WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

# Agentic SPARQL: Evaluating SPARQL-MCP-powered Intelligent Agents on the Federated KGQA Benchmark

Daniel Dobriy, Frederik Bauer, Amr Azzam, Debayan Banerjee, Axel Polleres

Arbeitspapiere zum Tätigkeitsfeld
Informationsverarbeitung, Informationswirtschaft und Prozessmanagement
*Working Papers on Information Systems, Information Business and Operations*

Nr./No. 01/2026
ISSN: 2518-6809

Herausgeber / Editor:
Department für Informationsverarbeitung und Prozessmanagement
Wirtschaftsuniversität Wien · Welthandelsplatz 1 · 1020 Wien
*Department of Information Systems and Operations · Vienna University of Economics and Business · Welthandelsplatz 1 · 1020 Vienna*

## Abstract

Standard protocols such as the Model Context Protocol (MCP) that allow LLMs to connect to tools have recently boosted "agentic" AI applications, which, powered by LLMs' planning capabilities, promise to solve complex tasks with the access of external tools and data sources. In this context, publicly available SPARQL endpoints offer a natural connection to combine various data sources through MCP by (a) implementing a standardised protocol and query language, (b) standardised metadata formats, and (c) the native capability to federate queries. In the present paper, we explore the potential of SPARQL-MCP-based intelligent agents to facilitate federated SPARQL querying: firstly, we discuss how to extend an existing Knowledge Graph Question Answering benchmark towards agentic federated Knowledge Graph Question Answering (FKGQA); secondly, we implement and evaluate the ability of integrating SPARQL federation with LLM agents via MCP (incl. endpoint discovery/source selection, schema exploration, and query formulation), comparing different architectural options against the extended benchmark. Our work complements and extends prior work on automated SPARQL query federation towards fruitful combinations with agentic AI.


## 1 Introduction

In recent years, Large Language Models (LLMs) have impacted nearly every industry and many aspects of daily life, prompting transformations in the Web's ecosystem (such as blurring the boundaries between 'search' and 'question answering') as well as motivating research focused on addressing their major limitations: hallucinations, bias, knowledge cut-offs, reasoning deficiencies and lacking explainability/interpretability [Matarazzo and Torlone, 2025]. A particular challenge, but also opportunity, in this context lies in the combination of LLMs with external, structured data sources, which has been addressed in different, complementary research strands:

**RAG & Text-to-Query.** Firstly, Retrieval-Augmented Generation (RAG), a technique of retrieving and injecting relevant and structured external knowledge into an LLM's model context [Lewis *et al.*, 2020], has been proposed and widely adopted in various forms. While RAG is often used in combination with textual embeddings and vector databases to map and lookup in relational or graph data most related items to a prompt by simple "top-k lookups" from a database in terms of vector similarity [Karpukhin *et al.*, 2020], another common approach to retrieval is text-to-query that (i) translates LLM prompts or contexts to database queries, with, most prominently, Text-to-SQL [Liu *et al.*, 2025] or Text-to-SPARQL [Banerjee *et al.*, 2022; D'Abramo *et al.*, 2025] as central components of Knowledge-Graph-Question-Answering (KGQA) systems, and (ii) feeds the query results, directly or via templated result-to-text translations, back to the LLM.

**Agentic AI.** More recently and significantly boosting the base idea of mostly hard-coded RAG pipelines, the Model Context Protocol (MCP) has been proposed as an open standard for context management in "agentic" settings.[1] At its core [Anthropic, 2025], besides server-provided resources (contextual data), prompts (templated workflows), and tools (executable methods), the main novelty over hard-coded RAG pipelines lies in that the standard agent protocol and standard reasoning/planning capabilities within modern LLMs, such as Reason+Act (ReACT) [Yao *et al.*, 2023].

**Federated Querying.** Use cases on agentic AI in the context of querying databases explored so far in the literature have mostly focused on interactions with *single* (e.g. relational) databases and endpoints: while tasks such as schema exploration, and query formulation over single databases have been successfully delegated to such agents, the power of exploiting *federated* query endpoints in agentic AI applications has received less attention so far. Yet, in terms of APIs potentially usable for agentic AI pipelines, W3C standard recommendations such as the Resource Description Framework (RDF) [Cyganiak *et al.*, 2014] along with the SPARQL Protocol and RDF Query Language (SPARQL) [Harris and Seaborne, 2013] already provide standardized API access to federated graph data, as primary standards for publishing and querying linked open data (LOD) and FAIR data [Wilkinson *et al.*, 2016]. Additionally, SPARQL 1.1 introduced support for *federated querying* natively in the standard, through the SERVICE operator [Harris and Seaborne, 2013], as well as dedicated RDF vocabularies to describe and advertise SPARQL endpoints [Williams, 2013; Böhm *et al.*, 2011]. The availability of such standards has motivated research on various aspects of federated SPARQL query processing, addressing challenges such as endpoint discovery, query decomposition, and query planning [Schwarte *et al.*, 2011b; Acosta *et al.*, 2011], including the definition of benchmarks such as FedBench [Schmidt *et al.*, 2011b].

**Towards "Agentic SPARQL".** As such, flexibly integrating SPARQL endpoints and query federation into agentic LLM applications seems a natural next step towards allowing agents to integrate federated graph data served via SPARQL endpoints. In the present paper, we will refer to *Agentic SPARQL* as the combination of agentic AI, and federated SPARQL querying capabilities providing agents with the access to the Web of Data. In this context, we see the following main open challenges, which we aim to address in our work:

**C1: Interface Heterogeneity.** "Endpoints" may range from lower level (below-SPARQL) interfaces like RDF dumps only, to triple-pattern based interfaces and, finally, to "full" SPARQL endpoints. Azzam et al.[Azzam *et al.*, 2024] have recently characterized and reviewed these different interfaces in a uniform formalization.

**C2: Uneven SPARQL 1.1 Support.** Closely related to C1, even endpoints that natively support the standard SPARQL protocol may vary greatly in terms of uneven SPARQL 1.1 support, in terms of language features (e.g. aggregates, supporting VALUES or SERVICE clauses),

[1] See https://www.anthropic.com/news/model-context-protocol

or limiting the result sizes, all of which require bespoke strategies for federation.

