Technical Appendix for InteracSPARQL

Xiangru Jian
University of Waterloo
Canada
xiangru.jian@uwaterloo.ca

Zhengyuan Dong
University of Waterloo
Canada
zhengyuan.dong@uwaterloo.ca

M. Tamer Özsu
University of Waterloo
Canada
tamer.ozsu@uwaterloo.ca

1 EXTENDED METHODOLOGY

Here, we provide an extended version of the methodology described in the main paper, elaborated in the exact identical set of subsections but with additional details and examples.

1.1 Parsing and Representation of Original SPARQL Query

The process of converting an original SPARQL query into a form amenable to NLE and iterative refinement involves several key steps. We begin with the user's input and systematically transform it into a structured format.

Original SPARQL Queries. The pipeline starts with an initial SPARQL query, which may be provided directly by a user or produced by a text-to-SPARQL model from on a given natural language expression. This original query, while functionally valid, typically remains opaque and difficult to understand without significant domain expertise. **Parsing SPARQL Queries.** To address this opacity, we parse the original SPARQL query into a JSON-based AST. This parsing step is essential for delineating the query's logical structure. The resulting AST highlights key elements—such as `SELECT`, `FILTER`, and BGPs—and identifies variables and entities, as illustrated in the example in Example 2. This structured form enables systematic analysis: we can isolate specific clauses, pinpoint variables of interest, and comprehend the hierarchical composition of the query, not only for humans but also for LLMs.

1.2 Natural Language Explanation

Producing clear, accurate, and intuitive explanations for SPARQL queries poses significant challenges, particularly when aiming to support iterative refinement by both human users and language models. To address this, we introduce a two-stage approach that carefully balances clarity, accuracy, and computational efficiency. This section guides readers through each stage, emphasizing how our design choices facilitate a seamless refinement experience.

Initially, we employ a structured, rule-based technique to extract precise, deterministic explanations directly from AST (Section 1.2.1). This foundational step ensures each SPARQL query element is clearly represented, thereby providing an interpretable, reliable baseline. Subsequently, these structured explanations are passed into an LLM, enriched with carefully selected few-shot examples (Section 1.2.2). Leveraging the rule-based foundation allows the LLM to concentrate on linguistic refinement—enhancing readability and capturing nuanced contextual insights—without sacrificing factual accuracy.

Example 1: A SPARQL Query Example in `QALD-10`

```
SELECT ?tvShow WHERE {
  ?tvShow wdt:P31 wd:Q5398426;
          wdt:P161 wd:Q23760;
          wdt:P2437 ?seasons;
          wdt:P580 ?startDate.
  FILTER(?seasons = 4)
  FILTER(YEAR(?startDate) = 1983)}
```

1.2.1 Extracting Rule-based NLE from the AST. Once the JSON-based AST has been constructed, we employ a rule-based strategy to produce an initial NLE for each query. This approach incrementally traverses the AST, extracting structural elements (such as BGPs, `FILTER`, `UNION`, etc.) and converting them into concise, human-readable sentences. By design, each element of the query is mapped to a clear textual statement (e.g., *"The entity* `X` *has the property* `Y`*, and its value is* `Z`*."*), ensuring transparency and interpretability of even complex SPARQL constructs like `UNION` or nested subqueries. The generation of the NLE given the query in Example 1 is provided in Example 3.

Hierarchical Parsing and Explanation. The AST parsing yields a hierarchical view of the query, which the rule-based mechanism leverages to traverse nodes in an ordered fashion systematically. At each node, we identify the specific clause (e.g., `FILTER`) and generate short explanatory text describing its semantics. This modular process exposes smaller units of meaning, making it easier to isolate where a query might be refined or expanded. Thus, the explanation is not merely a linear paraphrase; it preserves the tree-like structure of the query and ensures consistency across repeated patterns or sub-clauses.

Entity/Predicate Label Enrichment. To further enhance clarity, we optionally integrate the knowledge base on which the query in question is based (e.g., Wikidata for the example query in Example 3) to enrich original entity identifiers with labels. This transformation minimizes cognitive overhead by turning opaque references (e.g., `wd:Q5398426`) into meaningful names like *"television series"*. Such label enrichment is particularly useful when multiple identifiers are present or when the query references unfamiliar entities or predicates, as it provides readers with immediate context about the underlying data.

