Does SPARQL federation work in the real world? A case study over large biological SPARQL endpoints

Elias Crum[1], Bryan-Elliott Tam[1], Jonni Hanski[1], Ana-Claudia Sima[2], Tarcisio Mendes de Farias[2], Jerven Bolleman[3], and Ruben Taelman[1]

[1] IDLab, Ghent University – imec, Belgium
[2] SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland
[3] Swiss-Prot group, SIB Swiss Institute of Bioinformatics, Geneva, Switzerland

Abstract. Federated SPARQL querying theoretically enables data integration over distributed knowledge sources without requiring centralized data replication. In practice, federated SPARQL querying can be performed manually (i.e., the user provides `SERVICE` clauses that specify targets for operations) or algorithmically (i.e., an adaptive approach for automatic source assignment for operations). To date, most evaluations of SPARQL federation, both manual and algorithmic, are conducted under controlled benchmark conditions and say relatively little about how federation behaves against public endpoints in everyday use. We present a longitudinal study that executes 67 real-world federated SPARQL queries that target over 20 widely-used, large public SPARQL endpoints, using both manual and algorithmic federation approaches, at 4 different time-points between Spring 2025 and Spring 2026. The federated queries were obtained from users of these endpoints and the majority represent complex, biologically relevant questions. There were two crucial findings. First, we found that the current state-of-the-art algorithmic federation approaches perform substantially worse than manual approaches across all time points, and in most cases, encounter errors when executing the queries tested. Second, we found that query execution success decreased over the time points tested, for both manual and automatic methods. We frame the discussion of our findings from two perspectives: the user executing the federated queries, and the endpoint maintainer who must balance resource availability and usage demands. Ultimately, we aim to bring attention to the current state of SPARQL federation in complex, real-world scenarios – specifically that this federation does not consistently work – to encourage collaborative, community-driven improvements for users and data source maintainers alike.

Keywords: SPARQL Querying · Federated Querying · Real-World Querying · SPARQL Endpoints

1 Introduction

RDF and Knowledge Graph technologies enable data from independently maintained sources to be interlinked through shared, universal identifiers and common

vocabularies, supporting a web of distributed, interoperable data [20]. Federated SPARQL querying enables the virtual integration of interlinked data in a query-driven manner and empowers users to answer questions over multiple data sources without requiring central replication. This is particularly useful in life sciences, where specialized knowledge is distributed across many data sources [18], often maintained by different organizations [21], subject to different licensing constraints [26], or too large to be centralized [22], and where cross-source questions are common [27].

SPARQL endpoints expose RDF graphs through remote query interfaces and have become important resources for integrating biological and chemical knowledge, with prominent examples including UniProt [41], ChEMBL [44], PubChem [13], IDSM [14], and SIB RDF resources [39]. Although benchmarks often treat endpoints as stable, protocol-compliant services, public endpoints are live operational systems shaped by changing data, software, and shared production constraints. This distinction is especially important for biological data federation, where widely used endpoints support active user communities [41] and enable queries across resources, such as federated queries over multiple resources that lead to novel biomedical discoveries [27]. Consequently, availability, latency, interoperability, timeouts, and usage limits can vary substantially in practice [42,23]; such operational constraints are therefore not exceptional failures, but recurring factors that shape federated query execution in these real-world environments.

In this study, we assess the operational effectiveness of federated querying by examining how domain-relevant, real-world queries perform over public SPARQL endpoints from the user's perspective. Beyond simply measuring query success and failure, we capture temporal variation in execution behavior through a longitudinal study spanning 18 months, during which the same set of queries was executed across over 20 public SPARQL endpoints. Rather than treating endpoint behavior as fixed background noise, we explicitly consider it a central factor influencing federated query performance.

Our work specifically assesses three main questions: (**RQ1**) Do meaningful real-world federated biological queries execute successfully over public SPARQL endpoints? (**RQ2**) To what extent does success vary by federation approach? (i.e., manual server-side federation vs. automatic client-side algorithms) (**RQ3**) How do real-world federated query executions evolve over time in a public deployment setting? By answering these questions, we aim to complement prior evaluations under controlled conditions [35,32,10] with evidence from dynamic real-world endpoint behavior, and catalog client and server-side real-world operational constraints.

2 Background

The SPARQL query language, SPARQL federation, federated source selection, and SPARQL query complexity underpin our evaluation of SPARQL federation

in practice. This section provides a brief overview of these concepts to support the remainder of the study.

Federated SPARQL Query with and without SERVICE clauses

```
# Datasources: https://beta.sparql.swisslipids.org/sparql
PREFIX SWISSLIPID: <https://swisslipids.org/rdf/SLM_>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rh: <http://rdf.rhea-db.org/>
PREFIX up: <http://purl.uniprot.org/core/>
# Example 7
SELECT ?startId ?startLabel ?id ?name
WHERE {
  # Queried lipid species
  VALUES ?startId { SWISSLIPID:000056871 }
  ?startId rdfs:label ?startLabel .
  # Corresponding Isomeric subspecies
  ?id rdfs:subClassOf+ ?startId .
  ?id SWISSLIPID:rank SWISSLIPID:Isomeric_Subspecie ;
      rdfs:label ?name .
  # Mapped ChEBI
  ?id owl:equivalentClass ?chebi .
  # federated query to Rhea
  SERVICE <https://sparql.rhea-db.org/sparql> {
    ?rhea rh:equation ?rheaEquation .
    ?rhea rh:side/rh:contains/rh:compound/rh:chebi ?chebi .
  }
  # federated query to UniProt
  SERVICE <https://sparql.uniprot.org/sparql/> {
    ?uniprot up:reviewed true .
    ?uniprot up:recommendedName/up:fullName ?uniprotName .
    ?uniprot up:annotation/up:catalyticActivity/up:catalyzedReaction ?rhea .
  }
}
```

```
# Datasources: https://beta.sparql.swisslipids.org/sparql
# https://sparql.rhea-db.org/sparql
# https://sparql.uniprot.org/sparql/
PREFIX SWISSLIPID: <https://swisslipids.org/rdf/SLM_>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rh: <http://rdf.rhea-db.org/>
PREFIX up: <http://purl.uniprot.org/core/>
# Example 7
SELECT ?startId ?startLabel ?id ?name
WHERE {
  # Queried lipid species
  VALUES ?startId { SWISSLIPID:000056871 }
  ?startId rdfs:label ?startLabel .
  # Corresponding Isomeric subspecies
  ?id rdfs:subClassOf+ ?startId .
  ?id SWISSLIPID:rank SWISSLIPID:Isomeric_Subspecie ;
      rdfs:label ?name .
  # Mapped ChEBI
  ?id owl:equivalentClass ?chebi .
  # federated query to Rhea
  {
    ?rhea rh:equation ?rheaEquation .
    ?rhea rh:side/rh:contains/rh:compound/rh:chebi ?chebi .
  }
  # federated query to UniProt
  {
    ?uniprot up:reviewed true .
    ?uniprot up:recommendedName/up:fullName ?uniprotName .
    ?uniprot up:annotation/up:catalyticActivity/up:catalyzedReaction ?rhea .
  }
}
```

Fig. 1. Comparison of a federated SPARQL query manual server-side federation with `SERVICE` clauses (left) versus the equivalent query using client-side automated federation strategies without `SERVICE` clauses (right). Black box shows SPARQL endpoint URLs, Blue, Purple, and Red boxes designate sub-sections of the query sent to different endpoints.