**C3: Metadata Heterogeneity.** Catalogues of source metadata to inform endpoint discovery, schema exploration and query planning are largely absent and neither fully standardized, and as a consequence efficient source selection remains difficult, often leading to slow query executions due to overestimation of relevant sources [Saleem et al., 2016].

**C4: Latency, Availability and Timeouts.** SPARQL endpoints are sometimes blocked or exhibit high latency, timeouts, and even up non-availability due to high concurrent server loads or complexity of queries, which requires federation engines to support non-blocking joins [Saleem et al., 2016]. In general, the variable availability of endpoints and non-predictability of timeouts largely affects query planning (and potentially also schema exploration).

Overall, C4 may affect the effectiveness of predominant "plan-and-execute" approaches, under circumstance where latency and up-to-date information changes dynamically during execution, or query (parts) are not executable in reasonable time.

**C5: Query Formulation.** Orthogonal to the above challenges, when starting from a (natural language) question, query formulation accuracy, is still not entirely solved. Firstly, this depends on the ability of (small) LLMs to produce *syntactically* correct queries and, secondly, on the semantic challenges in "knowing" the correct identifiers within queried KGs.

Thus, while impressive progress in regards of query formulation for specific SPARQL endpoints such as Wikidata [Liu *et al.*, 2024] have been made, the challenge of getting from natural language queries to results is aggravated in dynamic federated environments where neither the relevant endpoint URLs nor the schema is known upfront, as illustrated in Example 1:

**Example 1.** *As an illustrating example, prompted to gpt-oss-20b*[2]*, let us attempt to generate a federated SPARQL query:*

> *"Write a federated SPARQL query that retrieves all of Tim Berners-Lee's publications from DBLP, whose DOIs also appear on Wikidata."*

```
PREFIX dblp:   < http: //dblp.org/rdf/schema#>
PREFIX bibo:   <http://purl.org/ontology/bibo/>
PREFIX wdt:    <http://www.wikidata.org/prop/direct/>
SELECT DISTINCT ?dblpPub ?title ?doi ?wikidataItem
WHERE {
 SERVICE < https://dblp.org/sparql > {
   ?dblpPub a bibo:Article ;
           dblp: author  ?author ;
           dblp:title  ?title ;
           bibo: doi  ?doi .
           ?author dblp: name  "Tim Berners-Lee".}
SERVICE <https://query.wikidata.org/sparql> {
   ?wikidataItem wdt:P356 ?doi . } }
```

[2]See https://openai.com/index/introducing-gpt-oss/

As the example shows, both the endpoint address as well as the schema (incl. the prefix) were predicted erroneously. Furthermore, while on the surface level the query looks reasonable, a naive query planner, executing this query on a local engine capable of running `SERVICE` queries, could likely find a better plan than retrieving and locally joining all DOIs (320,960) from Wikidata and locally joining them with the (far fewer) DOIs retrievable for Tim Berners-Lee from DBLP, or resp. we need to rely on the query optimization capabilities of an overall local engine that executes the resulting query.

Our main hypothesis is that a combined agentic approach can help address these challenges, but this combination needs to go beyond current works: while existing approaches (see Section 8) have already introduced SPARQL querying capabilities into MCP, these have thus far been restricted to single endpoints and have not addressed dynamic endpoint discovery, schema exploration or federated querying. Also, KGQA benchmarks for SPARQL query formulation do so far not cover federation; consequently, we will need to combine existing KGQA and federation benchmarks. In our work, we aim to take a first crucial step by both (i) establishing such a combined benchmark and (ii) examining how to enable "agentic SPARQL" with an implementation, based on MCP, which we (iii) evaluate using different state-of-the-art LLMs.

The remainder of this work is structured as follows: Section 2 introduces preliminary concepts; Section 3 presents our SPARQL-MCP server; Section 4 describes the combined federated KGQA benchmark; Section 5 and Section 6 present the evaluation setup as well as results; Section 7 and Section 8 discuss limitations and related works; finally, Section 9 concludes the paper and gives an overview of future work.

## 2 Preliminaries

In the following, we assume familiarity with RDF, the SPARQL query language and protocol, and the MCP request/response protocol, and will only recapitulate briefly the parts relevant for our paper.

**RDF & SPARQL** provide standards to publish and query Web-accessible datasets consisting of (RDF) triples, that could be viewed as subject-predicate-object statement, or, likewise, as edges of a directed, labelled graph: let $G$ be the RDF graph of a dataset, i.e., a set of RDF triples $(s_d, p_d, o_d) \in (I \cup B) \times I \times (I \cup B \cup L)$ with $I$, $B$,$L$ denoting IRIs, blank nodes, or literals, resp. A *triple pattern* is an RDF triple $(s_q, p_q, o_q)$ where also variables from set $V$ are allowed in either position. A *basic graph pattern* (BGP) is a finite set of such triple patterns, and can be read as a conjunctive query. As illustrated in Example 1, BGPs can be written in Turtle syntax [Cyganiak *et al.*, 2014] in SPARQL, and be composed to more complex patterns, incl. a SERVICE $e$ $P$ operator that allows to delegate the execution of a SPARQL query pattern $P$ to an *endpoint* $e \in \mathcal{E}$ (the set of all endpoint URIs. A UNION$(P_1, \ldots, P_n)$ operator can be used express unions of conjunctive queries. For details of other operators such aggregating subqueries (e.g. COUNT, SUM, etc.), as well as FILTER expressions and LIMIT/OFFSET operators we refer to [Buil-Aranda *et al.*, 2013].