Advantages and Downstream Benefits. Our rule-based NLE technique offers a fast and cost-effective means to produce a fundamental interpretation of SPARQL queries before turning to large-scale language models. Specifically, we achieve:

- **Low Overhead:** The explanation is generated locally based on a set of deterministic transformations, requiring minimal computational resources compared to a full LLM pipeline.

Example Parsed SPARQL Query

```
{'queryType':'SELECT','variables':[{'termType':'Variable','value':'tvShow'}],
 'where':[
  {'type':'bgp','triples':[
    {'subject':{'termType':'Variable','value':'tvShow'},
     'predicate':{'termType':'NamedNode','value':'wdt:P31'},
     'object':{'termType':'NamedNode','value':'wd:Q5398426'}},
    {'subject':{'termType':'Variable','value':'tvShow'},
     'predicate':{'termType':'NamedNode','value':'wdt:P161'},
     'object':{'termType':'NamedNode','value':'wd:Q23760'}},
    {'subject':{'termType':'Variable','value':'tvShow'},
     'predicate':{'termType':'NamedNode','value':'wdt:P2437'},
     'object':{'termType':'Variable','value':'seasons'}},
    {'subject':{'termType':'Variable','value':'tvShow'},
     'predicate':{'termType':'NamedNode','value':'wdt:P580'},
     'object':{'termType':'Variable','value':'startDate'}}
  ]},
  {'type':'filter','expression':{'type':'operation','operator':'=',
    'args':[{'termType':'Variable','value':'seasons'},
            {'termType':'Literal','value':'4','language':'','datatype':{'termType':'NamedNode','value':'xsd:integer'}}]}},
  {'type':'filter','expression':{'type':'operation','operator':'=',
    'args':[{'type':'operation','operator':'year','args':[{'termType':'Variable','value':'startDate'}]},
            {'termType':'Literal','value':'1983','language':'','datatype':{'termType':'NamedNode','value':'xsd:integer'}}]}}
 ],
 'type':'query','prefixes':{'wd':'http://www.wikidata.org/entity/',
                            'wdt':'http://www.wikidata.org/prop/direct/'}}
```

Example 2: Parsed AST representation of the SPARQL query from Example 1 with semantic highlights: *SELECT (blue), Modules (red), Variable (purple), NamedNode/Literal (green), prefixes (brown).*

- **Consistent Output:** Because each query component is handled by pre-defined rules, the resulting explanations follow a stable, predictable format that is easy to understand and verify.
- **Effective Primer for LLMs:** Once the query structure has been distilled into a set of concise statements, it becomes significantly simpler to feed this *pre-digested* explanation into an LLM for further refinement or domain-specific customization. This staged process reduces the risk of hallucinations of LLMs and improves overall quality by guiding the model with a high-level yet accurate skeleton of the query.

As a result, the rule-based approach serves as an essential first step in our broader pipeline, bridging the gap between original SPARQL syntax and NLEs while retaining the flexibility to integrate with advanced LLMs in subsequent phases.

1.2.2 Refinement of NLE via LLMs: Structured Guidance and Enriched Narrative. Building upon the concise but sometimes brief explanations generated by the rule-based approach (Section 1.2.1), we employ LLMs to transform these systematically derived outputs into a more fluent, context-rich narrative. This section provides a comprehensive overview of our second-stage design, detailing both the overarching workflow and the structured format of how the LLM should present its refined explanations.

High-Level Workflow. In this phase, the LLM receives the rule-based NL explanation sourced from the method mentioned in the previous section, along with a carefully crafted *Instruction* that dictates both the structure and style of the final text. Crucially, this design ensures the model's creative enhancements remain grounded in verified semantic content. Rather than interpreting the original SPARQL query afresh, the LLM elaborates on pre-validated elements (e.g., identified triple patterns, filters, or subqueries). This tightly controlled setup mitigates erroneous expansions and encourages the model to add value primarily in terms of linguistic clarity and domain-specific insights. To make sure LLMs' generation strictly follows the format, we manually annotate NLEs a set of queries (i.e. few-shot examples) with diverse coverage of patterns, following the detailed designs described in the *Instruction* as following sections.

