INTERACSPARQL: An Interactive System for SPARQL Query Refinement Using Natural Language Explanations

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

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

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

Abstract

Current approaches for Natural Language to SPARQL (NL2SPARQL) generation primarily rely on one-turn, training-intensive models. While effective in specific settings, these models often lack generalizability and fail to provide transparency or mechanisms for error recovery in realistic scenarios. Additionally, prior interactive works are largely outdated and incompatible with modern large language model (LLM) workflows. In this paper, we introduce INTERACSPARQL, a training-free interactive refinement pipeline that acts as a plug-and-play enhancement for existing SPARQL generation systems. Our approach integrates a set of efficient entity and property lookup tools within a self-correction loop, guided by a novel hybrid Natural Language Explanation (NLE) module. This module combines rule-based Abstract Syntax Tree (AST) parsing with LLM semantic enrichment to produce explanations that are both structurally accurate and linguistically fluent. We evaluate INTERACSPARQL on standard benchmarks (QALD-9 and QALD-10), showing that our tool-augmented self-refinement significantly boosts the accuracy of base models without fine-tuning. Furthermore, human evaluation confirms that our structured explanations substantially improve user understanding and ability to correct queries compared to unstructured baselines. The code and datasets are available at https://github.com/Edward-JianQAQ/InteracSPARQL.

1 Introduction

Querying RDF (Resource Description Framework) data remains a formidable challenge due to the complexity of the SPARQL syntax and the strict requirement to identify resources using precise *Internationalized Resource Identifiers (IRIs)* (Li et al., 2023; Amsterdamer and Callen, 2021; Arenas and Ugarte, 2017; Diaz et al., 2016; Mohamed et al., 2022). While NL2SPARQL systems offer a promising bridge for non-expert users, existing approaches face significant reliability hurdles. As discussed in Section 5, current State-of-the-Art (SoTA) methods typically rely on *one-turn generation* (Omar et al., 2023; Jiang et al., 2023; Xie et al., 2022; Yu et al., 2023), where a (often fine-tuned) model attempts to produce a perfect query in a single pass. However, these models are inherently static: if the generated query contains a hallucinated IRI or a subtle logical flaw, which are common issues in low-resource languages like SPARQL, the user is left with a failed execution and no transparency into *why* the query failed (Li et al., 2023; Diallo et al., 2024).

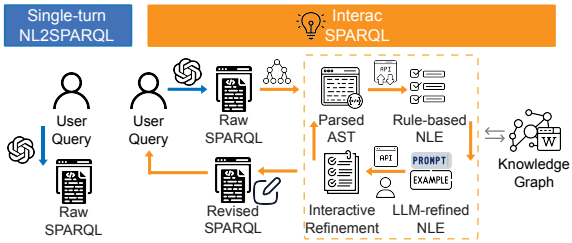


Figure 1: Overview of the INTERACSPARQL pipeline. Functioning as an add-on refinement layer, it transforms potentially flawed SPARQL queries from any source into verifiable code via AST parsing, Hybrid NLE generation, and a tool-augmented interactive feedback loop.

Furthermore, while end-to-end Knowledge Base Question Answering (KBQA) systems exist, they often function as "black boxes," creating a trust gap in enterprise applications where verifiable query logic is required, which is elaborated in Section 5. To bridge this gap, an effective interface must not only generate code but also explain it and allow for correction. As detailed in Section 5, prior works on *interactive SPARQL refinement* (Amsterdamer and Callen, 2021; Abramovitz et al., 2018; Ochieng, 2020; Letelier et al., 2012) have attempted to solve this, but they are largely outdated, closed-source, and lack the integration with modern LLMs required to handle complex natural language intent.

Findings of the Association for Computational Linguistics: ACL 2026, pages 29599–29624

In this paper, we introduce **INTERACSPARQL**, a training-free, interactive framework designed to act as a robust "add-on" layer for any SPARQL generation system. Unlike previous approaches that assume specific "golden entities" are known a priori, INTERACSPARQL addresses the realistic, zero-shot scenario where neither the user nor the model has perfect knowledge of the underlying graph. Our system provides a set of well-designed, efficient tools that allow base models to verify their own assumptions and self-correct hallucinations.

A core scientific challenge in this domain is providing feedback that is both structurally accurate and understandable to humans. Pure rule-based explanations are often too rigid to be helpful, while LLM-generated explanations are prone to hallucinations. We address this via a novel **Hybrid NLE** module. This module first uses deterministic AST parsing to capture the exact logical skeleton of the query, which is then rewritten by an LLM. This ensures that the explanation serves as a faithful "Chain of Thought", allowing users (or the model itself) to pinpoint errors in logic or entity linking.