We herein treat "endpoint" as an HTTP/HTTPS interface

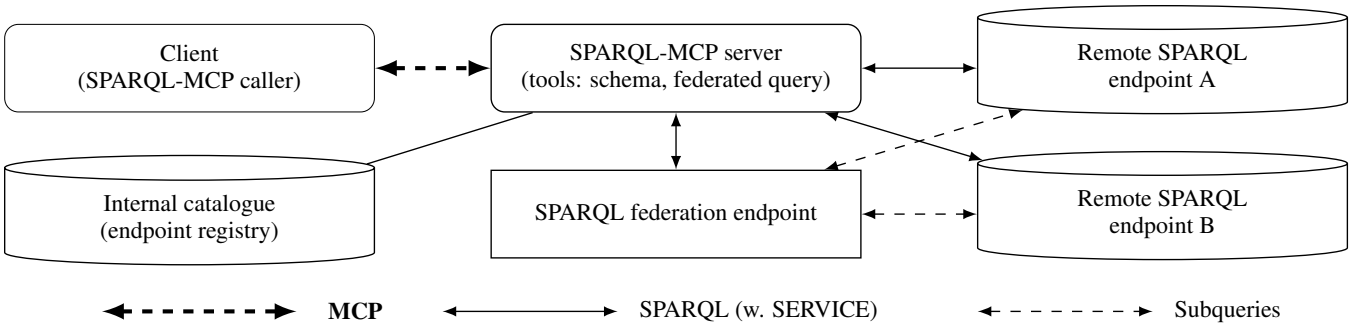


Figure 1: Architecture: the client communicates with the SPARQL-MCP server via the MCP protocol. The server includes the internal catalogue (endpoint registry) as a component, and issues SPARQL queries either directly to remote endpoints (A, B) or to a federation endpoint which decomposes and forwards subqueries to A and B for multi-SERVICE queries.

which accepts and answers SPARQL queries using the standard SPARQL protocol (and assume lower-level interfaces such as triple pattern fragments as simply endpoints only accepting restricted SPARQL queries, cf. [Azzam *et al.*, 2024]). Conveniently, the SPARQL protocol offers a JSON result format amenable to MCP: we assume all endpoints in $\mathcal{E}$ to support such JSON results.

For the sake of our paper and benchmark, we will assume that any natural language question $q_n$ we start from can be translated (minimally) to a single BGP query, and *query federation* involves splitting up this pattern to either (i) disjoint SERVICE sub-patterns delegated to different endpoints (*conjunctive split*), or (ii) creating UNIONs-concatenated copies of SERVICE-annotated patterns delegated to different endpoints (*disjunctive split*).

We note that under the assumption that a BGP can be answered by the union of all endpoints in $\mathcal{E}$, a trivial federation can be constructed by conjunctively splitting all triple patterns into disjunctive splits over all endpoints, i.e. for a BGP $P = \{t_1, \ldots, t_n\}$ the *trivial federation* is given by the pattern $\{\text{UNION}_{e\in\mathcal{E}}(\text{SERVICE } e\{t_1\}), \ldots, \text{UNION}_{e\in\mathcal{E}}(\text{SERVICE } e\{t_n\})\}$.

**Endpoint descriptions.** We note that SPARQL foresees in its standard a normative way to access a description (in the form of RDF again) from the endpoint. While the vocabulary for endpoint description is itself not part of the SPARQL standard, the accompanying VoID (vocabulary for interlinking datasets [Alexander *et al.*, 2011]) standard provides respective meta-data terms to describe the endpoint's RDF dataset in terms of a natural language description, its schema (all classes and properties), as well as base statistics (overall counts of triples, classes and properties). We herein assume that endpoints provide a VoID description and/or such descriptions are automatically created through the respective tool provided as part of SPARQL-MCP.

**MCP** allows MCP-compatible LLM client LLMs (client) to initiate retrieval via the JSON-RPC interactions with a (pre-configured) MCP server. First, the client performs *capability discovery* by calling the standardized `mcp_discover` method on the server. This returns a "manifest" of available resources (such as documents and database records, retrieved through `get_resource` or tool-specific equivalent passing the resource identifier, and streamed over HTTP or SSE), prompts (for interacting with language models; allowing the client to receive the fully instantiated prompt based on the predefined template), and tools (executed via the `run_tool` or eq., passing the tool identifier and JSON-encoded arguments, and receiving the pre-defined output schema). Underpinning all retrieval operations is a stateful session preserving context (authentication tokens, intermediate results).

## 3 SPARQL-MCP

In order to integrate SPARQL federation with agentic AI systems, we present SPARQL-MCP,[3] an MCP extension for SPARQL 1.1 federation engines that, together with schema exploration, enables AI agents to explore and query federated SPARQL endpoints. Unlike existing SPARQL-MCP implementations (see Section 7 below), SPARQL-MCP specifically addresses federated querying while tackling also uneven SPARQL feature support and metadata heterogeneity (by dynamic exploration/retrieval of VoID descriptions).

One notable limitation of SPARQL 1.1 includes the tendency of broadly adopted endpoints such as Wikidata and DBPedia to block SERVICE calls, with the major reason for such policy being that unconstrained SERVICE calls are prone to generating extensive result sets [Wu *et al.*, 2015] and thus unacceptable processing load on the endpoint. We solve this problem similarly to [Moos and Galgonek, 2025] by redirecting all multi-agent SERVICE queries to a proxy SPARQL federation engine, which, in turn, choreographs requests to the remote endpoints. For the purposes of modularity, concrete federation engines should be easily interchangeable, delimiting the query plan generation from the agentic endpoint discovery and exploration features. Figure 1 illustrates our architecture together with the protocols used for communication/data exchange between the components. We specify a particular lightweight federation engine used for evaluation in Section 5.

**Query Execution.** For federated query execution, the SPARQL-MCP server implements the `run_sparql_query` tool accepting a SPARQL query string, requested result for-

[3] See `https://github.com/semantisch/sparqlmcp` for the SPARQL-MCP documentation and code (MIT License)

mat and execution timeout. Queries must include SERVICE operators naming the target SPARQL endpoint(s). If the query contains more than one SERVICE, it is executed via a local SPARQL federation endpoint, which decomposes and forwards subqueries to respective remote endpoints. If the query contains exactly one SERVICE operator, the SERVICE wrapper will be removed and the query body sent directly to the endpoint (to address the blocking behaviour discussed above).

**VoiD Computation.** Computing VoID descriptions can be resource-intensive, especially for live endpoints. Therefore, the VoID retrieval tool (`get_void_descriptions`) follows a step-wise approach: (1) check cached versions of VoID descriptions in the internal catalogue, (2) if not available, retrieve the VoID descriptions (via "well-known" URLs, VoID triples in the default graph, VoID in named graphs, and service-description links) from the endpoint and cache them, (3) if VoID descriptions are not directly available, recreate parts of the VoID descriptions following the SPORTAL approach [Hasnain *et al.*, 2016] for computing VoID-style descriptions using SPARQL 1.1 self-descriptive queries, and then cache them.