Structured Specification of the Design. Our *Instruction* directs the LLM to produce a layered JSON structure that mirrors the core components of a SPARQL query, including the overall query purpose, specific clauses, and advanced operators. This structured format is illustrated in Figure 4, showing how each code fragment and explanation is mapped in a consistent manner.

A. Overall NL Explanation. This section introduces the primary aim of the SPARQL query, summarizing how different segments (for instance, a `BGP` or a `FILTER`) together fulfill the question's objective. By providing a concise summary up front, readers can grasp the query's overarching intention, such as clarifying why a "manner of death" query for a living person might naturally yield no results.

Example 3: The rule-based natural language explanation for the query in Example 1 based on the parsing result.

Rule-Based Natural-Language Explanation

1. Query Type:
This query is asking to SELECT data.

2. Distinct:
The query does not request DISTINCT results.

3. Variables:
The query contains the following variables:
3.1 Variables.var.1:
The query returns the variable 'tvShow'.

4. Patterns:
The following conditions should be satisfied:
4.1. The Basic Graph Pattern (BGP)
includes the following statements:
4.1.1. Triple 1:
The entity 'tvShow' has the property http://www.wikidata.org/prop/direct/P31 (*instance of*), and its value is http://www.wikidata.org/entity/Q5398426 (*television series*).
4.1.2. Triple 2:
The entity 'tvShow' has the property http://www.wikidata.org/prop/direct/P161 (*cast member*), and its value is http://www.wikidata.org/entity/Q23760 (*Rowan Atkinson*).
4.1.3. Triple 3:
The entity 'tvShow' has the property http://www.wikidata.org/prop/direct/P2437 (*number of seasons*), and its value is the variable 'seasons'.
4.1.4. Triple 4:
The entity 'tvShow' has the property http://www.wikidata.org/prop/direct/P580 (*start time*), and its value is the variable 'startDate'.

4.2. Filter on seasons
A filter is applied to include only those results where the value of variable 'seasons' = literal '4'.

4.3. Filter on startDate year
A filter is applied to include only those results where the year of variable 'startDate' = literal '1983'.

B. Query Type. Here, the output identifies whether the query is SELECT, ASK, or another form, together with an explanation of how it interacts with the dataset. For a SELECT query, the explanation clarifies which variables are retrieved and how they relate to the query's purpose. This helps readers understand exactly what the query returns and how data is being filtered or combined.

C. Variable. Every variable used in the query is described both by name (e.g., ?varName) and by a short rationale that links it to the broader retrieval logic. The goal is to let users recognize a variable's role—for example, whether it captures the main entity or a subordinate resource in the query's filters.

D. Modules. Major elements of the query, such as a basic graph pattern (BGP) or filter statement, appear as *modules* in the JSON output. Each module contains:

(1) **SPARQL Statement.** This section displays the relevant snippet from the original SPARQL code (for example, a property path or a filter clause). It helps users cross-check correctness by comparing the code directly with the accompanying explanation.
(2) **Explanation.** This portion articulates the logic behind the snippet, referencing key variables or entity identifiers as needed. Under the system's requirements, variable names are shown in the format (?varName) to maintain clarity, and each IRI is combined with a human-friendly label (e.g., <Q123> [Berlin]) so that readers can match original identifiers to more familiar names or concepts.

