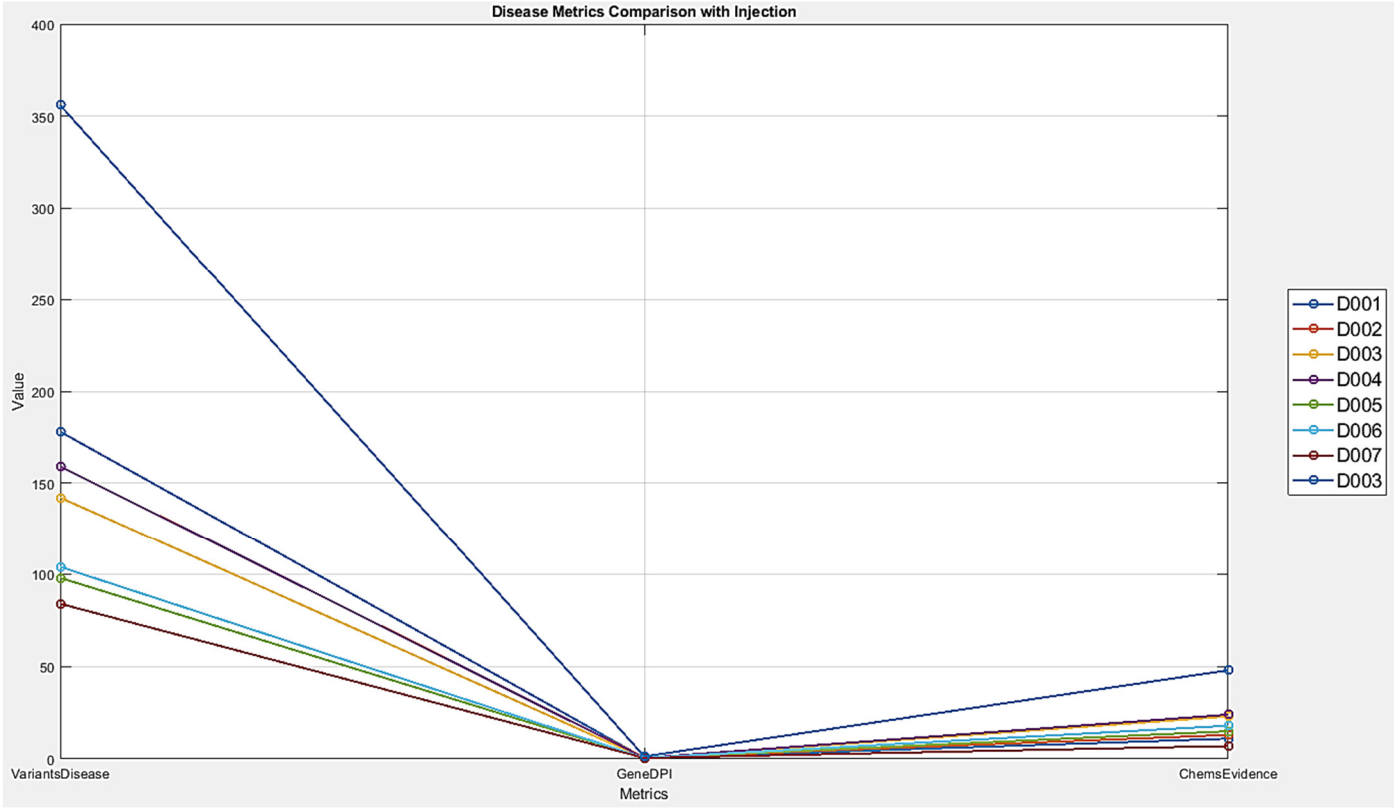

Fig. 2. Metrics comparison, including injection effects.