The INTERACSPARQL pipeline (Fig. 1) supports two modes of operation: (1) *Interactive Refinement*, where users correct queries based on our transparent explanations; and (2) *Self-Refinement*, where the system utilizes a suite of external lookup tools to resolve incorrect IRIs and logic errors autonomously. This design positions INTERACSPARQL not as a competitor to existing one-turn models (like SPINACH (Liu et al., 2024)), but as a necessary infrastructure layer that enhances their generalizability and accuracy without requiring expensive fine-tuning. In general, our contributions are as follows:

1. We propose a training-free, tool-augmented refinement framework that can be integrated with any NL2SPARQL model. It moves beyond simple generation to provide a robust infrastructure for error detection and correction in zero-shot scenarios.

2. We introduce a hybrid NLE methodology that combines rule-based AST analysis with LLM semantic enrichment. This approach solves the trade-off between structural fidelity and linguistic fluency, providing a transparent basis for refinement.

3. We design a dynamic entity and property linking toolset within the refinement loop. Our ablation studies demonstrate that this explicit tool use is critical for resolving hallucinations in large-scale knowledge graphs (KGs), rather than relying on implicit LLM knowledge.

4. We conduct comprehensive evaluations on `QALD` benchmarks and a rigorous human user study. Results show that INTERACSPARQL significantly improves the F1 scores of base models and that our structured explanations are preferred by users for their clarity and utility in debugging.

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

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

2 Background of RDF and SPARQL

The Resource Description Framework (RDF) is the standard model for data interchange on the Semantic Web (Ali et al., 2022; Wylot et al., 2018), where most of KGs, including those mentioned in this work, are built under this model. RDF represents data as triples (s, p, o) consisting of a subject, predicate, and object, where resources are uniquely identified by *Internationalized Resource Identifiers (IRIs)*. To query this data, the W3C standardized SPARQL (W3C, 2006; Group, 2013). A typical SPARQL query consists of a `SELECT` clause (defining output variables) and a `WHERE` clause containing *Basic Graph Patterns (BGPs)* and operators like `FILTER` or `OPTIONAL`, like the one in Example 1. Formal definitions of RDF graphs and SPARQL syntax are provided in Appendix B.

3 The INTERACSPARQL Framework

INTERACSPARQL is designed as a framework to provide a transparent, interactive refinement layer for SPARQL generation. Unlike end-to-end generation methods, it does not aim to replace the initial generation step; instead, it functions as an environment that ingests a raw query (from any LLM or directly from a human user), explicates its logic to the user, and facilitates error correction through interactive feedback and tool use.

As depicted in Figure 2, the pipeline consists of four phases: ① **AST Parsing:** The raw SPARQL is parsed into a JSON-based Abstract Syntax Tree (AST), creating a machine-readable blueprint; ② **Rule-Based NLE with Grounding:** We apply deterministic rules to the AST to produce a structural explanation skeleton, replacing opaque IRIs