Example NLE in JSON Format with Highlighting

```
{
    "Overall NL explanation":"This SPARQL query retrieves television series that feature Rowan Atkinson as a cast member
          , have exactly four seasons, and started in the year 1983. It filters the results to ensure that only series
          meeting these criteria are returned.",
    "Query Type":{
        "SPARQL":"SELECT ?tvShow",
        "Explanation":"This is a SELECT query that retrieves data from the dataset. It returns television series that
              match the specified criteria."
    },
    "Variables":{
        "Description":"The query retrieves the following variable:",
        "Variables":[
            {
                "Variable":"?tvShow",
                "SPARQL":"SELECT ?tvShow",
                "Explanation":"The variable (?tvShow) holds the television series that meet the specified criteria,
                      including having Rowan Atkinson as a cast member, having four seasons, and starting in 1983."
            }
        ]
    },
    "Patterns":{
        "Description":"The query includes the following patterns and operations:",
        "Modules": [
            {
                "Module":"bgp",
                "SPARQL":"?tvShow wdt:P31 wd:Q5398426; wdt:P161 wd:Q23760; wdt:P2437 ?seasons; wdt:P580 ?startDate",
                "Explanation":"Identify television series (?tvShow) that are instances of [television series] <wd:
                      Q5398426>, have [Rowan Atkinson] <wd:Q23760> as a [cast member] <wdt:P161>, have a [number of
                      seasons] <wdt:P2437> equal to ?seasons, and have a [start time] <wdt:P580> equal to ?startDate."
            },
            {
                "Module":"filter",
                "SPARQL":"FILTER(?seasons = 4)",
                "Explanation":"Apply a filter to include only those television series where the number of seasons (?
                      seasons) is equal to 4."
            },
            {
                "Module":"filter",
                "SPARQL":"FILTER(YEAR(?startDate) = 1983)",
                "Explanation":"Apply a filter to include only those television series that started in the year 1983. The
                      YEAR function extracts the year from the ?startDate variable."
            }
        ]
    },
    "Prefixes":{
        "Description":"The query uses the following prefixes:",
        "Prefixes":[
            {
                "Prefix":"wd",
                "URI": "http://www.wikidata.org/entity/",
                "Explanation":"Used to reference entities in Wikidata."
            },
            {
                "Prefix":"wdt",
                "URI": "http://www.wikidata.org/prop/direct/",
                "Explanation":"Used to reference direct properties in Wikidata."
            }
        ]
    }
}
```

Example 4: The structured Natural Language Explanation (NLE) of the query in Example 1 based on the parsing result in Example 2 and rule-based NLE in Example 3. Semantic bracket are colored to help easier understanding: *[] (green)* for lists and labels in natural language, *<> (blue)* for URLs, and *() (purple)* for variables.

(3) **Subquery Pattern (if applicable).** For queries that contain nested SELECT blocks or property paths, the explanation is arranged in a smaller "mini" query layout. By repeating the same modular structure at each nesting level, the NLE preserves a clear hierarchy, ensuring that complex multi-level queries remain easy to navigate and understand.

E. Advanced Clauses. If the SPARQL query incorporates additional features like `GROUP BY`, `ORDER BY`, `LIMIT`, or `OFFSET`, these appear as separate modules. This organization helps users see how each advanced clause shapes the outcome, such as limiting the number of returned rows or grouping results for aggregation. Both the code snippet and a succinct explanation clarify the effect of these features on the overall query logic.

F. Prefixes. Finally, if any prefixes (e.g., `wd:`, `wdt:`) are used, the LLM can optionally list them, along with a brief note on their typical usage. For example, it might indicate that `wdt:` relates to direct properties in Wikidata. Such details mitigate confusion for users who may not be familiar with the namespace conventions involved.

Summary. Taken together, these layers provide a clear, modular explanation of each SPARQL snippet while also illuminating its overall purpose in retrieving data. The design can address special or domain-specific nuances—for example, embedding guidance on why a certain query might yield no results (e.g., a "manner of death" request for a living person)—in either the *Overall NL Explanation* or the relevant module's *Explanation* field. By anchoring such pointers to concrete parts of the query (like a `FILTER` or a property path), the system prevents them from becoming tangential commentary. Additionally, demanding explicit references to variables, IRIs, and advanced clauses ensures the explanation aligns rigorously with the rule-based stage, reducing the chance of missing or conflated details. As a result, the final output is both richly contextual and reliably coherent: machine-readable in its structured format and intuitive for users who need to follow or refine each logical step of the query.