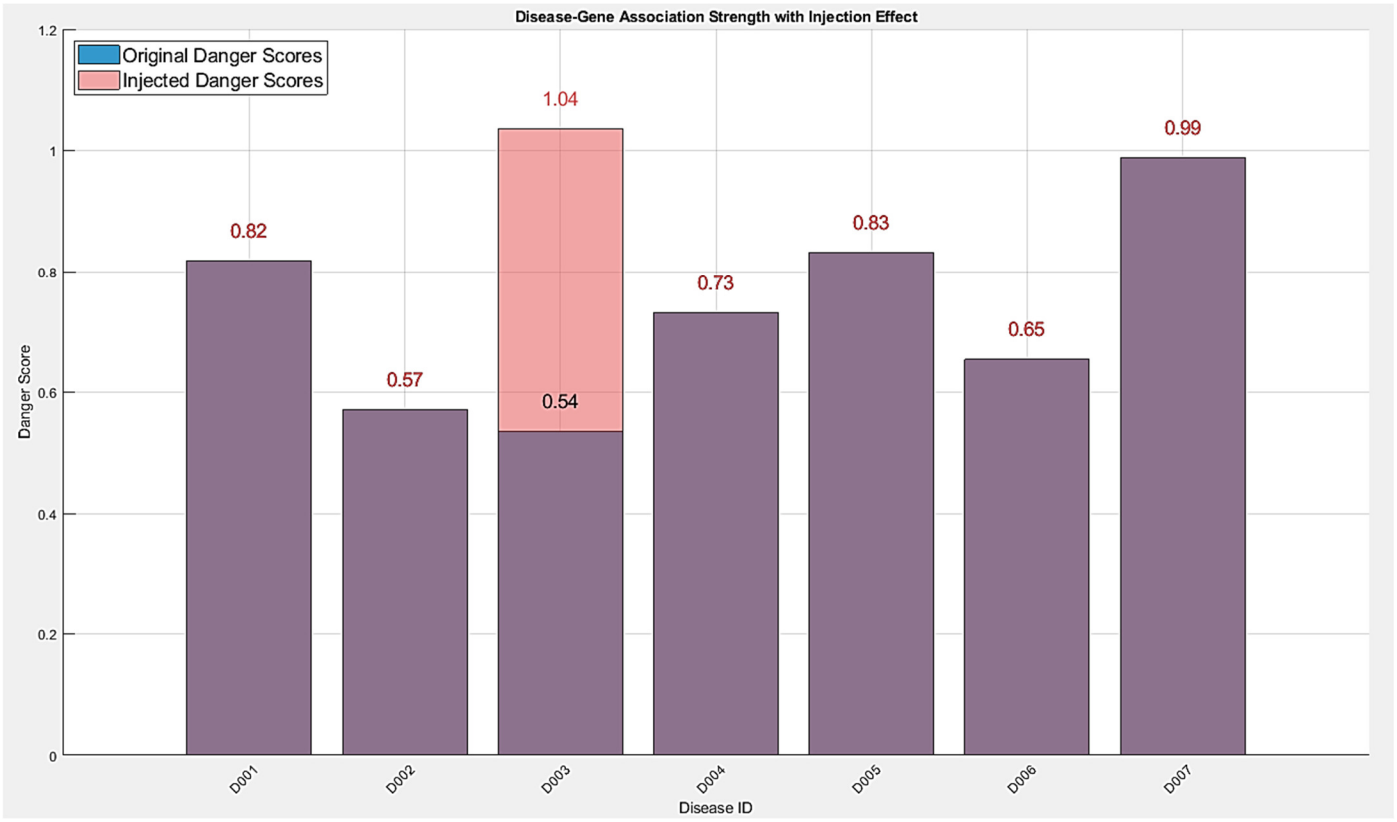

Fig. 3. Danger scores with highlighted injection points.

