tools4RDF: A Python toolkit for working with RDF data

Sarath Menon[1,2], **Abril Azócar Guzmán**[3], **Osamu Waseda**[1], **Stefan Sandfeld**[3], and **Tilmann Hickel**[4]

1 Max Planck Institute for Sustainable Materials, 40237 Düsseldorf, Germany **2** ICAMS, Ruhr University Bochum, 44780 Bochum, Germany **3** Institute for Advanced Simulations – Materials Data Science and Informatics (IAS-9), Forschungszentrum Jülich GmbH, 52425 Jülich, Germany **4** Bundesanstalt für Materialforschung und -prüfung, 12489 Berlin, Germany

DOI: 10.21105/joss.09482

Software

- Review
- Repository
- Archive

Editor: Chris Vernon

Reviewers:

- @r0hin
- @fcdimitr

Submitted: 20 November 2025
Published: 26 January 2026

Summary

tools4RDF is a lightweight Python framework designed to simplify working with RDF-based ontologies and data models. It allows one or more ontologies to be parsed and represented as Python classes, making it easier to navigate and explore their structure through features like autocompletion in interactive environments such as Jupyter notebooks. The framework preserves key semantic details such as domain and range information, which can then be used programmatically within Python workflows. A central goal of the tool is to make querying knowledge graphs with SPARQL more accessible to users without deep expertise in semantic web technologies. This is achieved through a programmatic interface that abstracts away much of the complexity involved in writing SPARQL queries. The overall workflow of the approach is illustrated in Figure 1, showing how tools4RDF parses ontologies, constructs a network representation, and generates SPARQL queries to retrieve data from knowledge graphs. This design lowers the barrier to entry for domain scientists and developers unfamiliar with RDF, while still exposing enough control for advanced use cases. Originally developed for ontology-driven data integration in materials science, tools4RDF is ontology-agnostic and can be used across a wide range of scientific domains.

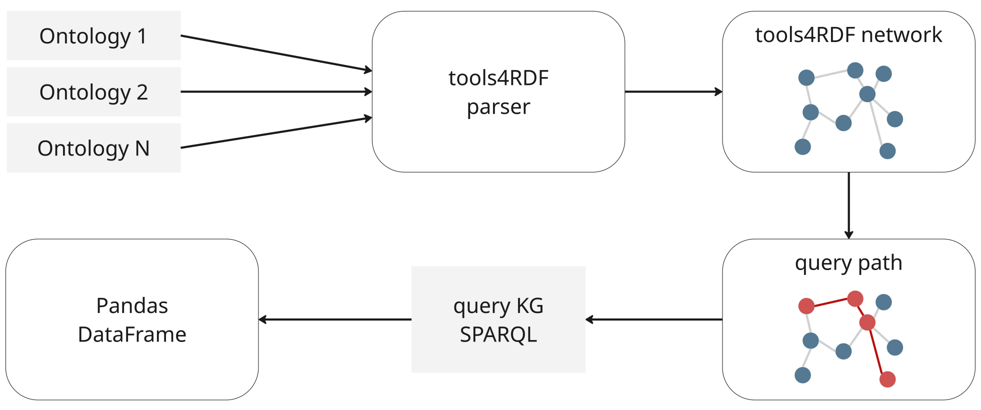

Figure 1: Illustration of the working principle of tools4RDF. One or more ontologies are parsed using the tools4RDF parser and organized into a network representation. When the user specifies a source and destination, one or more connecting paths between these nodes are identified. Based on the selected pathway, a corresponding SPARQL query is generated, with optional constraints that can be applied. The knowledge graph is then queried, and the results are returned as a Pandas DataFrame.

Statement of need

Knowledge graphs (Gutierrez & Sequeda, 2020), coupled with formal ontologies, offer a promising technical approach for implementing the FAIR Guiding Principles (Wilkinson et al., 2016) for research data management. They support semantic interoperability, especially for aggregation and description of heterogeneous data (Vogt et al., 2025). Accessing and querying these graphs typically relies on SPARQL (W3C SPARQL Working Group, 2013), the standard query language for RDF data (W3C SPARQL Working Group, 2013). However, despite its potential, SPARQL remains difficult to use for many domain scientists due to unfamiliar syntax, steep learning curve, and the absence of a user-friendly tool stack (Khamparia & Pandey, 2017). Libraries such as RDFLib (Krech et al., 2025) offer core functionality for RDF parsing and graph storage but require users to write SPARQL queries and manipulate low-level graph objects. Tools like Protégé (Musen, 2015) and SemTK (Cuddihy et al., 2018) provide graphical interfaces for ontology exploration, but they focus on GUI-based interaction and lack integration with the Python ecosystem commonly used in scientific computing.