1.3 Interactive Query Refinement Framework

Extending the foundation established by our NLE framework (Section 1.2.2), we employ an *iterative refinement* procedure, called InteracSPARQL, that keeps SPARQL queries aligned with the user's original question. While the system can readily incorporate direct user feedback, we also offer a *self-refinement* mode in which an LLM simulates user suggestions for automated evaluation. In this section, we outline the key steps of the refinement loop, explain how the NLE underpins each iteration, and highlight a tool-based entity/property lookup mechanism that reduces the domain knowledge burden for query authors. Our current prototype implementation features these tools for popular knowledge graphs like Wikidata and DBpedia, but the same methodology can be adapted to other semantic datasets with minimal modification.

1.3.1 Motivation for Tool-based Entity and Property Search. When writing or refining SPARQL queries, it is often necessary to reference exact entity and property URIs[1]. Users or LLMs may not recall these IRIs offhand, leading to guesswork or errors. To address this challenge, we incorporate dedicated search tools that the LLM can invoke on demand. These functions query the target knowledge graph's API or index to identify proper IRIs for entities (e.g., "Batman") or properties (e.g., "native language"). By delegating entity/property linking to a well-defined utility, the iterative refinement loop becomes more fluid and robust, relieving users and the LLM of low-level domain details.

1.3.2 Detailed Workflow. InteracSPARQL workflow proceeds through four primary stages, supplemented by a **tool invocation** sub-step calling the entity/property search tool whenever the query is missing or misusing an entity or property:

(1) **Explain and Validate.** The system executes the SPARQL query on the chosen knowledge graph and collects results. Concurrently, it creates or updates the NLE to reflect the query's logical structure, enumerating triple patterns, filters, and so on. So the output of this step will be the execution results and NLE of the given query.
(2) **Evaluate and Provide *Feedback*.** Should the query's output prove not to accurately reflect the user's intention (i.e. the natural language question given), a feedback mechanism (either a human user or an LLM) identifies possible reasons for the mismatch—for instance, a wrong property URI. This feedback delineates which segments of the query (variables, filters, patterns) demand revision. If the current outputs already meet the user's intention, a feedback will be also given to indicate no further steps are required.
(3) **Refine the Query.** Using the feedback, the system selectively updates the query. If the feedback indicates that a particular entity or property is missing or erroneous, the LLM may invoke a tool to perform an on-demand search and discover the correct IRI. The refined query modifies only the problematic references, preserving previously validated logic. This incremental approach lowers the risk of introducing new errors.
(4) **Repeat if Necessary.** The system executes the refined query anew, generating updated results and a refreshed NLE, then going back to step (2) above. This loop repeats until satisfactory outputs are obtained or the process reaches a designated iteration limit.

By interweaving targeted feedback, query execution, and incremental corrections (including entity/property lookups), InteracSPARQL gradually rectifies any discrepancy between the user's question and the evolving SPARQL query. Algorithm 1 illustrates how these stages coalesce into a cohesive whole. Although this structure naturally accommodates genuine user feedback at each iteration, we also demonstrate a *self-refinement* variant in which an LLM simulates user input, which will be introduced with details in Section 1.3.4.

[1] For example, <http://www.wikidata.org/entity/Q27722874> in WikiData, whose label is "Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases".

Algorithm 1: Interactive Query Refinement Algorithm

Input : U: Natural Language Question (i.e. user's intent)
Q: Initial SPARQL query
K: Target knowledge graph
N: Maximum refinement iterations

Output : Q^*: Refined SPARQL query aligned with user intent

1 $i \leftarrow 0$
2 **while** $i < N$ **do**
  // 1. Explain & Validate
3 $\mathcal{R} \leftarrow$ `execute`(Q, K)
4 NLE $\leftarrow$ `generateOrUpdateNLE`$(Q, \mathcal{R})$
5 **if** `isConsistent` $(\mathcal{R}, U)$ **then**
6 **break** // Stop if results align with user's question
  // 2. Evaluate & Provide Feedback
7 fb $\leftarrow$ `getFeedback`$(Q,$ NLE$, \mathcal{R})$
  // 3. Refine the Query
8 **if** *feedback indicates incorrect entity or property* **then**
9 `toolCall` (search function for entity/property)
  // The LLM retrieves the appropriate IRI
10 $Q \leftarrow$ `applyFeedback`$(Q,$ fb$,$ NLE$)$
11 $i \leftarrow i + 1$
12 **return** $Q^* \leftarrow Q$