# 4 FKGQA Benchmark

Several benchmarks exist for federated SPARQL query processing and KGQA over single knowledge graphs, but no general-purpose benchmark evaluates federated KGQA end-to-end. We extend Spider4SPARQL [Kosten *et al.*, 2023] (19 RDF datasets, 1,034 natural language questions, 542 SPARQL queries are publicly available) by defining a principled approach for dataset federation, cross-partitioning each dataset across multiple endpoints. Unlike existing benchmarks that assume explicit endpoint references, our benchmark requires agents to select relevant endpoints themselves, as is expected in real-world use cases.

## 4.1 Partitioning Method

Our partitioning goal is to maximize federation patterns (conjunctive and disjunctive splits) while achieving broad coverage, balanced shard sizes, and semantic coherence. Unlike approaches that minimize cross-partition queries [Azzam *et al.*, 2024], we construct minimal partition sets such that every BGP would require federation, using three strategies: vertical (predicate-based), class-based, and horizontal (subject-hash-based) partitioning.

**Class Sharding.** Under the assumption (which holds for all Spider4SPARQL queries when extended by axioms included in the benchmark) that all triple subjects are typed (i.e., each node appears in a $(s_d, \texttt{rdf:type}, C)$ triple in the dataset, we can assign each class IRI to a shard, and place all RDF triples in $G$ whose *subject* is an instance of that class into the corresponding shard. Formally, if a BGP mentions subject with two different types, e.g. explicitly $(s_1, \texttt{rdf:type}, c_1), (s_2\texttt{rdf:type}, c_2)$, or where resp. types $c_1$ or $c_2$ are inferrable by domain or range in the dataset (e.g. $s_1, p, o_1$ in the query with $p$, `rdfs:domain`, $c_1$ in the dataset), and where $c_1 \neq c_2$ and $c_1, c_2 \in I$, we split the dataset by placing all triples with subjects of these two classes in the data in different shards.

**Predicate (Vertical) Sharding** places triples in separate shards based on their predicates. Formally, for a BGP mentioning triple patterns $(s_1, p_1, o_1)$, $(s_2, p_2, o_2)$ where $p_1 \neq p_2$ with $p1, p2 \in I$, we split the dataset by placing all triples with these two predicates in different shards.

Both the above methods are intended to require *conjunctive splits* in the federation.

**Horizontal Sharding,** lastly, is intended to require a *disjunctive split*: here, again we exploit that all subjects are typed in the dataset, but we partition the dataset by instances (subjects) of a single class mentioned in the query: for each inferred class $c$, all subjects $s_d$ that are instances of $c$ are assigned to shards by a deterministic rule (e.g., hash of the IRI, or a skolemized blank node in place of the IRI). Every data triple $(s_d, p_d, o_d)$ is placed in the shard of $s_d$. This separates entities within a single class across shards, and is particularly suitable for queries with very small BGPs (e.g., a single triple pattern).

**Applicability Criteria.** Let $T(q)$ be the BGP of query $q$. Define the predicate set $\mathcal{P}(q) = \{p_q \mid (s_q, p_q, o_q) \in T(q), p_q \text{ is an IRI}\}$. Let $\mathcal{C}(q)$ be the set of class IRIs inferable from $T(q)$ via explicit `rdf:type` patterns or OWL `rdfs:domain`/`rdfs:range`. Let $\mathcal{S}(q) = \{s_q \mid (s_q, p_q, o_q) \in T(q), s_q \text{ is a variable}\}$ be the set of subject variables. Then predicate sharding is applicable iff $|\mathcal{P}(q)| \geq 2$, class sharding is applicable iff $|\mathcal{C}(q)| \geq 2$, and horizontal sharding is applicable iff $|\mathcal{S}(q)| \geq 1$ and $|\mathcal{C}(q)| \geq 1$. On the query level, across the benchmark, semantic/class sharding applies to 40.14% of queries, vertical/predicate sharding to 95.94%, and horizontal sharding applies to all of queries. Thus, the strategies are broadly applicable and can be combined in various ways to create diversely federated datasets.

**Rule Selection as Set Cover.** In our implementation, we define picking sharding candidates in a manner that distributes the dataset graph $G$ over a minimal set of shards, selecting applicable candidate shardings for each query, which we can formalize as a set-cover problem (NP-hard) [Karp, 2009] per dataset. As $Q$ be the set of queries, let $\mathcal{R}$ be the set of applicable shardings (predicate, class and horizontal splits). Each $r \in R$ then covers a number of queries in $Q$. The objective is to select a minimum-size subset $\mathcal{R}^* \subseteq \mathcal{R}$ such that $\bigcup_{r \in \mathcal{R}^*} C(r) = Q$. Formally we pick:

$$\min |\mathcal{R}^*| \quad \text{s.t.} \quad \forall q \in Q,\ \exists r \in \mathcal{R}^* : q \in C(r).$$

Since class shardings are naturally dominated by horizontal, we omit horizontal shardings as candidates for any query where a class sharding is applicable, ensuring higher variety in the federation patterns. We can then encode this set-cover formulation as an Answer Set Programming (ASP) problem [Brewka *et al.*, 2011] to compute such a minimal set of shardings per dataset.[4]

**Shard Materialisation.** Given a selected sharding set, we materialize shards by assigning each data triple $(s_d, p_d, o_d) \in G$ to the shard dictated by the sharding approaches (predicate-based for vertical shardings, class-of-$s_d$ for class shardings,

[4]See https://github.com/semantisch/fkgqa for the benchmark dataset, documentation and all code for generating the benchmark (MIT License)

and class-conditioned hash of $s_d$ for horizontal shardings), and place all remaining triples in a base shard.