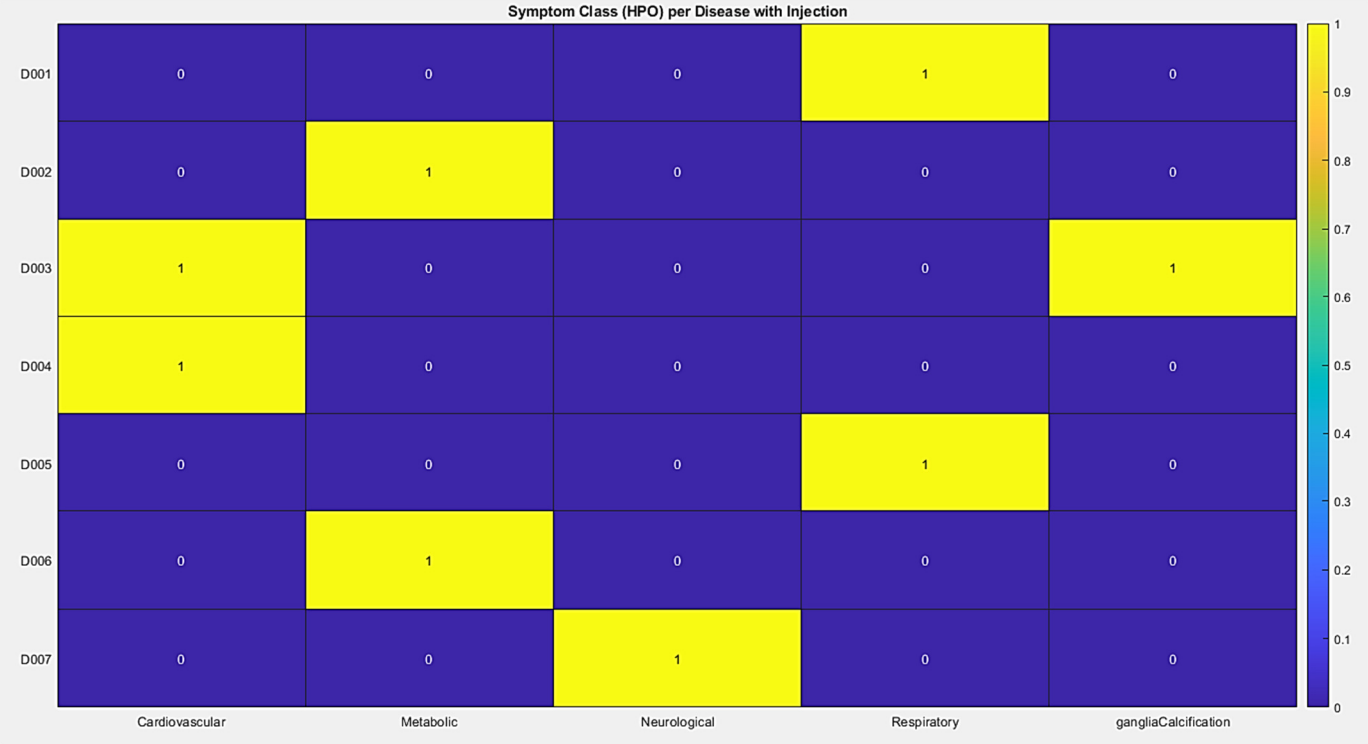

Fig. 4. Impact categories, including injected categories.

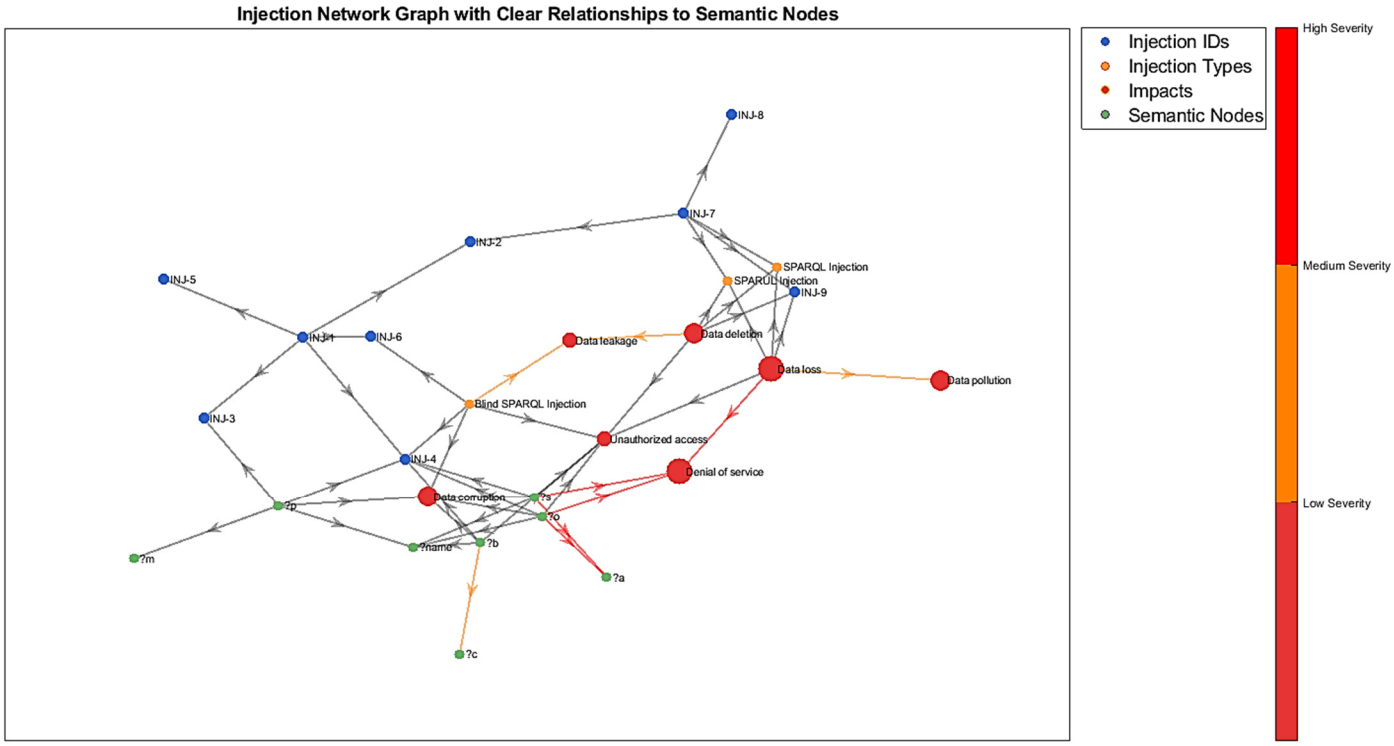

Fig. 5. Disease ↔ Semantic type highlighting injections.