tools4RDF fills this gap by providing a Python-native interface for ontology exploration and query construction. By ingesting one or more ontologies and converting them into navigable Python classes, allowing users to navigate class hierarchies, properties, and relationships through autocompletion. This enables users to perform predicate-based graph traversal and generate SPARQL queries programmatically, without requiring manual SPARQL scripting. The framework tools4RDF supports a code-first workflow that allows domain scientists to explore semantic data in familiar environments such as Jupyter notebooks. Query results can be parsed directly into Pandas (McKinney, 2010) dataframes, facilitating integration with existing data processing pipelines.

Beyond simplifying ontology interaction, this programmatic access to ontologies may also be useful for integrating semantic data into applications involving agentic AI or large language models (LLMs). Since tools4RDF guarantees the generation of structurally valid SPARQL queries that conform to the ontology schema, the task for the LLM is reduced to identifying source and destination terms, rather than constructing complete query syntax. This approach could eliminate common failure modes such as hallucinated entities (Emonet et al., 2025), invalid property paths, and syntax errors (Meyer et al., 2025), ensuring that every generated query is executable and semantically correct.

Key features

Parser for ontologies

tools4RDF includes a parser for ontologies that extracts classes, datatype properties, annotation properties, and hierarchical relationships such as subclasses and subproperties. It also parses and preserves domain and range definitions. The ontology is then represented as Python classes, which can be used programmatically in scripts and workflows to provide contextual knowledge.

Multiple ontologies can be parsed separately and their extraction can be combined using the + operator. Links between classes from different ontologies can be defined in external files or added programmatically.

Automated creation of SPARQL queries

Once the ontology terms are parsed, tools4RDF can generate SPARQL queries automatically if the source and destination terms are specified. Autocompletion helps users find relevant terms without needing detailed knowledge of the ontology. Internally, tools4RDF realises the ontology as a graph and traverses it to determine valid paths between terms. These paths are then converted to triples, which are then used in a SPARQL query.

Advanced features include: (i) restricting queries based on datatype properties, (ii) applying restrictions to queries using <, >, <=, >=, == operators (iii) chaining restrictions and constructing complex queries using logical operators such as & and |, and (iv) manually specifying intermediate terms for fine-grained control.

A simple SPARQL query, and the corresponding programmatic query creation. SPARQL code for querying a Person, and the corresponding familyName from DBpedia (Auer et al., 2007) using the FOAF ontology, and showing the first ten results is as follows:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?Person ?familyNamevalue
WHERE {
    ?Person foaf:familyName ?familyNamevalue .
   { ?Person rdf:type foaf:Person . }
}
LIMIT 10
```

Corresponding code example demonstrating how tools4RDF can be used to construct the SPARQL query:

```python
from tools4rdf import OntologyNetwork
onto = OntologyNetwork('http://xmlns.com/foaf/0.1/')
df = onto.query(
    kg = 'https://dbpedia.org/sparql',
    source = onto.terms.foaf.Person,
    destinations = onto.terms.foaf.familyName,
    limit=10,
)
```

An `OntologyNetwork` object is first created using a selected ontology. The `query` function takes four arguments: `kg` specifies the SPARQL endpoint (which may also be local), `source` and `destination` define the two endpoints connected by the query, and `limit` determines the number of records to return. The query results are returned as a Pandas `DataFrame`.

tools4RDF employs a graph-based approach internally to ensure generated queries are ontologically valid. When a user specifies source and destination terms, the tool constructs a NetworkX (Hagberg et al., 2008) graph representation of the ontology and computes the paths between the specified nodes. This ensures that only valid connections through the ontology structure are used. When multiple valid paths exist, the paths are ordered by increasing length. The user can retrieve the required number of paths or manually specify intermediate terms to select a specific path. The generated queries preserve domain and range constraints from the ontology, and type assertions are automatically included to ensure RDF-compliant results. Additionally, filter operations are validated against datatypes.

More examples, including the demonstration of the use of comparison and logical operators are available in the documentation.

Ontology visualization

tools4RDF also includes a simple ontology visualisation tool. It uses Graphviz (Gansner & North, 2000) to generate quick visual representations of ontologies within Jupyter notebooks.

Current limitations

Translation of SPARQL functionality to Python

tools4RDF currently supports a subset of SPARQL operations designed to cover common query patterns, making it accessible to users without prior experience in semantic web technologies. Advanced query features are not yet supported. However, expert users can generate an initial query with tools4RDF and then extend or refine it manually, and execute it. We expect the user community to engage with the tool and contribute toward extending SPARQL support.

Reasoning

The queries generated by tools4RDF are based only on the information explicitly present in the input RDF-based files. No description logic reasoning is performed, so implicit relationships are not taken into account. This limitation could be addressed by combining tools4RDF with reasoning tools such as HermiT (Glimm et al., 2014) as a preprocessing step to materialize inferences prior to query construction.

Additional details

tools4RDF is available under MIT license from the repository: github.com/OCDO/tools4RDF. The documentation, which includes installation instructions and examples, is available in the webpage.

Acknowledgements

This work is part of/supported by the consortium NFDI-MatWerk, funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the National Research Data Infrastructure – NFDI 38/1 – project number 460247524.

References

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A nucleus for a web of open data. *Proceedings of the 6th International the Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*, 722–735. https://doi.org/10.1007/978-3-540-76298-0_52

Cuddihy, P., McHugh, J., Williams, J. W., Mulwad, V., & Aggour, K. S. (2018). *SemTK: An ontology-first, open source semantic toolkit for managing and querying knowledge graphs.* https://arxiv.org/abs/1710.11531

Emonet, V., Bolleman, J., Duvaud, S., Farias, T. M. de, & Sima, A. C. (2025). *LLM-based SPARQL query generation from natural language over federated knowledge graphs.* https://doi.org/10.48550/arXiv.2410.06062

Gansner, E. R., & North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software: Practice and Experience*, *30*(11), 1203–1233. https://doi.org/10.1002/1097-024X(200009)30:11%3C1203::AID-SPE338%3E3.0.CO;2-N

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 Reasoner. *J. Autom. Reason.*, *53*(3), 245–269. https://doi.org/10.1007/s10817-014-9305-1

Gutierrez, C., & Sequeda, J. F. (2020). Knowledge Graphs: A Tutorial on the History of Knowledge Graph's Main Ideas. *Proceedings of the 29th ACM International Conference on*

Information & Knowledge Management, 3509–3510. https://doi.org/10.1145/3340531.3412176

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15). https://doi.org/10.25080/TCWV9851

Khamparia, A., & Pandey, B. (2017). Comprehensive analysis of semantic web reasoners and tools: A survey. *Educ Inf Technol*, *22*(6), 3121–3145. https://doi.org/10.1007/s10639-017-9574-5

Krech, D., Grimnes, G. A., Higgins, G., Car, N., Hees, J., Aucamp, I., Lindström, N., Arndt, N., Sommer, A., Chuc, E., Herman, I., Nelson, A., McCusker, J., Gillespie, T., Kluyver, T., Ludwig, F., Champin, P.-A., Watts, M., Holzer, U., … Stuart, V. (2025). *RDFLib* (Version 7.1.2). https://doi.org/10.5281/zenodo.6845245

McKinney, Wes. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Meyer, L.-P., Frey, J., Brei, F., & Arndt, N. (2025). *Assessing SPARQL capabilities of large language models*. https://doi.org/10.48550/arXiv.2409.05925

Musen, M. A. (2015). The protégé project: A look back and a look forward. *AI Matters*, *1*(4), 4–12. https://doi.org/10.1145/2757001.2757003

Vogt, L., Strömert, P., Matentzoglu, N., Karam, N., Konrad, M., Prinz, M., & Baum, R. (2025). Suggestions for extending the FAIR Principles based on a linguistic perspective on semantic interoperability. *Sci Data*, *12*(1). https://doi.org/10.1038/s41597-025-05011-x

W3C SPARQL Working Group. (2013). *SPARQL 1.1 query language*. World Wide Web Consortium. https://www.w3.org/TR/sparql11-query/

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., … Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data*, *3*(1). https://doi.org/10.1038/sdata.2016.18