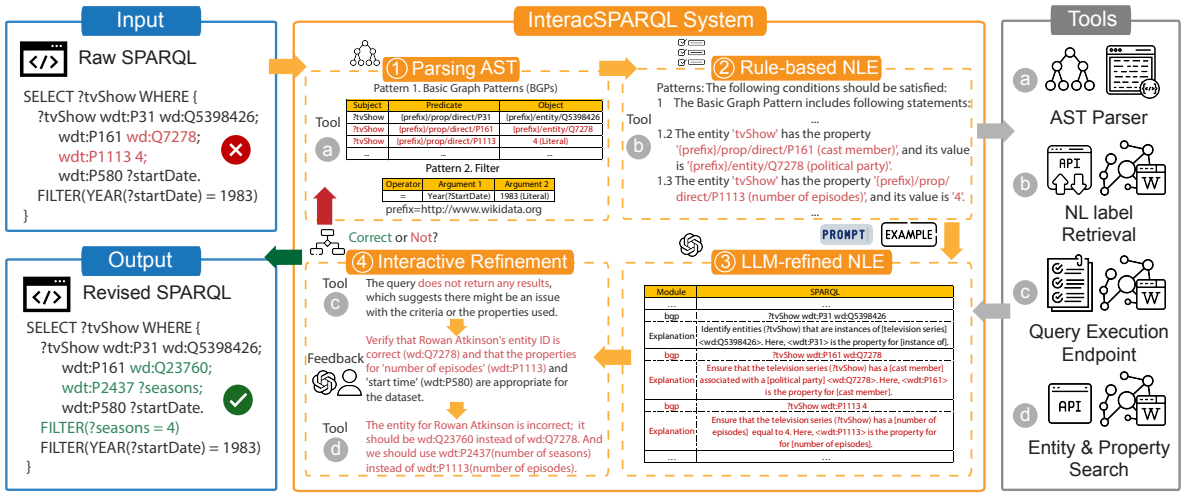


Figure 2: The INTERACSPARQL pipeline. The input is a potentially flawed SPARQL query generated by any base model. Our system parses this query into an AST (①), generates a Hybrid Natural Language Explanation (②-③), and executes a Tool-Augmented Refinement Loop (④) to resolve hallucinations and logic errors. An actual data example from `QALD-10` is applied to help the illustration, the corresponding natural language question is *"What is the TV-show that starred Rowan Atkinson, had 4 seasons and started in 1983?"*.

with human-readable labels via knowledge graph lookups; ③ **LLM-refined NLE with Semantic Enrichment via Constrained Generation:** An LLM, guided by few-shot examples, refines this skeleton into a fluent, structured JSON explanation; ④ **Interactive Refinement:** The system enters a feedback loop where problematic clauses are identified (by a human or the model) and corrected using external tools.

3.1 AST Parsing of SPARQL

The framework begins by transforming the raw SPARQL query into a machine-readable JSON Abstract Syntax Tree (AST), i.e. step ①. This parsing stage is deterministic, identifying core components such as `SELECT` variables, `Basic Graph Patterns (BGPs)`, and `FILTER` expressions. By creating a structured hierarchy, we isolate logical units for targeted refinement. The full schema of this AST is detailed in Appendix C.1.

3.2 Hybrid NLE

A major challenge in refining SPARQL is the trade-off between fidelity and fluency. Rule-based explanations are accurate but rigid, while LLM-generated explanations are fluent but prone to hallucinating logic that does not exist in the code. We resolve this via a two-stage hybrid approach.

3.2.1 Rule-Based NLE with Grounding

In Step ②, we apply a deterministic traversal to the AST to extract a skeletal explanation. While the parsing (Section 3.1) establishes the structure, this stage focuses on *semantic grounding*. Raw IRIs (e.g., `wd:Q5398426`) are opaque to most users and a frequent source of LLM hallucinations. Our system performs on-demand lookups against the Knowledge Graph schema to resolve these IRIs into human-readable labels (e.g., *"television series"*). The output is a set of factually verifiable statements (e.g., *"The entity ?tvShow has property [instance of]..."*). By mechanically anchoring the explanation to the code, we prevent the model from inventing relationships that do not exist in the graph. The extracted output of Example 1 is Example 2 and details are elaborated in Appendix C.2.1.

3.2.2 LLM-refined NLE with Semantic Enrichment via Constrained Generation

In the second stage (Step ③), we leverage an LLM to transform the rigid AST outputs into a fluent and semantically rich narrative. However, to prevent the model from deviating from the code's logic, we constrain the generation process using a structured tree-shaped schema. The LLM receives the rule-based statements and a set of few-shot examples, and is instructed to produce an NLE containing specific modules. The most important ones are: (1) **Overall Intent:** A high-level summary of the query's goal; (2) **Variable Roles:** Explanations of what each variable represents in the retrieval logic; (3) **Clause Modules:** Step-by-step breakdowns of BGPs and Filters, where the LLM synthesizes the label-enriched triplets into natural sentences.

Algorithm 1: Tool-Augmented Refinement Loop

Input : Q_{init}: Initial Query, K: Knowledge Graph
Output : Q_{final}: Refined Query

1 $Q \leftarrow Q_{init}$
2 **while** $i < MaxIterations$ **do**
    // 1. Validate via Execution & NLE
3 $Results \leftarrow \texttt{Execute}(Q, K)$
4 $NLE \leftarrow \texttt{GenerateHybridNLE}(Q)$
5 **if** *Results align with User Intent* **then**
6 **return** Q
    // 2. Diagnosis via Feedback
7 $Feedback \leftarrow \texttt{GetCritique}(Q, NLE, Results)$
    // 3. Correction via Tool Use
8 **if** *Feedback detects IRI/Schema Error* **then**
9 $CorrectedIRI \leftarrow \texttt{ToolCall}(\text{EntitySearch}, Feedback)$
10 $Q \leftarrow \texttt{UpdateQuery}(Q, CorrectedIRI)$
11 **else**
12 $Q \leftarrow \texttt{UpdateLogic}(Q, Feedback)$
13 $i \leftarrow i + 1$

We have the output of Example 2 after this process as Example 3. This structured output serves as a "Chain of Thought" for the user, allowing them to verify exactly how the model interprets complex constructs like property paths or negations before execution. Details are in Appendix C.2.2.