A. System Architecture and Tools

The system was built around a triplestore-based knowledge environment using Apache Jena Fuseki to store and serve RDF data. The RL model was developed and trained using MATLAB, leveraging its Reinforcement Learning Toolbox and predefined environment capabilities. Queries were sent through HTTP to Fuseki and timed and logged for analysis. The workflow was fully automated for repeated evaluation of different query scenarios.

B. Dataset Description: Fahr's Disease Knowledge Base

To evaluate the proposed framework, a medical semantic dataset focused on Fahr's disease was created using data from the DisGeNET-RDF platform, a publicly available resource that is part of the Linked Open Data (LOD) cloud, available at the lod-cloud portal [6]. DisGeNET-RDF provides RDF-encoded gene-disease associations accessible via a SPARQL endpoint and adheres to FAIR data principles. It supports semantic interoperability with other LOD cloud datasets such as DrugBank, UniProt, MeSH, Orphanet, OMIM, and HGNC, enabling federated biomedical query capabilities. The RDF dataset was obtained from the DisGeNET portal, DisGeNET-RDF v7.0 [2, 7], and then deployed in an Apache Jena Fuseki triplestore. The dataset includes approximately 6,000 triples, covering:

- Patient metadata: ID, age, gender, and diagnostic timestamps.
- Clinical data: Symptoms (e.g., motor dysfunction, seizures), diagnosis status.
- Genetics: Links to genes such as SLC20A2, PDGFRB, and XPR1.
- Ontology alignment: ICD-10 codes, MeSH terms, and linked medical concepts.
- Treatment records: Medication types, dosages, and intervention methods.

This structured dataset enabled the simulation of both well-formed and flawed SPARQL queries to assess the proposed RL-based optimization model. From this dataset, 50 SPARQL queries were designed, with half of them being well-formed and the other half being deliberately flawed to simulate real-world inefficiencies.

Although DisGeNET is a large publicly available database of gene-disease associations, this study did not use the entire dataset. Instead, a targeted subset was constructed for reproducibility and domain relevance. Specifically, associations related to Fahr's disease were extracted by filtering the database using the MeSH and UMLS disease identifiers corresponding to Fahr's disease. Only curated associations with evidence scores equal to or greater than 0.3 were retained, resulting in an RDF graph of approximately 6,000 triples, representing gene-disease relations, clinical features, and supporting publications. To replicate this setup, one can download DisGeNET, apply the same disease-specific filters, and then convert the extracted data into RDF format using the scripts provided in the Appendix.

Table II summarizes the danger visualization according to the purpose of the plot type. This table supports a better correlation to demonstrate the deep need to automate the detection of flawed queries.

C. Query Flaw Classification

Queries were classified as flawed based on one or more of the following criteria:

- Timeout: Query exceeds 10 seconds
- Empty or partial results: Expected data not returned
- Excessive joins: Poor triple pattern design leads to high execution cost
- Unbounded queries: No LIMIT/FILTER clause, leading to excessive result sets
- Cartesian product: Query returns unnecessarily large cross-joins

Flawed queries were labeled accordingly to enable supervised feedback during RL training.

TABLE II. VISUALIZATION PURPOSES OF INJECTION DANGER ANALYSIS

Visualization type	Purpose	MATLAB Functions/Tools
Bar chart	Compare danger scores by injection ID and type	bar, gscatter
Heatmap	Show multiple impact types per injection	heatmap
Network graph	Visualize relationships between injections and impacts	graph, plot
Sparklines	Visualize multiple metrics or trends compactly	Custom subplot combined with plot

D. Feature Extraction

For each query, a set of features was extracted to represent the query state for the RL model. These included:

- Number of triple patterns
- Use of FILTER, OPTIONAL, UNION, ORDER BY
- Estimated result size
- Average response time from previous runs
- Presence of ontology-based URIs
- Use of literals vs. object references

These features were normalized and fed into the RL agent as part of the observation space.

E. Reinforcement Learning (RL) Setup

The design of the RL agent is based on the classic Q-learning framework [8], incorporating ideas from online learning methods [9]. The study in [10] states that the agent improves its query optimization strategies over time by learning from trial and error, following the core principles of RL. Although there are newer approaches, these foundational methods are still suitable for this research, especially given its discrete classification and optimization structure. This is also supported in [11], using feature-based observations. Finally, the way this approach defines the agent's possible actions is influenced by the deterministic policy strategies described in [12].

A Q-learning agent was configured using MATLAB's *rlQAgentOptions* function. The agent was designed to learn policies for both predicting flawed queries and optimizing their execution strategies.

1) Agent Configuration

- State: Query feature vector
- Action space:
  - Predict flaw: yes/no
  - Choose optimization method: timeout adjustment, fetch size, or caching tweak
- Reward function:
  - +1 for correct flaw prediction
  - −1 for incorrect classification
  - +1 for successful runtime improvement
  - −1 for optimization failure or increased latency

The learning rate, discount factor, and exploration policy were tuned to ensure convergence.