### 4.2 Resulting Benchmark

We materialise federated shards for 19 datasets, yielding 118 shards in total, each of which are hosted in a separate endpoint. The minimal sharding-set size averages 3.7 (median 4), ranging from 2 to 8 rules per dataset, and the sharding composition becomes 18.30% vertical, 30.42% semantic, and 51.28% horizontal, which, we hypothesise, is an inverse of the ranking of partitioning approaches to *minimize* query costs in proposed federated systems (see Section 8). The number of shards per dataset ranges from 3 to 14 (median 6, mean 6.21). Shard balance (as the average coefficient of variation of shard sizes per dataset) is 0.88, with a moderate skew while keeping shards in comparable magnitudes of triples.

**Federation Coverage.** By construction, the selected shardings federate every query. We define the *fan-out* $f(Q)$ as the number of distinct shards/endpoints a BGP query $Q = \{t_1, \ldots, t_n\}$ may need to contact in order to ensure complete results across all endpoints in $\mathcal{E}$ as follows: let $\mathcal{E}_t$ be the set of all endpoints in $\mathcal{E}$ returning a non-empty result on triple pattern $t$, then we define $f(Q) = |\bigcup_{t \in Q} \mathcal{E}_t|$. Note that $f(Q)$ is an upper bound, i.e. based on the joins and underlying data, $Q$ may be correctly/completely answered using less endpoints as shown by Example 2.

**Example 2.** *Example BGP for the question "How many ships ended up being 'Captured'?":*

($?s$, `rdf:type`, `:ship`) *and*
($?s$, `:ship#disposition_of_ship`, `"Captured"`).

Under horizontal sharding on `:ship`, ship instances are split across multiple shards. However, in the materialized data all `"Captured"` ships hash to `shard #5`, so the query touches only that shard. In the evaluation (see Section 5), we additionally explore whether all potentially matching shards are actually consulted by the agent.

Based on shardings-implied fan-out, queries touch 6.48 shards on average (median 6, min 2, max 14). As shown by the example, in practice, realized fan-out can be lower: 4.84% have no matches and 24.49% queries match exactly one shard in realisation.

**VoiD Computation.** We pre-compute VoID descriptions for all 118 shards using the SPARQL-MCP `get_void_descriptions` tool, providing both standard and extended (with linksets) variants.

## 5 Evaluation

We implement the evaluation suite using the Agents SDK.[5] Tasks are executed by a ReAct-style agent that alternates *LLM calls* and *tool calls*. We structure the evaluation around an ablation design, looking into performance in the baseline case (endpoint URLs provided, federated querying tool available), basic high-level descriptions scenario and the full scenario with both federated querying and void retrieval tools available. For all evaluation setups, we limit turns to 10 per evaluation run, and log every step. We evaluate `GPT-5.2` (`gpt_5_2_2025_12_11`) as a high-capability frontier model and `Qwen3-8B` (`qwen_8b`) as a reproducible open-weight baseline representative of compact agent-capable models [Yang *et al.*, 2025]. As hyperparameters, we set `temperature` and thinking-related hyperparameters to the lowest values.

Runs yielding failed, unsound, or incomplete results are counted as evaluation errors and final query results are matched to the ground truth results based on signature sets. Prior to each evaluation, we issue a ramp up mix to warm caches and exclude cold-start effects. All experiments ran on Debian 13 server with 16 CPUs (Intel Xeon), 128 GB RAM; Python 3.13 and GCC 14.2.[6]

**Evaluation Setups.** We evaluate in three levels: (i) `baseline`: the agent only knows the endpoint URLs and has access to the `run_sparql_query` tool; (ii) `high_level`: the agent receives URLs plus one-sentence, topical natural-language endpoint descriptions,[7] and has access to the `run_sparql_query` tool; (iii) `void_tool`: `baseline`, plus the agent has access to the `get_void_descriptions` tool for schema exploration. In all conditions, agents must discover endpoints, synthesize federated SPARQL queries, execute via `run_sparql_query`, and refine based on results, submitting the final query via the evaluation environment-introduced `submit_final_query` call.

**Evaluation Metrics.** As results addressing various aspects of the agentic SPARQL performance, we report (i) a descriptive analysis of execution traces, (ii) syntactic validity of generated SPARQL queries, (iii) overall pipeline accuracy, (iv) an evaluation of endpoint accuracy (i.e., assessing which endpoints are consulted by the agent), and, finally, (iv) assess distinctive behavioural patterns.

## 6 Results

**Descriptive Analysis.** In total, the evaluation has included 5,814 separate agent runs. In terms of cost, a total of 82,574,734 tokens has been used. Of these, GPT-5.2 contributed 61,429,831 tokens (median 20,480 tokens per run, min. 2,457, max. 172,614), while Qwen3-8B contributed 21,144,903 tokens in total (median 8,288 tokens per run, min. 1,550, max. 335,299), illustrating that the higher-capacity model dominates overall token usage. In total, end-to-end runtime of all agent runs has been above 65 hours, with a median runtime of 22.2 seconds per run (max. 11 minutes). Breaking this down by model, GPT-5.2 achieved a median end-to-end runtime of 16.3 s. per run, whereas Qwen3-8B required almost twice as much, 31.9 s. In terms of call patterns, GPT-5.2 averaged 5.5 LLM calls and 9.0 SPARQL calls per run, whereas Qwen3-8B averaged 7.2 LLM calls and 6.4

[5] See https://github.com/openai/openai-agents-python

[6] See the full evaluation results, evaluation scripts and the respective documentation (MIT License): https://github.com/semantisch/sparql-mcp-evaluation

[7] E.g., "A catalog of automobile models associated with their manufacturers." and "A reference-style catalog of automobile makes and models with associated geographic origin information." All high-level descriptions are included in the benchmark repository.

SPARQL calls per run; VoID retrieval calls averaged 1.0–1.1 per run in the respective setup, with SPARQL call counts highest in the baseline (11.1 for GPT-5.2, 6.4 for Qwen3-8B) and reduced to 6.2 for GPT-5.2 when high-level endpoint descriptions were provided, while Qwen3-8B showed similar counts (6.8) in that mode.

**Syntactic Analysis.** Syntactic analysis shows that 29,431 out of 38,886 SPARQL queries (75.7% overall, across all LLMs and setups) executed successfully, i.e., were syntactically correct. GPT-5.2 achieved high success rates (97.4%–98.0% across setups), while Qwen3-8B showed lower rates (41.5%–61.1%), with the highest rate in the `high_level` setup. For Qwen3-8B, the primary pitfalls were: 41.6% failed parsing, 24.3% had prefix issues, 1.3% had unmatched braces, and 1.5% had malformed SERVICE syntax.