SPARQL is the W3C-standard query language for retrieving and manipulating data stored in Resource Description Framework (RDF) format [12]. **SPARQL federation** extends this model by enabling a single query to integrate the data of multiple, distributed data sources. Figure 1 displays two forms of a representative federated query (query 7 in this study).

A crucial step in federated SPARQL query processing is *source selection*: deciding which parts of a query are evaluated by which data sources. This can be specified *manually* with the SPARQL 1.1 `SERVICE` keyword, as in the left panel of Figure 1, or handled *automatically* by a client-side federation engine given only the query and target sources, as in the right panel. In both cases, *query planning*, including join ordering, reducing intermediate results, and deciding whether joins are executed locally or at remote endpoints, also has a significant impact on overall query performance [36]. Automatic federation engines use different strategies for source selection, query partitioning, and optimization, and are further discussed in Section 3.1.

The SPARQL query itself is a key factor in federated query performance. Even over a single endpoint, evaluation becomes more challenging as query structure grows in complexity and size. In particular, the evaluation of basic

graph patterns is known to be NP-complete, and the addition of operators such as `OPTIONAL` and `UNION` further increases evaluation complexity [29]. Queries with many triple patterns can also produce large intermediate results and costly join operations, making planning and execution challenging [36]. In federated settings, these challenges are amplified: query engines must not only optimize joins and intermediate results locally, but also coordinate query decomposition, data transfer, and join execution across multiple remote endpoints with limited statistics and higher communication costs. Consequently, even moderately complex SPARQL algebra can become substantially harder to execute efficiently across multiple endpoints.

3 Related Work

Prior work on federated SPARQL querying has advanced algorithms for source selection, query planning, and distributed join processing, but has typically been evaluated under controlled benchmark conditions. While traditional, controlled benchmark settings yielded important insights into algorithmic efficiency, they provide a limited view of how federation behaves against public endpoints under dynamic real-world conditions. This study therefore, shifts the focus from stable benchmark environments to operational behavior in public settings.

3.1 Federated Querying Algorithms

SPARQL federation has been studied primarily as a problem of *source selection* and efficient intermediate-result processing. Systems such as FedX and SPLENDID illustrate two influential execution styles: FedX performs source selection dynamically at query time, avoiding reliance on precomputed source catalogs [37], whereas SPLENDID uses standardized dataset metadata, in particular VoID descriptions [9], to guide source selection and planning [16]. Other approaches target volatile environments, such as ANAPSID [1], or rely on custom statistics and metadata, including HiBISCuS [33], CostFed [34], and FedUP [2]. From an endpoint maintainer's perspective, however, publishing and maintaining such metadata is optional and often costly; consequently, endpoint-exposed metadata are often incomplete or unavailable in practice [42]. Federation is therefore commonly framed as query optimization under incomplete statistics and limited coordination between independently operated endpoints [25]. Given the real-world focus of our study, we evaluate only algorithms that do not depend on non-standard metadata for automated federation. Other relevant approaches, such as ANAPSID, were not included because no public implementations were available.

Recent systems have also made practical experimentation with federation strategies easier. In particular, Comunica provides a modular query engine in which source-selection methods, query planners, and join algorithms can be combined and compared [40]. This modularity enables hybrid strategies, such as

replacing `ASK`-based source selection, as used in FedX, with `COUNT`-based alternatives. Using a shared execution framework also reduces performance differences caused by implementation details, such as programming language, libraries, or threading models, allowing comparisons to focus more directly on the federation strategy itself.

3.2 Federated Querying Benchmarks & Evaluations

Benchmarks and empirical evaluations have played an important role in federated SPARQL research. FedBench provides curated datasets and queries for comparing federation engines [35,31], while LargeRDFBench extends this approach to larger data volumes and more demanding federation scenarios [32]. Other work has questioned how well curated or synthetic workloads capture the structure and difficulty of real query behavior, for example, in Wikidata-oriented benchmarking [19].

Although valuable for controlled comparison, these benchmarks typically assume stable execution environments and abstract away from operational characteristics of public SPARQL services, such as endpoint timeouts, rate limits, transient HTTP failures, changing metadata, and fluctuating responsiveness. In real-world conditions, public endpoints can vary substantially in availability, interoperability, and performance [42,23], and these factors can directly affect federated query execution, sometimes causing partial or complete failure. Work has been done to mitigate these challenges, an example is SPARQL continuation queries [30], but such solutions have not yet been adopted at scale.

Rather than proposing a new federation algorithm or benchmark suite, here we study how federated biological queries behave longitudinally over widely used public endpoints using three different federation strategies (one server-side and two client-side). Our work therefore, complements prior algorithmic and benchmark-focused research by assessing the impact of real-world operational fragility, failure modes, and the gap between controlled benchmark assumptions and real-world use.

4 Methods

4.1 General Study Design

This study is an 18-month longitudinal evaluation of SPARQL query federation over public endpoints, focusing on the practical executability of real federated queries rather than controlled benchmark performance. Using the curated real-world query records described in Section 4.2, we analyzed individual query executions within federation approach configuration–date experiments, enabling comparison across both federation approaches and changes in endpoint behavior over time. Because the evaluated strategies were implemented in Comunica rather than by directly invoking native FedX or SPLENDID binaries, we refer to them as *metadata-free* (e.g., FedX) and *metadata-guided* (e.g., SPLENDID)

federation approaches throughout the study. References to the companion results webpage are made throughout the paper as embedded links and direct to the same base webpage at the URL: `https://ecrum19.github.io/fed-survey-results/`.

4.2 Real-World Queries

Queries. The query corpus was obtained from users of SIB-hosted biological SPARQL endpoints [5]. It contains 67 federated queries, all created by domain experts to answer real-world questions or test realistic scenarios. These queries were characterized in a structured CSV containing the original example query URL, natural-language description, target endpoint, federated endpoints, relevance/background notes, a "toy" flag, execution notes, similarity annotations, known problems, and aliases [8]. The experimental repository stores the same corpus, with reduced metadata, in JSON format [7]. More than half of the queries have a confirmed real-world application or address an open question, and around one quarter have produced results published in peer-reviewed papers.