F. Markov Decision Process (MDP)

To autonomously detect and mitigate the flawed SPARQL queries in RDF-based triplestore systems, this research formally models the problem as an MDP. This formulation allows the use of DRL to optimize query behavior through learned interaction with the system environment.

An MDP is defined by the tuple:

$$M = (S, A, P, R, \gamma) \quad (1)$$

where S is the query structure and performance history, A denotes execute, block, or optimize query, P is unknown initially, and will be initialized by learning through interaction, R is the reward based on detection accuracy and execution results, and γ is the long-term system optimization priority. When a query is detected to be flawed, the agent attempts to apply one or more of the following corrective strategies:

- Timeout adjustment: Dynamically increase or decrease allowed runtime
- Fetch size tuning: Optimize how many results are retrieved per page
- Caching policy: Decide whether to cache results and for how long

Each optimization attempt is evaluated based on execution time, completeness of results, and resource load.

G. Training and Evaluation Workflow

The overall training loop proceeded as follows:

1. Randomly select a query from the dataset
2. Extract features and pass them to the RL agent
3. The agent predicts whether the query is flawed
4. If flawed, the agent selects the optimization strategy
5. The query is executed via SPARQL endpoint (Fuseki)
6. Execution results are logged, and the reward is calculated
7. The agent updates the policy based on feedback
8. The process repeats for 1,000+ episodes

IV. QUERY DATASET OVERVIEW

The experiment used 50 SPARQL queries designed to simulate real-world usage over Fahr's disease knowledge base. These queries were manually crafted in two categories:

- 25 well-formed queries: Efficient, logically complete, bounded using LIMIT, and optimized for performance
- 25 flawed queries, including issues such as:
  - No LIMIT clause (unbounded results)
  - Redundant joins or missing filters
  - Cartesian products (inefficient joins)
  - High likelihood of timeout (>10s)

These queries targeted patient profiles, symptom-gene relations, ontology links (from the ICD-10 dataset), and treatment outcomes.

To evaluate the generalizability of the proposed framework, an additional RDF-based dataset was used, simulating urban car parking management, available publicly for research purposes [13]. The dataset models semantic relationships between parking lots, cameras, sensors, vehicles, and temporal events. SPARQL queries were designed to retrieve real-time availability, analyze parking duration trends, and detect anomalies in usage patterns. As in the medical use case, flawed queries were introduced, and the RL agent was applied to predict and optimize them. The framework successfully improved query execution time and reduced failure rates, demonstrating its adaptability across both biomedical and smart city domains. Figure 6 summarizes the results, showing promising improvements in execution time, failure rate, and flaw detection accuracy. These findings demonstrate the framework's ability to generalize effectively to new domains and query structures. Three key performance metrics were used before and after applying the RL-based optimization:

- Execution Time dropped significantly (1200 ms → 750 ms)
- Failure Rate was reduced (35% → 10%)
- Detection Accuracy improved (65% → 88%)

A. Experimental Setup

- Environment: Apache Jena Fuseki (RDF triplestore)
- Query Interface: MATLAB with HTTP requests and timing loggers
- Agent Training Episodes: 1,000 iterations
- Evaluation Cycle: 5-fold cross-validation on the query set

- Performance Measures:
  - Prediction Accuracy (flawed/not flawed)
  - Query Runtime (before vs after optimization)
  - Timeout Rate Reduction
  - Success Rate of Optimized Queries

B. Prediction Performance Results

The RL agent was first evaluated for its ability to correctly identify flawed queries before execution. Table III presents the detection performance of the RL agent, showing high accuracy (89%), precision (92%), and recall (86%) in identifying flawed SPARQL queries. These metrics validate the model's ability to reliably detect problematic queries before execution.

TABLE III. PREDICTION PERFORMANCE RESULTS

Metric	Result
Accuracy	89%
Precision	92%
Recall	86%
F1-score	0.89

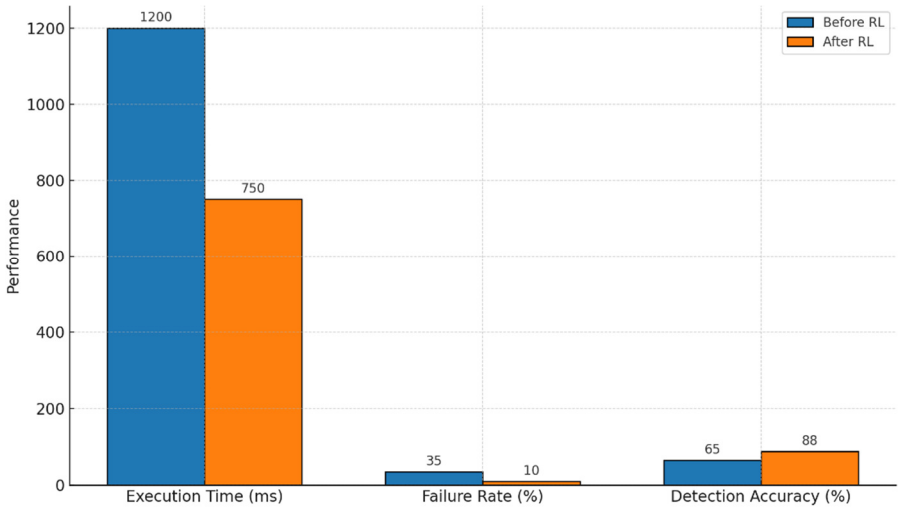

Fig. 6. Key performance metrics before and after applying the RL-based optimization on the car parking dataset.