**Final Results Analysis.** GPT-5.2 achieved accuracy rates of 42.1% in `baseline`, 45.4% in `high_level`, and 43.5% in `void_tool`. By contrast, Qwen3-8B achieved accuracy rates of 13.1% in `baseline`, 13.2% in `high_level`, and 13.8% in `void_tool`. For the larger LLM, we could, therefore, in the federated setup, achieve the exact same success rates as the state-of-the-art (including LLM-driven) approaches evaluated on the original benchmark (45%) [Kosten *et al.*, 2023].

**Endpoint Accuracy.** GPT-5.2 achieved high consultation rates in `baseline` (90.7% for successful) and `void_tool` (91.7% for successful), but lower rates in `high_level` (25.8% for successful), suggesting that high-level descriptions enable effective source pre-selection. Qwen3-8B showed consistently high rates in `void_tool` (98.6% for successful). A large fraction of queries were *trivial federations* (consulting all endpoints when fewer would suffice): GPT-5.2 produced trivial queries in 90.2%–91.7% of successful runs in baseline modes but only 11.0% in `high_level`, while Qwen3-8B showed high trivial query rates across all modes (68.5%–98.6%), indicating a tendency to query all endpoints rather than performing selective discovery.

**Behavioural Analysis.** Behavioral analysis reveals distinct agent strategies: GPT-5.2 exhibits exploration behavior with frequent endpoint switching and minimal query redundancy, gradually discovering endpoints. By contrast, Qwen3-8B shows almost no exploration, rarely switches endpoints, but exhibits high query redundancy, discovering endpoints quickly by querying all endpoints simultaneously. GPT-5.2 achieves faster time-to-first-result compared to Qwen3-8B, while both models show similar correction dynamics.

## 7 Discussion

Our evaluation demonstrates that agentic federated SPARQL querying is viable for sufficiently capable models: GPT-5.2 achieved accuracy rates (42.1%–45.4%) comparable to state-of-the-art approaches on the original Spider4SPARQL benchmark (45%) [Kosten *et al.*, 2023], despite the added complexity of federation. High-level endpoint descriptions proved more effective than detailed VoID metadata for guiding source selection, reducing trivial queries from 90.2% to 11.0% for GPT-5.2 while improving accuracy. However, smaller models (Qwen3-8B) showed significantly lower performance (13.1%–13.8% accuracy) and high syntactic error rates (41.5%–61.1%), indicating that model scale and SPARQL-specific capabilities are still critical. The behavioral differences such as GPT-5.2's exploration strategy versus Qwen3-8B's approach of querying all endpoints simultaneously suggest that different models require tailored tool designs and prompting guidance. While LLMs can leverage semantic understanding for endpoint selection, they often lack the cost-awareness of traditional federated query optimizers, leading to inefficient query plans that consult unnecessary endpoints (trivial plans).

## 8 Related Work

**Federated Query Processing** enables data integration across distributed endpoints. Since federated execution must account for heterogeneous source capabilities and incomplete or imprecise metadata (and, to a lesser extent, network effects), substantial work has focused on query optimisation. FedX [Schwarte *et al.*, 2011b] introduced query decomposition and source selection techniques, while CoDA [Ibragimov *et al.*, 2015] proposed cost-based strategies for aggregates and alternative join plans. Heuristic reordering approaches [Yannakis *et al.*, 2018] further improve performance without requiring complete statistics, and Heling and Acosta [Heling and Acosta, 2022a] extended optimisation to heterogeneous Linked Data Fragment (LDF) interfaces via interface-aware physical operators.

Endpoint discovery and schema exploration are central to federated querying, with prior work covering metadata-driven, ad-hoc query-based, and hybrid discovery strategies [Montoya *et al.*, 2017; Heling, 2019], as well as VoID-based and probing-based schema inference [Zeimetz and Schenkel, 2019; Heling and Acosta, 2022a]. Evaluation of federated SPARQL querying rely on benchmarks including FedBench [Schmidt *et al.*, 2011b], QFed [Rakhmawati *et al.*, 2014], and LargeRDFBench [Saleem *et al.*, 2018]. Nonetheless, most federated SPARQL systems require manual query formulation, limiting accessibility. Recent efforts have aimed to bridge the gap between natural language interfaces and federated SPARQL querying, with Emonet et al. [Emonet *et al.*, 2024] introducing a RAG framework that translates natural language questions into federated SPARQL queries over bioinformatics knowledge graphs. Widely used single-source KGQA benchmarks include QALD [Unger *et al.*, 2012], LC-QuAD [Trivedi *et al.*, 2017], and Spider4SPARQL [Kosten *et al.*, 2023].

**KG Schema Exploration** Prior work in KGQA highlights structure- and schema-aware exploration, including agentic approaches [Gu *et al.*, 2023], iterative reasoning methods [Sun *et al.*, 2023; Luo *et al.*, 2023; Chen *et al.*, 2024], ranking and pruning strategies [Tian *et al.*, 2024; Tan *et al.*, 2025], graph-based importance measures [Jimenez Gutierrez *et al.*, 2024; Gutiérrez *et al.*, 2025], AMR-guided mappings [Emonet *et al.*, 2024], and learned retrievers [Li *et al.*, 2024]. Complementary work pre-computes indexes for schema access [Walter and Bast, 2025]. While effective for single-KG settings, these methods assume a fixed, known

graph. Our work focuses on endpoint- and schema-level exploration in federated settings, exposing metadata (e.g., VoID descriptions) operationally via MCP.

**MCP and Federated Retrieval.** MCP wrappers have been proposed for SPARQL. The Wikidata SPARQL MCP Server[8] targets the Wikidata endpoint and exposes a single `sparql_query` tool. Triplestore vendors also provide MCP servers, including the Apache Jena MCP Server[9], compatible with Jena Fuseki[10], as well as corresponding solutions for GraphDB[11] and Stardog[12]; these primarily support direct query/update operations and basic graph inspection. The RDF Explorer MCP server[13] additionally offers endpoint-specific statistics and search utilities. To the best of our knowledge, none of these implementations support schema exploration or federated SPARQL querying, nor has prior work systematically evaluated ReAct-style MCP-based SPARQL agents.