To test client-side *automatic* source selection, a preprocessing script [7] generates two versions of each query: a SERVICE form, corresponding to the original query with one target source and explicit `SERVICE` clauses, and a NO-SERVICE form, in which `SERVICE` clauses are removed, and multiple source targets are provided. Figure 1 shows an example of both forms. We refer to experiments using the former as SERVICE experiments and those using the latter as NO-SERVICE experiments.

All 67 queries were manually executed through the target endpoints' web interfaces to establish positive-control result sets for each query, reported in the *service-control-31-03-2025-endpoint* experiment [8].

To characterize workload complexity, we extracted per-query features from the curated corpus, including triple-pattern counts, `OPTIONAL` and `UNION` usage, property paths, `DISTINCT`, `LIMIT`, and the number of federation members. Corpus-level summaries are available on the companion results webpage. Queries contain 14.90 triple patterns on average, ranging from 2 to 34, and involve an average of 2.24 federation members. About half use `DISTINCT` (mean 0.54), while `LIMIT` is less common (mean 0.18). Property paths appear frequently (mean 1.27), but recursive paths are less common (mean 0.49), suggesting that complex path recursion is concentrated in a subset of queries. Most queries contain no `OPTIONAL` or `UNION` clauses, with both averaging about 0.15 occurrences per query. Figure 2, available in a larger, interactive version on the companion results webpage, presents a per-query view of workload complexity by ordering queries by triple-pattern count and grouping them by the number of federation members. Query features were extracted using a reproducible analysis script [6], and the full per-query feature table is available in the results repository [8].

Overall, the query set consists of moderately complex federated queries, typically spanning two or three sources, rather than simple joins over small fragments. At the same time, these queries are not dominated by advanced SPARQL

features, suggesting that execution difficulty arises primarily from federation itself rather than from unusually complex query syntax.

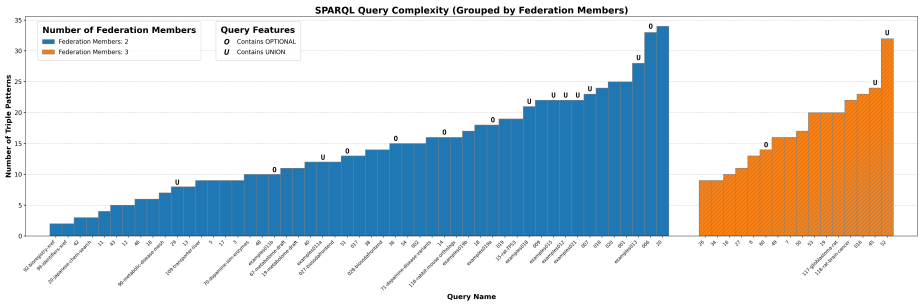

Fig. 2. Query complexity as a function of the number of triple patterns, federation members, and the presence of graph patterns. More complex queries are those with OGPs, with a higher number of triple patterns and federation members, followed by those with UGPs, following the same ordering in terms of triple patterns and federation members.

Of the original 67 queries, 7 encountered errors early in experimentation and were excluded from NO-SERVICE experiments.

Endpoints. The workload involved seven *target endpoints*, which receive complete SERVICE queries, and 14 *federates-with endpoints*, which are contacted remotely through embedded `SERVICE` clauses. The complete endpoint list, URLs, descriptions, triple-stores [11,4], and citations is available on the results webpage. During the study, METRIN-KG replaced the earlier DBGI/EMI endpoint. neXtProt [45] was excluded from the study because its public endpoint was no longer maintained.

4.3 Federation Approaches

The COMUNICA query engine framework [40] v4.3.0 was used for both server-side manual federation (SERVICE experiments) and client-side automated federation (NO-SERVICE experiments), while keeping the client-side execution environment constant. In SERVICE experiments, COMUNICA acts only as an external client: the complete query, including its explicit `SERVICE` clauses, is submitted to a single target endpoint, which is then responsible for federation tasks, carrying out remote requests, intermediate result joins, and returning the final results. In contrast, in NO-SERVICE experiments, with `SERVICE` clauses removed and multiple sources provided, COMUNICA performs client-side automated federation according to the predefined configuration. Thus, the same engine is used in both settings, but automated federation is only performed in NO-SERVICE experiments.

Table 1. Comunica federation configurations used in our server-side automated federation experiments. A checkmark indicates the presence, while a dash indicates the absence of an approach within a configuration. `-NRL` variants disable client-side request rate limiting while leaving other configuration choices unchanged.

Configuration	ASK	COUNT	VoID	Large est.	Rate limit
`NOMETA-ASK`	✓	–	–	–	✓
`NOMETA-COUNT`	–	✓	–	–	✓
`VOID-TRIPLE`	✓	–	✓	–	✓
`VOID-BLOCK`	✓	–	✓	✓	✓
`-NRL`	–	–	–	–	–

The client-side configurations in Table 1 isolate the main source-selection and planning assumptions tested in this study. NOMETA-ASK and NOMETA-COUNT rely on live endpoint probing, comparing lightweight source pruning against cardinality-oriented probing [37]. VOID-TRIPLE and VOID-BLOCK use available VoID descriptions and precomputed statistics at triple-pattern or larger query-block granularity [16,9,17]. The `-NRL` variants disable client-side rate limiting without changing the underlying federation strategy. All configurations were applied through the experiment repository's Comunica configuration script [7].

4.4 Experimental Set-up

Observation period. The testing campaign comprised 25 experiments conducted between March 2025 and June 2026. We therefore treat the study as longitudinal, with repeated executions of the same workload and configuration families at multiple time points.

Execution environment. Experiments ran on an Ubuntu 24.04.1 virtual machine with an Intel Core i5-9500 CPU (6 cores, 3.0 GHz), 64 GB RAM, a 468 GB NVMe SSD, and Linux kernel 6.8.0-45-generic. All queries targeted public endpoints, not local mirrors or replicated stores.

SERVICE experiment workflow. In SERVICE experiments, queries were executed in their original form, preserving explicit `SERVICE` clauses and submitting the complete query to a single target endpoint. Each query was submitted with a single target endpoint as a datasource while retaining its embedded `SERVICE` clauses.

NO-SERVICE experiment workflow. The reproducible NO-SERVICE workflow, documented in the study's experimental repository [7], follows four steps: (1)QUERY_SORT.PY, (2)COMUNICA_CONFIGURATION.PY, (3)COMUNICA_RUN.PY, and (4)LOCALIZE_RESULTS.SH & ORGANIZE_DATA.PY. These correspond to query preparation, configuration selection, execution, and post-processing / summary generation, respectively. During execution, `comunica_run.py` initializes a fresh Comunica engine instance via `query-dynamic.js` for each query executed. Datasources are read from the query header, SPARQL-JSON[38] is requested as output, debug logging is enabled, and standard error is written to a per-query log file from which experiment-level summaries are generated.