3.3 Tool-Augmented Interactive Refinement

The core of INTERACSPARQL is its ability to recover from errors, specifically the hallucinated IRIs and property mismatches common in zero-shot generation. We implement this via an agentic loop (Step ④) that integrates *execution feedback* with *external tool use*.

We formalize this workflow in Algorithm 1. The process iterates until the query execution aligns with the user's intent or a maximum step limit is reached. We have included descriptions with examples of the whole workflow to show every minute detail in Appendix C.3. Concisely, the workflow consists of three critical phases:

1. Explain and Validate (Lines 2-6): The system executes the query and generates the NLE. By presenting both the *results* (often empty in failed queries) and the *explanation*, the user (or the model) can diagnose whether the failure is due to logic or data coverage.

2. Feedback Generation (Line 7): Feedback is generated based on the discrepancy between the NLE and the user's intent. In *Human-in-the-Loop* mode, the user points out the error (e.g., "The property for 'cast member' seems wrong"). In *Self-Refinement* mode, the LLM acts as the critic, comparing the execution result (e.g., Empty Set) against the NLE to hypothesize potential errors.

3. Tool-Assisted Correction (Lines 8-12): This is the distinct feature of our framework. Standard generative models often attempt to "guess" a better IRI when correcting errors, leading to repeated hallucinations. INTERACSPARQL instead invokes a deterministic **Entity/Property Search Tool**. If the feedback flags an entity error, the system calls the KG API to retrieve the correct IRI (e.g., swapping a hallucinated ID for `wd:Q23760`) and surgically patches the AST.

This methodology transforms SPARQL generation from a static prediction task into a dynamic, evidence-based reasoning process, significantly increasing robustness in zero-shot domains.

4 Experimental Evaluation

We design our experiments to assess INTERACSPARQL as an enhancement layer for existing models. Specifically, we measure: (1) the **quality improvement** in SPARQL generation when base LLMs are augmented with our refinement framework; and (2) the **utility** of our Hybrid NLEs in facilitating human understanding.

4.1 Experimental Setup

Implementation. We implemented the INTERACSPARQL framework in Python (approx. 6k lines of code), adaptable to any environment supporting Python 3.8+. The system is lightweight and model-agnostic; it requires only API access to the base LLM (or a local GPU for open-source models) and network access to public SPARQL endpoints. For our experiments, we utilize the official public endpoints for Wikidata and DBpedia to execute queries. Full implementation details, including computational costs and hardware specifications, are provided in Appendix D.1.

Datasets. We utilize the Question Answering over Linked Data (`QALD`) benchmarks, specifically `QALD-10` (Wikidata), `QALD-9-Wikidata` (Wikidata), and `QALD-9-DBpedia` (DBpedia) (Usbeck et al., 2023). We selected these human-annotated datasets because they reflect realistic usage scenarios with complex properties, unlike synthetic datasets. We focus on `SELECT` and `ASK` queries, which constitute the vast majority of real-world use cases. More details about dataset information

Table 1: Experimental results on QALD-10, QALD-9-Wikidata, and QALD-9-DBpedia, reported as macro-averaged Precision/Recall/F1. For each setting and dataset, the top three F1 scores are highlighted: **1st**, **2nd**, and **3rd**.