## 9 Conclusions and Future Work

This work introduces agentic federated SPARQL querying, enabling LLM-based agents to autonomously discover endpoints, explore schemas, and formulate federated queries from natural language questions. Our contributions include: (i) a combined FKGQA benchmark for evaluating agentic federated Knowledge Graph Question Answering; (ii) the SPARQL-MCP server implementation with tools for federated querying, endpoint discovery, and schema exploration; and (iii) a comprehensive evaluation across 5,814 agent runs comparing different architectural options. Our evaluation reveals that agentic federated SPARQL querying is viable for sufficiently capable models: GPT-5.2 achieved accuracy rates (42.1%–45.4%) comparable to state-of-the-art approaches on the original Spider4SPARQL benchmark (45%) [Kosten *et al.*, 2023], despite the added complexity of federation. High-level endpoint descriptions proved more effective than detailed VoID metadata, reducing trivial queries from 90.2% to 11.0% while improving accuracy. However, smaller models showed significantly lower performance, indicating that model scale and SPARQL-specific capabilities are critical for production use. Below, we outline important directions for future work:

**FW1: SPARQL Capabilities:** Addressing **C5** discussed in Section 1, the significantly lower syntactic validity rates for Qwen3-8B (41.5%–61.1% vs. 97.4%–98.0% for GPT-5.2) indicate that smaller models require substantial improvements in SPARQL-specific fine-tuning or more structured query generation pipelines to be viable.

**FW2: Schema Exploration:** Even beyond inherent metadata heterogeneity (**C3**), enabling the VoID tool did not consistently improve accuracy over high-level descriptions, consultation rates in `void_tool` (91.7% for GPT-5.2, 98.6% for Qwen3-8B) or trivial-query rates. This finding contrasts with VoID-based schema inference work [Zeimetz and Schenkel, 2019; Heling and Acosta, 2022a], which assumed more sophisticated planning. In agentic settings, simpler high-level descriptions may be more effective, which prompts future work to test how to structure metadata for agents.

**FW3: SPARQL Registry:** As thematised in **C4** and **C2**, real-world endpoints face unavailability, latency and timeouts, have heterogeneous interfaces, and missing metadata while catalogues (e.g., SPARQLES [Vandenbussche *et al.*, 2017]) are largely inaccessible. Future work should extend catalogue metadata (partitions, availability/quality) and provide searchable, up-to-date endpoint and ontology repositories.

**Acknowledgements**

This research was funded in whole or in part by the Austrian Science Fund (FWF) Cluster of Excellence "Bilateral AI" (BILAI) [10.55776/COE12].

## References

[Acosta *et al.*, 2011] M. Acosta, M.E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *The Semantic Web – 10th International Semantic Web Conference, Bonn, Germany*, volume 7031 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011.

[Alexander *et al.*, 2011] K. Alexander, R. Cyganiak, M. Hausenblas, J. Zhao. Describing Linked Datasets with the VoID Vocabulary. W3C Interest Group Note, February 2011. https://www.w3.org/TR/void/.

[Anthropic, 2025] Anthropic. Model context protocol specification, version 2025-06-18. MCP Specification, June 2025. Accessed: 2025-09-14.

[Azzam *et al.*, 2024] A. Azzam, A. Polleres, J.D. Fernandez, M. Acosta. smart-KG: Partition-based linked data fragments for querying knowledge graphs. *Semantic Web*, 15(5):1791–1835, 2024.

[Banerjee *et al.*, 2022] D. Banerjee, P. Ajit Nair, J. Neet Kaur, R. Usbeck, C. Biemann. Modern baselines for sparql semantic parsing. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2260–2265, New York, NY, USA, 2022. Association for Computing Machinery.

[Böhm *et al.*, 2011] C. Böhm, J. Lorey, F. Naumann. Creating void descriptions for web-scale data. *Journal of Web Semantics*, 9(3):339–345, 2011.

[Brewka *et al.*, 2011] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, December 2011.

[8] See https://github.com/QuentinCody/wikidata-sparql-mcp-server

[9] See https://github.com/ramuzes/mcp-jena

[10] See https://jena.apache.org/documentation/fuseki2/fuseki-docker.html

[11] See https://graphdb.ontotext.com/documentation/11.1/using-graphdb-llm-tools-with-external-clients.html

[12] See https://lobehub.com/mcp/vairpower-stardog_graphrag_mcp_poc

[13] See https://github.com/emekaokoye/mcp-rdf-explorer

[Buil-Aranda *et al.*, 2013] C. Buil-Aranda, M. Arenas, O. Corcho, A. Polleres. Federating queries in SPARQL1.1: Syntax, semantics and evaluation. 18(1), 2013.

[Chen *et al.*, 2024] L. Chen, P. Tong, Zhongming Jin, Y. Sun, J. Ye, H. Xiong. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. *Advances in Neural Information Processing Systems*, 37:37665–37691, 2024.

[Cyganiak *et al.*, 2014] R. Cyganiak, D. Wood, M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, February 2014. https://www.w3.org/TR/rdf11-concepts/.

[D'Abramo *et al.*, 2025] J. D'Abramo, A. Zugarini, P. Torroni. Investigating large language models for text-to-SPARQL generation. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pages 66–80, Albuquerque, New Mexico, USA, May 2025. Association for Computational Linguistics.

[Emonet *et al.*, 2024] R. Emonet, G. Heidsieck, V. Emonet. Llm-based sparql query generation from natural language over federated knowledge graphs. arXiv:2410.06062, 2024.

[Gu *et al.*, 2023] Y. Gu, X. Deng, Y. Su. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4928–4949, Toronto, Canada, July 2023. Association for Computational Linguistics.

[Gutiérrez *et al.*, 2025] B. Jiménez Gutiérrez, Y. Shu, W. Qi, S. Zhou, Y. Su. From RAG to memory: Non-parametric continual learning for large language models. *arXiv:2502.14802*, 2025.

[Harris and Seaborne, 2013] S. Harris, A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, March 2013.