Timeouts and retries. The execution script does not impose a wrapper-level wall-clock timeout, but does record wall-clock runtime from the logged start and end timestamps. Query termination is therefore determined by completion, endpoint-side failure, engine-level failure, or external process termination, and timeout-like phenomena are captured indirectly through these measured timestamps and failure modes. To reduce rate-limit effects, the execution script uses `-httpRetryCount=50` and inserts a one-second pause between queries if an HTTP-429 error [28] is encountered.

Endpoint request burden and error types. During experiments, we tracked the number of HTTP requests per query and the final error text or category when execution failed. These data support three operational metrics: endpoint request burden, endpoint-associated error rate, and failure-type distribution. In a longitudinal public-endpoint setting, these metrics help distinguish whether execution outcome changes stem from client-side federation planning, network or endpoint instability, or endpoint-side request handling.

Definition of success and failure. An execution is considered to have *produced results* if it returns parseable output conforming to the SPARQL 1.1 Query Results JSON Format [38]. Empty and non-empty result sets are distinguished using result count. Executions that produce no parseable output or terminate through an exception/error path are treated as explicit failures. For the remainder of the study, we define *execution success rate* as the proportion of executions that have produced a non-empty result set (i.e., result count > 0).

Result correctness and temporal reproducibility. This study does not verify semantic correctness of individual bindings, which would require complete local replicas of all participating endpoints. Instead, we use *result-set size agreement* as a proxy: for each query, the reference size is the maximum result count observed under manual SERVICE execution, and later executions are considered consistent when they return the same count. Thus, "correctness" refers only to result-count agreement, not binding equality or semantic completeness. Temporal reproducibility is assessed by comparing identical query–configuration pairs across dated runs, including whether execution remained successful, whether non-empty results persisted, and whether error modes changed.

5 Results

We first compared the March 2025 SERVICE experiments to assess whether Comunica introduced systematic differences relative to direct endpoint execution. Manual endpoint execution performed slightly better than the Comunica-mediated control, with higher *execution success rates* (65.2% vs. 60.6%) and fewer explicit errors (30.3% vs. 34.8%). However, the close agreement suggests that Comunica did not introduce a major confounding effect in subsequent experiments.

An interactive all-query outcome heatmap, showing all experimental query executions chronologically, is shown in Figure 3 and is available in the *Overview Figures* section of the results webpage.

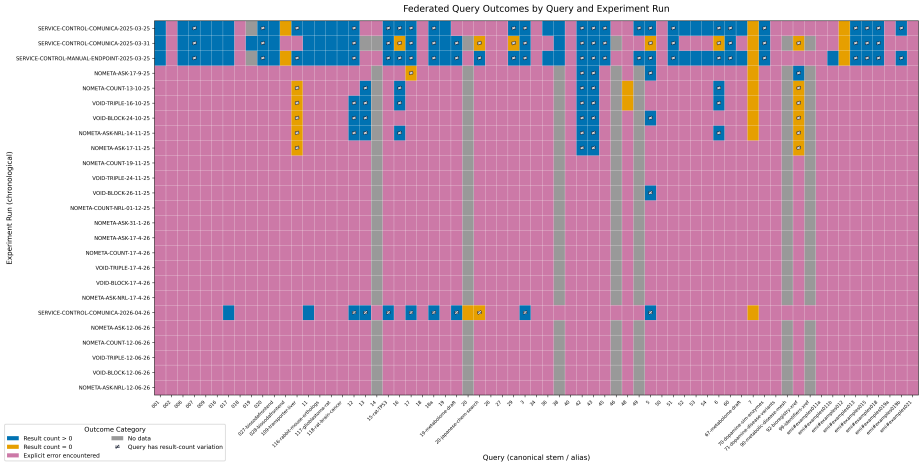

Fig. 3. Overview of federated query experiment outcomes across 25 experiment runs of the 67-query test set. Outcome matrix cells represent whether a query produced a non-empty result set (blue), an empty result set (yellow), an explicit error (pink), or data was missing for that run (gray). The heatmap shows that failures are widespread across the workload. For an interactive, expandable version of this figure see the results repository webpage.

Overall, SERVICE experiments performed far better than NO-SERVICE client-side automated federation experiments: SERVICE experiments had an *execution success rate* of 47.3%, and a 46.2% explicit error rate. In contrast, NO-SERVICE experiments had only a 2.3% *execution success rate*, and 96.3% failed with explicit errors. Thus, removing explicit `SERVICE` clauses and relying on automatic client-side federation reduced successful result production by roughly an order of magnitude and nearly doubled the error rate. NO-SERVICE experiments were also slower and more variable, with a median runtime of 4.47 seconds and a 90th percentile of 247.92 seconds, suggesting many failures involved severe slowdowns or stalled execution.

5.1 Temporal Degradation of Query Success

Figure 4 illustrates the degradation of federated query success chronologically, showing that more queries obtained results in chronologically earlier executions for both SERVICE and NO-SERVICE experiments. For ease of comparison, we separated experimental executions into 4 distinct chronological groups: Spring 2025, Fall 2025, Winter 2026, and Spring 2026. An in-depth, interactive data display of these time-based partitions can be viewed in the *Monthly Views* section of the results webpage.

During Spring 2025, which included only SERVICE experiments, performance was strongest, with 58.5% *execution success rate* and 34.2% explicit errors. By Fall 2025, *execution success rate* dropped to 8.5%, explicit errors rose

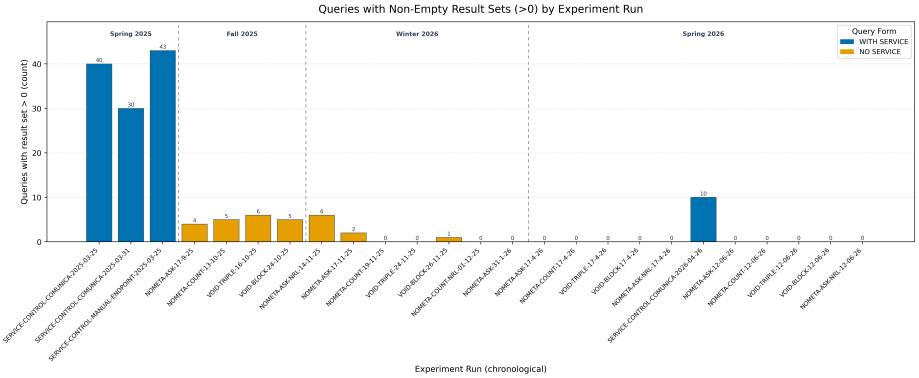

Fig. 4. The trend showing a decrease in the number of queries producing non-empty result sets in each run, grouped by experiment form. Explicit `SERVICE` control runs produced substantially more non-empty result sets than `no-service` runs, while later runs show a marked decline in successful result production.

to 85.9%, and runtimes became highly heavy-tailed, with the 90th percentile reaching approximately 1.07×10^4 seconds and a high HTTP request burden. Winter and Spring 2026 remained low-success, high-error phases, with *execution success rates* of 2.1% and 1.5%, and error rates of 96.7% and 98.05%, respectively. Notably, the Spring 2026 SERVICE experiments also performed far worse than Spring 2025, with only 14.9% *execution success rate*. Although extreme runtime tails decreased by Spring 2026, successful result production did not recover. Overall, Figure 4 shows this clear temporal decline in queries producing results.