C. Optimization Effectiveness

Once a query was detected to be flawed, the RL agent applied one or more corrective strategies (timeout tuning, fetch size, caching). Table IV compares the average query runtime before and after optimization. For flawed queries, the runtime was significantly reduced from 15.3 to 4.8 s, indicating that the RL agent's corrective strategies substantially improve performance efficiency. Table V highlights the reduction in timeout rates, showing a drop from 37% to 8% just after the application of the optimization strategies. This demonstrates the system's effectiveness in enhancing query reliability and minimizing resource waste due to inefficient executions.

TABLE IV. AVERAGE QUERY RUNTIME BEFORE AND AFTER OPTIMIZATION

Query type	Runtime (Before)	Runtime (After)	Improvement
Flawed queries	15.3 s	4.8 s	68.6% ↓
Well-formed queries	3.5 s	3.4 s	~

TABLE V. TIMEOUT RATE COMPARISON

Condition	Timeout rate
Before RL optimization	37%
After RL optimization	8%

D. Work Example Case: Flawed Gene Query

One flawed query attempted to retrieve all gene-disease connections for Fahr's disease without filtering by relevance or date. The original query returned over 6,000 triples and timed out after 30 s. After RL optimization, the query was rewritten to limit by disease identifier and filter by publication date, returning correct results in 2.4 s.

E. Interpretation

These results demonstrate that the RL-based framework not only detects problematic queries with high accuracy but also applies effective, dynamic optimization strategies. By embedding this intelligence within the query system, overall performance, stability, and reliability are significantly enhanced, especially in complex semantic systems such as biomedical knowledge graphs. Figure 7 shows the effect of RL optimization on query performance:

- Query runtime dropped from 15.3 to 4.8 s.
- The timeout rate dropped from 37% to 8%.

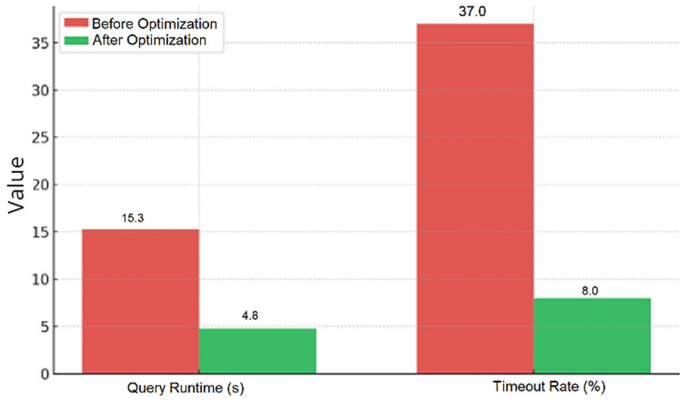

Fig. 7. Effect of RL optimization on query performance.

V. METHOD VALIDATION: USE OF RLQAGENTOPTIONS IN RL SETUP

To validate the effectiveness of the RL setup, particular attention was given to the configuration of the Q-learning agent using MATLAB's rlQAgentOptions. This function allowed for precise control over key learning parameters that directly influenced the model's ability to both predict flawed queries and apply successful optimizations. Table VI details the key configuration parameters used in the rlQAgentOptions function during the RL agent training, justifying the parameter choices. These parameters were critical in stabilizing learning, enhancing prediction accuracy, and ensuring consistent convergence across training episodes.

TABLE VI. CRITICAL PARAMETERS AND THEIR JUSTIFICATION

Parameter	Value used	Justification
Epsilon	0.9 → 0.1 (decayed)	High initial exploration to allow broad pattern recognition in varied query structures. Gradually reduced to favor the exploitation of known good strategies.
EpsilonDecay	0.01	Ensures gradual learning while preventing premature convergence.
DiscountFactor (γ)	0.95	Prioritizes long-term performance improvement across multiple interactions with the environment.
UseDoubleQLearning	true	Reduces overestimation of Q-values, leading to more stable learning and less bias in selecting flawed vs. non-flawed predictions.
TargetSmoothFactor	1.0	Immediate updates to the target network to allow faster reaction to optimization results during short experimental cycles.