[Hasnain *et al.*, 2016] A. Hasnain, Q. Mehmood, S. Sana e Zainab, A. Hogan. Sportal: profiling the content of public sparql endpoints. *International Journal on Semantic Web and Information Systems*, 12(3):134–163, 2016.

[Heling and Acosta, 2022a] L. Heling, M. Acosta. Federated SPARQL query processing over heterogeneous linked data fragments. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 1047–1057, New York, NY, USA, 2022. Association for Computing Machinery.

[Heling, 2019] L. Heling. Quality-driven query processing over federated rdf data sources. In *European Semantic Web Conference*, pages 209–219. Springer, 2019.

[Ibragimov *et al.*, 2015] D. Ibragimov, K. Hose, T. Bach Pedersen, E. Zimányi. Processing aggregate queries in a federation of SPARQL endpoints. In *The Semantic Web. Latest Advances and New Domains*, pages 269–285, Cham, 2015. Springer International Publishing.

[Jimenez Gutierrez *et al.*, 2024] B. Jimenez Gutierrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569, 2024.

[Karp, 2009] R. M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2009.

[Karpukhin *et al.*, 2020] V. Karpukhin, B. Oguz, S. Min, P. S. H. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020.

[Kosten *et al.*, 2023] C. Kosten, P. Cudré-Mauroux, K. Stockinger. Spider4sparql: a complex benchmark for evaluating knowledge graph question answering systems. In *2023 IEEE international conference on big data (BigData)*, pages 5272–5281. IEEE, 2023.

[Lewis *et al.*, 2020] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[Li *et al.*, 2024] M. Li, S. Miao, P. Li. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. *arXiv:2410.20724*, 2024.

[Liu *et al.*, 2024] S. Liu, S.J. Semnani, H. Triedman, J. Xu, I. Dan Zhao, M.S. Lam. Spinach: Sparql-based information navigation for challenging real-world questions, 2024.

[Liu *et al.*, 2025] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Zhang, J. Fan, G. Li, N. Tang, Y. Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[Luo *et al.*, 2023] L. Luo, Y.-F. Li, G. Haffari, S. Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv:2310.01061*, 2023.

[Matarazzo and Torlone, 2025] A. Matarazzo, R. Torlone. A survey on large language models with some insights on their capabilities and limitations. *arXiv:2501.04040*, 2025.

[Montoya *et al.*, 2017] G. Montoya, H. Skaf-Molli, K. Hose. The odyssey approach for optimizing federated sparql queries. In *International semantic web conference*, pages 471–489. Springer, 2017.

[Moos and Galgonek, 2025] P. Moos, J. Galgonek. SPARQL federated query debugging tool. *Journal of Information Science*, 2025. Forthcoming.

[Rakhmawati *et al.*, 2014] N.A. Rakhmawati, M. Saleem, S. Lalithsena, S. Decker. Qfed: Query set for federated SPARQL query benchmark. In *Proceedings of the 16th International Conference on Information Integration*

*and Web-Based Applications & Services*, pages 207–211, 2014.

[Saleem et al., 2016] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, A.-C. Ngonga Ngomo. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web*, 7(5):493–518, 2016.

[Saleem *et al.*, 2018] M. Saleem, A. Hasnain, A.C. Ngonga Ngomo. Largerdfbench: A billion triples benchmark for sparql endpoint federation. *Journal of Web Semantics*, 48:85–125, 2018.

[Schmidt *et al.*, 2011b] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference*, pages 585–600. Springer, 2011.

[Schwarte *et al.*, 2011b] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt. Fedx: optimization techniques for federated query processing on linked data. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, page 601–616, Berlin, Heidelberg, 2011. Springer-Verlag.

[Sun *et al.*, 2023] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L.M. Ni, H.-Y. Shum, J. Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv:2307.07697*, 2023.

[Tan *et al.*, 2025] X. Tan, X. Wang, Q. Liu, X. Xu, X. uan, W. Zhang. Paths-over-graph: Knowledge graph empowered large language model reasoning. In *Proceedings of the ACM on Web Conference 2025*, pages 3505–3522, 2025.

[Tian *et al.*, 2024] Y. Tian, D. Song, Z. Wu, C. Zhou, H. Wang, J. Yang, J. Xu, R. Cao, H. Wang. Augmenting reasoning capabilities of llms with graph structures in knowledge base question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 11967–11977, 2024.

[Trivedi *et al.*, 2017] P. Trivedi, G. Maheshwari, M. Dubey, J. Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International semantic web conference*, pages 210–218. Springer, 2017.

[Unger *et al.*, 2012] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, P. Cimiano. Template-based question answering over RDF data. In *Proceedings of the 21st international conference on World Wide Web*, pages 639–648, 2012.

[Vandenbussche *et al.*, 2017] P.-Y.. Vandenbussche, J. Umbrich, L. Matteis, A. Hogan, C. Buil-Aranda. Sparqles: Monitoring public SPARQL endpoints. *Semantic web*, 8(6):1049–1065, 2017.

[Walter and Bast, 2025] S. Walter, H. Bast. Grasp: Generic reasoning and sparql generation across knowledge graphs, 2025.

[Wilkinson *et al.*, 2016] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L.B. da Silva Santos, P. E Bourne, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

[Williams, 2013] G.T. Williams. SPARQL 1.1 Service Description. W3C Recommendation, March 2013. Editor: Gregory Todd Williams.

[Wu *et al.*, 2015] H. Wu, A. Yamaguchi, J.-D. Kim. Dynamic join order optimization for sparql endpoint federation. In *SSWS@ ISWC*, pages 48–62, 2015.

[Yang *et al.*, 2025] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv:2505.09388*, 2025.

[Yannakis *et al.*, 2018] T. Yannakis, P. Fafalios, Y. Tzitzikas. Heuristics-based query reordering for federated queries in SPARQL1.1 and SPARQL-LD. In *GeoLD-QuWeDa@ESWC*, 2018.

[Yao *et al.*, 2023] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[Zeimetz and Schenkel, 2019] T. Zeimetz, R. Schenkel. Analyzing online schema extraction approaches for linked data knowledge bases. In *Proceedings of the International Workshop on Semantic Big Data*, pages 1–6, 2019.