Beginning in Fall 2025, many previously successful queries began failing after only 3–6 HTTP requests and roughly 4 seconds. Per-query logs show repeated `Server reported client-side error` events for `https://sparql.uniprot.org/sparql/`, with HTTP `403 Forbidden` errors, including for formerly stable queries such as `42`, `12`, and `16`. This indicates that server-side restrictions became a major contributor to the observed drop in success. According to the primary UniProt server administrator, this period coincided with increased AI-crawler traffic, including crawlers that ignored `robots.txt` guidelines discussed further in Section 6.2.

Even when queries produced results, outputs were often unstable across repeated runs. Overall, 68.7% of queries returned varying result counts, and 24 queries produced three or more distinct result-count outcomes. For example, `92_uniprot_bioregistry_iri_translation` returned 194 results in Fall 2025, later returned parseable but empty outputs, and eventually failed in Winter and Spring 2026. Thus, live federation can degrade not only through hard failures, but also through empty or otherwise reduced result sets.

5.2 Explicit Error Taxonomy

Across experiments, failures clustered into recurring categories, visualized in the *Query error-type heatmap by experiment run* on the results webpage. The most frequent were CLIENT-SIDE HTTP errors (n=850; 4xx endpoint rejections), SERVER-SIDE HTTP errors (n=242; 5xx failures), OTHER errors (n=102; e.g., HTTP 200 responses containing HTML error pages), TRANSPORT-LEVEL failures (n=82; incomplete responses such as `fetch failed`), and TIMEOUT-RELATED errors (n=63). These failures reflect both endpoint-side constraints, including HTTP 403 rejections, server errors, malformed responses, and timeout-like interruptions, and client-side federation behavior, including large intermediate joins, heap exhaustion, and request explosion.

5.3 HTTP Requests and Queries that Obtain Results

We examined whether successful NO-SERVICE executions differed from failures in HTTP request burden, restricting the analysis to runs with valid request counts ($n = 1260$). Because request counts were highly skewed, we report both means and medians, and use permutation tests and Mann–Whitney U tests as assumption-light complements to correlation estimates [3,15,24].

Under a broad success definition, where success means any execution without an explicit error, successful runs issued more requests than failures on average (46,366 vs. 4,161) and by median (22 vs. 5). The association was positive but modest ($r = 0.223$) and supported by permutation tests ($p = 3.33 \times 10^{-5}$ for correlation; $p = 3.33 \times 10^{-5}$ for mean difference) and by a Mann–Whitney U test ($p = 1.33 \times 10^{-15}$, one-sided). Using a stricter definition of success as a non-empty result set, successful executions again showed higher request burden, both in mean (68,085 vs. 4,267) and median (580 vs. 5). The association remained moderate ($r = 0.266$) and was again supported by permutation tests ($p = 3.33 \times 10^{-5}$ for correlation; $p = 3.33 \times 10^{-5}$ for mean difference) and by a Mann–Whitney U test ($p = 2.04 \times 10^{-12}$, one-sided).

Overall, successful client-side automatic federation was not associated with fewer endpoint interactions. Non-empty results often required more HTTP activity, likely due to source probing, remote subquery execution, and intermediate-result transfer. However, the large mean–median gaps show that request burden remained highly uneven and dominated by outliers, making HTTP request count alone an incomplete predictor of success. The most extreme HTTP request explosion cases issued hundreds of thousands to millions of HTTP requests before failing, examples include query `6` with 924,299 requests, query `13` with 742,417, and query `16` with 553,202 in experiment `NOMETA-ASK-17-9-25`. Query- and experiment-specific plots are available in the *Data Explorer* section of the results webpage.

5.4 Comparison across federation approaches

Across the NO-SERVICE experiments, all client-side automated federation algorithm families failed in most executions, with *execution success rates* between

only 1.67% and 2.50%. The corresponding counts were 12/480 for NOMETA-ASK, 5/300 for NOMETA-COUNT, 6/240 for VOID-TRIPLE, and 6/240 for VOID-BLOCK. These results are visualized in the NO-SERVICE *algorithm efficacy comparison* graph on the results webpage.

Thus, all algorithms succeeded in a small minority of cases, but none were broadly robust for the tested workload. Differences between methods were modest: VOID-TRIPLE had the highest explicit-error-free rate, VOID-BLOCK the highest *execution success rate*, and NOMETA-COUNT was consistently weakest. Success also varied substantially across runs, including multiple with 0% success, indicating general instability. Overall, VoID-based *metadata-guided* strategies provided only a small advantage over *metadata-free* strategies in NO-SERVICE experiments.

The -NRL experiments show that client-side rate-limit mitigation did not seem to impact *execution success rates*. Similar success ratios are observed for queries completed in Fall 2025 with and without rate-limit mitigation, and by Spring 2026, all configurations converged to complete failure. This suggests that client-side naive throttling or backoff cannot overcome tighter server-side policies.

5.5 Unique Study Challenges and Endpoint Instability

A number of failures were caused by endpoint address changes outside the experiment's control. Queries involving the former DBGI/EMI endpoint (14/67 queries) were affected by its migration to the production METRIN-KG endpoint URL. Similarly, `https://biosoda.unil.ch/emi/sparql`, a temporary federation server targeted in (14/67 queries), was deprecated in September 2025 and was removed from service in November 2025. These changes contributed to higher failure rates in Winter and Spring 2026 experimental runs. For an in-depth view of endpoint-specific results by experiment, see the *Endpoint outcomes by run* graph on the results webpage.

Although this complicates longitudinal interpretation, it reflects a central challenge of real-world SPARQL federation: queries depend on the continued availability, stability, and discoverability of independently maintained endpoints. When endpoints are moved, renamed, deprecated, or taken offline without redirects or client-interpretable explanations, users and engines typically observe only execution failure. Such changes are therefore not merely experimental noise, but examples of how federation breaks when endpoint-level compatibility is not preserved.