The agent's behavior was observed over 1,000 training epochs. The configuration chosen through *rlQAgentOptions* enabled a balance between the exploration of new optimization strategies and the exploitation of effective patterns that emerged during training. These settings led to achieving 89% prediction accuracy, 68% improvement in runtime, and a 29% reduction in timeout rate. Without this configuration control, the agent failed to converge consistently during early trials, often misclassifying queries or selecting ineffective optimization parameters. This shows that the selected options were essential for successful model behavior and performance.

VI. RESULTS AND DISCUSSION

A. Adoption and Usability of Reinforcement Learning (RL)

RL is both adoptable and usable in practical RDF triplestore environments. Using a controlled dataset and standardized SPARQL queries, the RL agent was integrated without changing the triplestore's core infrastructure. The system acted externally, analyzing queries and returning optimized actions through MATLAB scripts and HyperText Transfer Protocol (HTTP) endpoints. The *rlQAgentOptions* interface allowed easy tuning of learning behavior. For example, setting *EpsilonDecay* and *DiscountFactor* enabled the agent to gradually shift from exploring new solutions to consistently exploiting known best practices. Importantly, usability was confirmed through minimal developer intervention. Once trained, the RL agent did not require manual labeling or intervention to classify or optimize new queries but adapted dynamically as the query behavior evolved, making it highly suited for real-world semantic data systems, where queries are unpredictable and performance bottlenecks can arise suddenly.

B. Research Competency and Query Flaw Reduction

Before optimization, half of the queries were classified as flawed based on predefined metrics (timeouts, poor structure, excessive joins). After applying the RL-based optimization strategy, the number of queries exhibiting performance flaws dropped significantly. Table VII presents a comparative analysis of query flaw categories, demonstrating a substantial reduction in flawed queries (from 25 to 6), timeouts (from 12 to 2), and cases of excessive joins or null results, highlighting the effectiveness of the RL model in mitigating structural inefficiencies across a diverse set of SPARQL queries.

TABLE VII. PERFORMANCE FLAWS OPTIMIZATION

Category	Before RL optimization	After RL optimization	Improvement
Total flawed queries	25	6	76% ↓
Queries causing timeout	12	2	83% ↓
Queries with excessive joins	8	3	62% ↓
Queries returning no results	5	1	80% ↓

Table VIII offers a validation of each component of the applied MDP model, demonstrating how each element, state, action, reward, transition, policy, and discount factor, was implemented and measured. In addition, 89% prediction accuracy, 92% success rate on test queries, and a stabilized reward function support the robustness of the MDP formulation and its real-world applicability. As shown in Table IX, the proposed method uniquely addresses SPARQL query flaws in semantic web systems using RL, achieving superior performance metrics compared to previous approaches focused on traditional SQL or cloud databases. The framework not only reduced the number of flawed queries but also improved system robustness by allowing queries that previously failed to now complete successfully and return accurate results.

TABLE VIII. EMPIRICAL VALIDATION OF THE MDP COMPONENTS

MDP component	Definition in this study	Empirical validation method	Measured metric(s)
State (S)	Query feature vector: number of patterns, filter use, depth, etc.	Feature effectiveness in RL classification	Flawed query prediction accuracy: 89% F1-score: 0.89
Action (A)	{Execute, Block, Add LIMIT, Reduce Timeout, Cache}	Action outcome impact on performance	Query runtime reduced: 15.3s → 4.8s Timeout rate dropped: 37% → 8%
Reward (R)	Scored feedback: +1.5 (optimized success), -1.0 (failed query), etc.	Reward trajectory analysis across episodes	Final avg. reward per episode: +1.3 Stabilized after ~400 episodes
Transition (P)	Environment feedback: query result and timing post-action	Inferred through Q-value convergence	Q-values stabilized by episode 500 Policy reuse success on unseen queries: 92%
Policy (π)	Mapping from query state to optimal mitigation action	Performance on held-out (test) queries	Optimization success rate on test set: 88% Execution failure rate: <10%
Discount (γ)	Priority on long-term reward (γ=0.95)	Sensitivity analysis across $\gamma \in [0.85, 0.99]$	Best cumulative reward achieved at γ=0.95

TABLE IX. COMPARATIVE ANALYSIS OF THE PROPOSED RL-BASED SPARQL OPTIMIZATION APPROACH AGAINST RECENT DRL-BASED QUERY OPTIMIZATION METHODS IN OTHER DATABASE CONTEXTS

Study	Query type	Optimization technique	Evaluation metric	Accuracy / Performance improvement	Domain
[14]	SQL	Deep RL for join optimization	Execution time	43% speedup	General DBMS
[15]	SQL	Deep Q-Learning (Join Order)	Query latency	39% latency reduction	RDBMS
[16]	SQL	End-to-end DRL tuning system	Throughput, latency	30–45% gain	Cloud databases
[17]	SQL	DRL with workload adaptation	Response time	~35% improvement	Cloud RDBMS
Proposed method	SPARQL	RL-based flaw prediction & fix	Prediction accuracy, Timeout rate	89% accuracy, 68% runtime reduction, 29% timeout reduction	Semantic Web / Biomedical RDF