1.3.3 Role and Significance of the NLE. Although the NLE is generated or updated in Step 1 (line 3-6) of Algorithm 1, it informs each iteration:

- **Clarity for Users.** By expressing the query's triples, filters, or other clauses in a human-friendly style, the NLE allows both non-specialists and domain experts to pinpoint problematic regions needing attention.
- **Anchor for Feedback and Tool Calls.** The NLE offers a structured blueprint of the SPARQL statement, so the LLM (or user) can reference specific IRIs or variables before invoking the relevant lookup tool.
- **Semantic Continuity.** After every iteration, the NLE is updated to mirror the refined query, ensuring subsequent feedback remains accurate and consistent with the latest version.

Overall, the NLE bridges the gap between original SPARQL code and high-level user reasoning, ensuring coherent and iterative query refinement.

1.3.4 Self-Refinement Baseline. Under normal circumstances, Step 2 (line 7) of Algorithm 1 assumes that human users (or domain experts) would review the query outputs and provide feedback on whether additional filters, entity substitutions, or property adjustments are needed. However, when real-time user involvement is unavailable or impractical, we employ a *self-refinement* variant that demonstrates the workflow's viability under reproducible conditions. In this mode, the LLM assumes both roles, *feedback* and *refine*, by:

(1) In Step 2 (line 7), it is now the LLM that generates feedback (i.e. `getFeedback`) based on discrepancies between the NLE, the executed query's results, and the intended user question, rather than human users.
(2) Replacing or modifying specific query elements (in Step 3, line 8-9) according to the LLM's own self-issued feedback, all while preserving validated segments from earlier iterations.

By embedding these automated feedback cycles and tool calls into the established refinement loop, we confirm the framework's capacity to converge on correct SPARQL queries without relying on direct human input. Once user interaction becomes feasible, *e.g.*, when a domain expert is available to oversee the refinement process, the system seamlessly transitions into a fully interactive paradigm, with the user or LLM calling upon the same entity/property search tools as needed. This design choice not only streamlines evaluation in controlled environments but also paves the way for a robust and flexible human-in-the-loop approach.

2 HUMAN EVALUATION PROTOCOL

Task. Compare two natural-language explanations (NLEs) for the same SPARQL query and decide which is better.

Materials Provided.

- The original natural-language question.
- The corresponding SPARQL query.
- Two candidate explanations, labeled "Version A" and "Version B."

Procedure.

(1) Read the question and examine the SPARQL query.
(2) Read both explanations in full.
(3) For *each* explanation, give a score from 1 (worst) to 5 (best) on all four dimensions below.
(4) Finally, choose the overall winner (Version A, Version B, or Tie).

Rating Dimensions (1–5).
Clarity:

- 1 – Confusing or disorganized; hard to follow.
- 2 – Frequently unclear; awkward wording.
- 3 – Moderately clear; some parts disjointed.
- 4 – Mostly clear; minor wording issues.
- 5 – Crystal-clear; natural, easy-to-follow flow.

Completeness:

- 1 – Omits most key components (variables, filters, clauses).
- 2 – Several important elements missing.
- 3 – Covers most parts but glosses over some details.
- 4 – Includes all critical parts; minor gaps.

- 5 – Every significant component is described in detail.

Aesthetics:

- 1 – Messy or distracting presentation.
- 2 – Awkward structure or jarring tone.
- 3 – Acceptable but not engaging.
- 4 – Well-organized and pleasant to read.
- 5 – Exemplary wording and formatting; highly readable.

Utility:

- 1 – Provides no real help in understanding or modifying the query.
- 2 – Minimal guidance; still difficult to use.
- 3 – Somewhat helpful; basic understanding but low confidence.
- 4 – Generally useful; reader could debug or adapt with little effort.
- 5 – Fully enables confident understanding, debugging, and modification.

Overall Preference. Choose one:

- Version A is better
- Version B is better
- Tie

Select "Tie" only if the two explanations are genuinely indistinguishable.