6 Discussion

This study asks whether SPARQL federation works in the real world. We found that it can work in some cases, but it was not reliable for our complex biological query set over public endpoints.

6.1 Manual and automated federation

Manual federation with explicit `SERVICE` clauses degraded substantially over the study period. Because the workload came from previously successful user queries, the Spring 2025 runs supported our expectation that many queries would remain executable. By Spring 2026, however, only 10 queries produced results, compared with more than 40 that did so consistently in Spring 2025. This decline reflects endpoint migrations, deprecations, and changing limit policies, showing that even manually specified federation can lose executability as independently maintained endpoints evolve.

Client-side automated federation was substantially more fragile and later became effectively non-viable. The fact that many of the same queries worked with explicit `SERVICE` clauses shows that the data and endpoints were not inherently unusable; failures arose when client-side engines had to infer source selection and plans under incomplete metadata, limited coordination, and public-service constraints. Two failure modes dominated: request explosion, where a single query produced hundreds of thousands of HTTP requests, and excessive local materialization, where large intermediate results exhausted client memory. From the endpoint perspective, such request-heavy plans can resemble abusive traffic; from the client perspective, they reveal planning assumptions that do not hold under live deployment constraints.

6.2 Temporal instability and endpoint operations

SPARQL federation was not temporally consistent. Queries that succeeded at one time point later returned empty results or failed under the same client-side configuration, and Figure 4 shows a clear decline in execution success over time. Some change is expected in public infrastructure: datasets grow, endpoints move, backends change, and temporary services are deprecated. For example, during the study, UniProt had three data releases, grew by approximately 28 billion triples, and migrated from Virtuoso 7.2+ Open Source edition to QLever. Because of the backend changes, some UniProt queries to `https://sparql.api.identifiers.org/` began failing because the identifiers.org endpoint did not support the SPARQL protocol case where a query is sent as the body of a POST request. These changes in public infrastructure pose challenges to federation but are feasible to remedy as discussed in Section 8

A faster-moving challenge was observed in stricter server-side limits in response to bot and crawler traffic. According to the lead UniProt server administrator, UniProt's SPARQL endpoint historically saw tens to hundreds of thousands of visitors per month between 2020 and 2024, but in Summer 2025, some months exceeded three million visitors, without a comparable increase in hardware. Traffic remained around five times higher than the 2024 baseline in 2026. UniProt therefore tightened limits on response time, concurrent queries per IP, request rates, and daily request volume. For federation, the main problem is that such limits, migrations, and deprecations are rarely communicated in machine-readable ways that client engines can interpret; users often see only failure and errors.

6.3 Implications for real-world federation

Public biological SPARQL endpoints are shared research infrastructure, typically optimized for interactive exploration, documented examples, and moderate programmatic access. This creates a mismatch with automated federation: a single user query can be decomposed into many rapid requests, large intermediate transfers, or long-running connections. As a result, an endpoint may be reliable for single-endpoint use while still being poorly suited to unrestricted cross-endpoint federation.

This mismatch also limits what controlled benchmarks can show. Benchmarks remain valuable for isolating algorithmic effects, but they can overestimate real-world viability when they abstract away from endpoint instability, incomplete metadata, request policing, backend changes, cache behavior, public-service load, and temporal drift. Practical federation therefore, requires more research into cooperation between servers and clients. Endpoint providers should invest in clearer operational metadata, migration signals, and machine-readable limits, while client-side engines should integrate request budgets, endpoint-aware planning, adaptive backoff, diagnostics, and safeguards against request explosion.

For heavier workloads, algorithmic improvements alone may not be sufficient. More sustainable architectures, such as negotiated access, mirrors, caches, precomputed source summaries, controlled federation gateways, or hybrid deployments, may be necessary. The main lesson is not that SPARQL federation is impossible, but that open-ended federation over public endpoints should not be assumed to be operationally sustainable by default.

7 Limitations

This study has three main limitations. First, it is domain-specific: the workload and endpoints come from the biological Linked Data ecosystem and should not be generalized automatically to all SPARQL federation settings. Second, the corpus contains 67 real-world queries over a limited set of public endpoints, which is meaningful but not exhaustive. Third, changing endpoint behavior makes causal attribution difficult; the artifacts show when outcomes changed, but not always the precise administrative or infrastructural cause.

8 Practical Implications and Recommendations

This study examines a demanding but high-value use case: complex biological federation over authoritative public resources maintained by communities that recognize its scientific value. The results do not disprove federation for simpler settings, but they show that current algorithms, protocols, and infrastructure are not robust enough for this realistic workload.

The practical priority is to make federation more operationally aware. Researchers should evaluate algorithms longitudinally over public or public-like endpoints, reporting not only runtime and result counts but also failure modes,

temporal reproducibility, and request burden. Engine developers should investigate adding request budgets, rate-limit-aware planning, adaptive backoff, result caching, and actionable diagnostics. Endpoint maintainers should expose machine-readable limits, retry behavior, redirects, supported request forms, pagination expectations, and service status. For workloads beyond what public interactive endpoints can sustainably support, controlled federation gateways, negotiated access, mirrors, shared caches, precomputed source summaries, or low-cost publishing approaches such as Triple Pattern Fragments [43] may be needed.

9 Conclusion

This study presented an in-use evaluation of SPARQL federation over large public biological endpoints, using real-world federated queries and multiple client-side federation strategies. We broadly assessed the question: "Does SPARQL federation work in the real world?" Generally, we found that execution success declined sharply over time, as public endpoints changed and tightened limits that client-side federation engines could not adapt to.

The main lesson is that SPARQL federation is not only an algorithmic problem, but an operational one. Even when the data exist and the queries are meaningful, successful execution depends on the alignment of query planning, endpoint availability, metadata, rate limits, backend behavior, and network reliability. When that alignment breaks, federation fails not as a Semantic Web principle, but as a practical workflow: queries return errors, empty outputs, or unstable results. Crucially, this delicate balance between data requester and data provider is not restricted to the public SPARQL endpoints tested in this study, but highlights significant emerging challenges to modern public data access in general.

SPARQL federation remains conceptually powerful, but dependable live federation over public endpoints cannot be assumed. Without federation-aware infrastructure, clearer endpoint policies, better diagnostics, and planners designed for public-service constraints, distributed Knowledge Graphs risk remaining theoretically linked but operationally difficult to integrate.