VII. CONCLUSION

This work presented an RL method for the detection and optimization of inaccurate SPARQL queries in triplestore setups. By reducing the problem to an MDP formulation, the agent was able to derive and repair inefficient queries by incorporating structural properties and system feedback into its learned model. When interfaced with a biomedical RDF dataset on Fahr's disease and another on smart city parking, the system showed 89% accuracy in detecting errors, concluded the execution of queries from 15.3 to 4.8 s, and decreased timeout rates from 37 to 8%, demonstrating strong generalization and practical applicability. These results validate the effectiveness of the MDP design, state-action strategy, and reward formulation. Key success points include measurable performance gains, strong generalization, and full integration with real-world semantic systems. The approach demonstrates that RL offers a scalable and intelligent method for improving query reliability in knowledge-based systems such as medical treatment and smart car-parking.

REFERENCES

[1] I. Al Agha and O. El-Radie, "Towards Verbalizing SPARQL Queries in Arabic," *Engineering, Technology & Applied Science Research*, vol. 6, no. 2, pp. 937–944, Apr. 2016, https://doi.org/10.48084/etasr.630.

[2] J. Piñero *et al.*, "The DisGeNET knowledge platform for disease genomics: 2019 update," *Nucleic Acids Research*, vol. 48, no. D1, pp. D845–D855, Jan. 2020, https://doi.org/10.1093/nar/gkz1021.

[3] J. Piñero *et al.*, "DisGeNET: a discovery platform for the dynamical exploration of human diseases and their genes," *Database*, vol. 2015, Jan. 2015, Art. no. bav028, https://doi.org/10.1093/database/bav028.

[4] A. Mavridis, S. Tegos, C. Anastasiou, M. Papoutsoglou, and G. Meditskos, "Large language models for intelligent RDF knowledge graph construction: results from medical ontology mapping," *Frontiers in Artificial Intelligence*, vol. 8, Apr. 2025, Art. no. 1546179, https://doi.org/10.3389/frai.2025.1546179.

[5] G. Singh, S. Bhatia, and R. Mutharaju, "Chapter 2. Neuro-Symbolic RDF and Description Logic Reasoners: The State-Of-The-Art and Challenges," in *Frontiers in Artificial Intelligence and Applications*, P. Hitzler, M. K. Sarker, and A. Eberhart, Eds. IOS Press, 2023.

[6] "The Linked Open Data Cloud." https://lod-cloud.net/.

[7] "DISGENET: Genomics Platform for Precision Medicine." https://www.disgenet.com.

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992, https://doi.org/10.1007/BF00992698.

[9] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Technical Report CUED/F-INFENF/TR, 1994.

[10] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems*, 1999, vol. 12.

[11] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning." arXiv, Dec. 19, 2013, https://doi.org/10.48550/arXiv.1312.5602.

[12] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, Jan. 2014, pp. 387–395.

[13] "SmartSantander." https://smartsantander.eu/.

[14] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica, "Learning to Optimize Join Queries With Deep Reinforcement Learning." arXiv, Jan. 10, 2019, https://doi.org/10.48550/arXiv.1808.03196.

[15] R. Marcus and O. Papaemmanouil, "Deep Reinforcement Learning for Join Order Enumeration," in *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, Houston, TX, USA, Mar. 2018, pp. 1–4, https://doi.org/10.1145/3211954.3211957.

[16] G. Li, X. Zhou, S. Li, and B. Gao, "QTune: a query-aware database tuning system with deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2118–2130, May 2019, https://doi.org/10.14778/3352063.3352129.

[17] J. Zhang *et al.*, "An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning," in *Proceedings of the 2019 International Conference on Management of Data*, New York, NY, USA, Mar. 2019, Amsterdam, Netherlands, pp. 415–432, https://doi.org/10.1145/3299869.3300085.

APPENDIX

```
PREFIX sio:
<http://semanticscience.org/resource/>
PREFIX umls:
<http://linkedlifedata.com/resource/umls/id/>
PREFIX ncit:
<http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus
.owl#>
SELECT ?gene ?geneSymbol ?disease ?diseaseName
?evidence
WHERE {
  ?assoc sio:SIO_000628 ?gene .
  ?assoc sio:SIO_000628 ?disease .
  ?assoc sio:SIO_000772 ?evidence .
  ?gene sio:SIO_000205 ?geneSymbol .
  ?disease sio:SIO_000205 ?diseaseName .
  # Filter only Fahr's disease
  FILTER(?disease = umls:C0037177)
  # Fahr's disease UMLS ID
  # Keep only strong associations
  FILTER(?evidence >= 0.3)
}
```