Acknowledgments. Elias Crum was supported by the Research Foundation – Flanders (FWO) (SB Fellowship 1S27825N) and (Long Research Stay V416635N). Ruben Taelman is an FWO postdoctoral fellow (1202124N). Tarcisio Mendes de Farias and Ana Sima were supported by the Swiss National Science Foundation through the MetaboLinkAI project (10.002.786), European Union's Horizon Europe research and innovation programme (grant No. 101131818 - project MICROBES-4-CLIMATE), and the Swiss Federal Government State Secretariat for Education, Research and Innovation (SERI). Jonni Hanski and Bryan-Elliot Tam were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Jerven Bolleman as member of the Swiss-Prot group and UniProt was supported by the NHGRI, Office of Director [OD/DPCPSI/ODSS]; NIAID, NIA, NIGMS, NIDDK, NEI, NCI, and NHLBI of the National Institutes of Health (NIH) [U24HG007822]; additionally supported by the Swiss Federal Government through SERI.

Declaration on Generative AI

The authors used ChatGPT-5.5 (OpenAI) to assist with grammar suggestions, spell-checking, LaTeX and BibTeX formatting, and create the result webpage[8] during manuscript preparation. The tool was not used to generate research data, perform experiments, conduct statistical analyses, or draw scientific conclusions. All AI-assisted text was reviewed, edited, and approved by the authors, who take full responsibility for the content of the manuscript.

References

1. Acosta, M., Vidal, M.E., Lampo, T., Castillo, J., Ruckhaus, E.: Anapsid: an adaptive query processing engine for sparql endpoints. In: Proceedings of the 10th International Conference on The Semantic Web - Volume Part I. p. 18–34. ISWC'11, Springer-Verlag, Berlin, Heidelberg (2011), https://dl.acm.org/doi/10.5555/2063016.2063019
2. Aimonier-Davat, J., Nédelec, B., Dang, M.H., Molli, P., Skaf-Molli, H.: Fedup: Querying large-scale federations of sparql endpoints. In: Proceedings of the ACM Web Conference 2024. p. 2315–2324. WWW '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3589334.3645704, https://doi.org/10.1145/3589334.3645704
3. Altman, D.G., Bland, J.M.: Statistics notes: Quartiles, quintiles, centiles, and other quantiles. BMJ **309**(6960), 996 (1994). https://doi.org/10.1136/bmj.309.6960.996
4. Bast, H., Buchhold, B.: Qlever: A query engine for efficient sparql+text search. p. 647–656. CIKM '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3132847.3132921
5. Bolleman, J., Emonet, V., Altenhoff, A., Bairoch, A., Blatter, M.C., Bridge, A., Duvaud, S., Gasteiger, E., Kuznetsov, D., Moretti, S., Michel, P.A., Morgat, A., Pagni, M., Redaschi, N., Zahn-Zabal, M., de Farias, T.M., Sima, A.C.: A large collection of bioinformatics question–query pairs over federated knowledge graphs: methodology and applications. GigaScience **14**, giaf045 (01 2025). https://doi.org/10.1093/gigascience/giaf045, https://doi.org/10.1093/gigascience/giaf045
6. constraintAutomaton: Query analysis for sib swiss federated queries (2026), https://github.com/constraintAutomaton/query-analysis-sib-swiss-federated-query
7. Crum, E.: Federated sparql survey experiments (2026), https://github.com/ecrum19/fed-sparql-survey-expts
8. Crum, E.: Federated sparql survey results (2026), https://github.com/ecrum19/fed-survey-results
9. Cyganiak, R., Alexander, K., Zhao, J., Hausenblas, M.: Describing linked datasets with the VoID vocabulary. W3C note, W3C (2011), https://www.w3.org/TR/void/
10. Dang, M.H., Aimonier-Davat, J., Molli, P., Hartig, O., Skaf-Molli, H., Le Crom, Y.: Fedshop: A benchmark for testing the scalability of sparql federation engines. In: The Semantic Web – ISWC 2023. Lecture Notes in Computer Science, vol. 14266, pp. 285–301. Springer (2023). https://doi.org/10.1007/978-3-031-47243-5_16

11. Erling, O., Mikhailov, I.: Rdf support in the virtuoso dbms. In: Pellegrini, T., Auer, S., Tochtermann, K., Schaffert, S. (eds.) Networked Knowledge – Networked Media, Studies in Computational Intelligence, vol. 221. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02184-8_2
12. Feigenbaum, L., Williams, G.T., Clark, K.G., Torres, E.: SPARQL 1.1 protocol. W3c recommendation, World Wide Web Consortium (W3C) (Mar 2013), https://www.w3.org/TR/sparql11-protocol/
13. Fu, G., Batchelor, C., Dumontier, M., et al.: Pubchemrdf: towards the semantic annotation of pubchem compound and substance databases. Journal of Cheminformatics **7**, 34 (2015). https://doi.org/10.1186/s13321-015-0084-4
14. Galgonek, J., Vondrášek, J.: Idsm chemwebrdf: Sparqling small-molecule datasets. Journal of Cheminformatics **13**, 38 (2021). https://doi.org/10.1186/s13321-021-00515-1
15. Good, P.I.: Permutation, Parametric, and Bootstrap Tests of Hypotheses. Springer, 3 edn. (2005). https://doi.org/10.1007/b138696
16. Görlitz, O., Staab, S.: Splendid: Sparql endpoint federation exploiting void descriptions. In: Proceedings of the Second International Conference on Consuming Linked Data – COLD'11. p. 13–24. COLD'11, CEUR-WS.org, Aachen, DEU (2011), https://dl.acm.org/doi/10.5555/2887352.2887354
17. Hagedorn, S., Hose, K., Sattler, K.U., Umbrich, J.: Resource planning for sparql query execution on data sharing platforms. In: 5th International Conference on Consuming Linked Data. pp. 49–60 (2014), https://ceur-ws.org/Vol-1264/cold2014_HagedornHSU.pdf
18. Hasnain, A., Mehmood, Q., Sana, E., Zainab, S., Saleem, M., Warren, C.N.J., Zehra, D., Decker, S., Rebholz-Schuhmann, D.: Biofed: federated query processing over life sciences linked open data. Journal of Biomedical Semantics **8**(1), 13 (Mar 2017). https://doi.org/10.1186/s13326-017-0118-0
19. Hernández, D., Hogan, A., Riveros, C., Rojas, C., Zerega, E.: Querying wikidata: Comparing sparql, relational and graph databases. In: The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II. p. 88–103. Springer-Verlag, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-319-46547-0_10, https://doi.org/10.1007/978-3-319-46547-0_10
20. Hogan, A., Gutierrez, C., Cochez, M., de Melo, G., Kirrane, S., Polleres, A., Navigli, R., Ngomo, A.C.N., Rashid, S.M., Schmelzeisen, L., Staab, S., Blomqvist, E., d'Amato, C., Gayo, J.E.L., Neumaier, S., Rula, A., Sequeda, J., Zimmermann, A.: Knowledge Graphs. Springer Cham (2022). https://doi.org/10.1007/978-3-031-01918-0
21. Khan, Y., Saleem, M., Mehdi, M., Hogan, A., Mehmood, Q., Rebholz-Schuhmann, D., Sahay, R.: Safe: Sparql federation over rdf data cubes with access control. Journal of Biomedical Semantics **8**(1), 5 (2017). https://doi.org/10.1186/s13326-017-0112-6
22. Lubiana, T., Rasberry, L., Mietchen, D.: The wikidata query service split and its impact on the scholarly graph. In: Posters, Demos, Workshops, and Tutorials at the 21st International Conference on Semantic Systems (SEMANTiCS 2025). CEUR Workshop Proceedings, vol. 4064 (2025), https://ceur-ws.org/Vol-4064/PD-paper3.pdf
23. Maillot, P., Corby, O., Faron, C., Gandon, F., Michel, F.: Kartographi: Drawing a map of linked data. In: The Semantic Web: ESWC 2022 Satellite Events. p. 112–117. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-11609-4_21

24. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics **18**(1), 50–60 (1947). https://doi.org/10.1214/aoms/1177730491
25. Montoya, G., Skaf-Molli, H., Hose, K.: The odyssey approach for optimizing federated sparql queries. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (eds.) The Semantic Web – ISWC 2017. pp. 471–489. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_28
26. Moreau, B., Serrano-Alvarado, P.: Ensuring License Compliance in Linked Data with Query Relaxation, pp. 97–129. Springer Berlin Heidelberg, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64148-4_4
27. Morgat, A., Lombardot, T., Coudert, E., Axelsen, K., Batista Neto, T., Géhant, S., Bansal, P., Bolleman, J., Gasteiger, E., de Castro, E., Baratin, D., Pozzato, M., Xenarios, I., Poux, S., Redaschi, N., Bridge, A., The UniProt Consortium: Enzyme annotation in uniprotkb using rhea. Bioinformatics **36**(6), 1896–1901 (March 2020). https://doi.org/10.1093/bioinformatics/btz817
28. Nottingham, M., Fielding, R.T.: Additional HTTP Status Codes. RFC 6585 (April 2012). https://doi.org/10.17487/RFC6585, https://www.rfc-editor.org/rfc/rfc6585
29. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Trans. Database Syst. **34**(3) (Sep 2009). https://doi.org/10.1145/1567274.1567278, https://doi.org/10.1145/1567274.1567278
30. Pham, T.H.T., Montoya, G., Nédelec, B., Skaf-Molli, H., Molli, P.: Passage: Ensuring completeness and responsiveness of public sparql endpoints with sparql continuation queries. In: Proceedings of the ACM on Web Conference 2025. p. 47–58. WWW '25 (2025). https://doi.org/10.1145/3696410.3714757
31. Polleres, A., Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngonga Ngomo, A.C.: A fine-grained evaluation of sparql endpoint federation systems. Semant. Web **7**(5), 493–518 (Jan 2016). https://doi.org/10.3233/SW-150186
32. Saleem, M., Hasnain, A., Ngonga Ngomo, A.C.: Largerdfbench: A billion triples benchmark for sparql endpoint federation. Journal of Web Semantics **48**, 85–125 (2018). https://doi.org/https://doi.org/10.1016/j.websem.2017.12.005
33. Saleem, M., Ngonga Ngomo, A.C.: Hibiscus: Hypergraph-based source selection for sparql endpoint federation. In: The Semantic Web: Trends and Challenges. Lecture Notes in Computer Science, vol. 8465, pp. 176–191. Springer (2014). https://doi.org/10.1007/978-3-319-07443-6_13
34. Saleem, M., Potocki, A., Soru, T., Hartig, O., Ngonga Ngomo, A.C.: Costfed: Cost-based query optimization for sparql endpoint federation. In: Proceedings of the 14th International Conference on Semantic Systems, SEMANTICS 2018. Procedia Computer Science, vol. 137, pp. 163–174. Elsevier (2018). https://doi.org/10.1016/j.procs.2018.09.016
35. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing. In: The Semantic Web – ISWC 2011. Lecture Notes in Computer Science, vol. 7031, pp. 585–600. Springer (2011). https://doi.org/10.1007/978-3-642-25073-6_37
36. Schmidt, M., Meier, M., Lausen, G.: Foundations of sparql query optimization. In: Proceedings of the 13th International Conference on Database Theory. p. 4–33. ICDT '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1804669.1804675, https://doi.org/10.1145/1804669.1804675

37. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization techniques for federated query processing on linked data. In: The Semantic Web – ISWC 2011. Lecture Notes in Computer Science, vol. 7031, pp. 601–616. Springer (2011). https://doi.org/10.1007/978-3-642-25073-6_38
38. Seaborne, A.: Sparql 1.1 query results json format. W3C Recommendation (2013), https://www.w3.org/TR/sparql11-results-json/
39. SIB Swiss Institute of Bioinformatics RDF Group Members: The sib swiss institute of bioinformatics semantic web of data. Nucleic Acids Research **52**(D1), D44–D51 (Jan 2024). https://doi.org/10.1093/nar/gkad902
40. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: A modular sparql query engine for the web. In: ISWC 2018: 17th International Semantic Web Conference. p. 239–255. Springer-Verlag, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-030-00668-6_15
41. The UniProt Consortium: Uniprot: the universal protein knowledgebase in 2025. Nucleic Acids Research **53**(D1), D609–D617 (Jan 2025). https://doi.org/10.1093/nar/gkae1010
42. Vandenbussche, P.Y., Umbrich, J., Hogan, A., Buil-Aranda, C.: SPARQLES: Monitoring public SPARQL endpoints. Semantic Web **8**(6), 1049–1065 (2017). https://doi.org/10.3233/SW-170254
43. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple pattern fragments: A low-cost knowledge graph interface for the web. Journal of Web Semantics **37–38**, 184–206 (2016). https://doi.org/10.1016/j.websem.2016.03.003
44. Willighagen, E.L., Waagmeester, A., Spjuth, O., et al.: The chembl database as linked open data. Journal of Cheminformatics **5**, 23 (2013). https://doi.org/10.1186/1758-2946-5-23
45. Zahn-Zabal, M., Michel, P.A., Gateau, A., Nikitin, F., Schaeffer, M., Audot, E., Gaudet, P., Duek, P.D., Teixeira, D., Rech de Laval, V., Samarasinghe, K., Bairoch, A., Lane, L.: The nextprot knowledgebase in 2020: data, tools and usability improvements. Nucleic Acids Research **48**(D1), D328–D334 (January 2020). https://doi.org/10.1093/nar/gkz995