Setting	Model	QALD-10			QALD-9-Wikidata			QALD-9-DBpedia		
		Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
Raw	GPT-5	0.404	0.509	**0.433**	0.415	0.535	**0.445**	0.352	0.600	0.418
	GPT-4o	0.135	0.143	**0.136**	0.280	0.275	0.264	0.473	0.480	**0.467**
	GPT-4o-mini	0.035	0.034	0.033	0.278	0.279	**0.265**	0.435	0.452	**0.430**
	Claude-3.5-Sonnet	0.172	0.176	**0.172**	0.356	0.376	**0.350**	0.524	0.547	**0.523**
	Qwen-2.5-32B	0.019	0.021	0.020	0.023	0.036	0.026	0.314	0.316	0.310
	Qwen-2.5-14B	0.057	0.057	0.057	0.009	0.015	0.010	0.315	0.325	0.312
	Qwen-2.5-7B	0.021	0.023	0.021	0.000	0.003	0.000	0.148	0.151	0.147
Upper Bound	GPT-5	0.926	0.932	0.928	0.877	0.887	**0.874**	0.886	0.879	0.880
	GPT-4o	0.930	0.930	0.930	0.833	0.831	0.837	0.931	0.931	0.931
	GPT-4o-mini	0.948	0.947	**0.947**	0.838	0.835	0.836	0.938	0.937	**0.937**
	Claude-3.5-Sonnet	0.873	0.873	0.873	0.671	0.673	0.671	0.964	0.964	**0.964**
	Qwen-2.5-32B	0.957	0.961	**0.959**	0.955	0.960	**0.957**	0.974	0.974	**0.974**
	Qwen-2.5-14B	0.953	0.951	**0.951**	0.935	0.936	**0.935**	0.914	0.913	0.913
	Qwen-2.5-7B	0.692	0.697	0.694	0.786	0.790	0.785	0.677	0.679	0.677
Self-refine	GPT-5	0.411	0.567	**0.455**	0.482	0.664	0.528	0.387	0.629	0.448
	GPT-4o	0.389	0.408	**0.393**	0.581	0.585	**0.561**	0.549	0.551	**0.532**
	GPT-4o-mini	0.326	0.345	0.328	0.544	0.546	**0.553**	0.522	0.521	0.512
	Claude-3.5-Sonnet	0.377	0.408	**0.383**	0.567	0.588	**0.560**	0.574	0.593	**0.567**
	Qwen-2.5-32B	0.314	0.361	0.325	0.382	0.497	0.411	0.532	0.530	**0.523**
	Qwen-2.5-14B	0.259	0.291	0.266	0.326	0.361	0.328	0.443	0.452	0.439
	Qwen-2.5-7B	0.308	0.331	0.312	0.355	0.377	0.350	0.307	0.315	0.307

and the scope of queries included are included in Appendix D.2.

Base Models. To demonstrate the generalizability of our framework, we evaluate INTERACSPARQL across seven distinct LLMs, ranging from proprietary SoTA models to smaller open-source models: `GPT-5`, `GPT-4o`, `GPT-4o-mini`, `Claude-3.5-Sonnet`, `Qwen-2.5-32B`, `Qwen-2.5-14B`, and `Qwen-2.5-7B`. Specific configurations are detailed in Appendix D.3.

Evaluation Metrics. We first execute each query on the target knowledge graph (Wikidata or DBpedia) with the online API service (as mentioned in Appendix D.3) and then compare its returned result set with that of the ground-truth query. For `ASK` queries, we directly check whether both queries produce the same boolean value. For `SELECT` queries, we gather and compare the sets of returned tuples. We then compute precision, recall, and F1 score to indicate how closely the results align. These metrics are collected across all queries tested, and we also aggregate them to identify overall performance and potential error patterns, e.g., average precision/recall and macro-averaged F1 (F1 scores reported in this work are macro-averaged). Given the cost of the experiment and the method's stability, we run each setting three times and report the average. The deviation of three runs is all less than 1% in the study.

4.2 Accuracy of INTERACSPARQL

We evaluate INTERACSPARQL by measuring how accurately it helps align SPARQL queries with user questions. Two distinct experiments are conducted to evaluate INTERACSPARQL against certain baselines. In the first, we examine an *upper-bound scenario* where ground-truth NLEs are treated as fully correct and used to guide a single-pass SPARQL generation. In the second, we test a more *realistic self-refinement scenario*, in which the system iteratively adjusts queries over several rounds, drawing on automatically generated NLEs. In both cases, the generated queries are executed and compared to the results of known ground-truth queries.

4.2.1 Raw and Upper-Bound Generation

We compare INTERACSPARQL against two contrasting scenarios. The results are in Table 1.

Raw generation fails, primarily due to ungrounded opaque identifiers. In this setup, the LLM directly produces a SPARQL query from a user's natural-language question (NLQ), without any intermediate explanation or refinement. This represents our *baseline*, testing how well the LLM can handle SPARQL generation in a single pass when guided only by a short prompt containing

the NLQ and the knowledge graph it is based on. As Table 1 demonstrates, raw generation typically yields very low F1 scores. To better understand the reason for this poor performance, we take the 123 examples where our self-refinement loop ultimately improved the raw query and run a fine-grained keyword search over every feedback message flagged as an actual error (using `GPT-4o` on `QALD-10`). We count all issues per entry (so one entry might contribute multiple error counts, at most one count per error type). The most frequent mistakes are incorrect entity IRIs (71) and incorrect property IRIs (55). Mistakes in all patterns other than BGPs appear 8 times (like `Filter` and `Bind`). We also observe 10 alignment-to-question errors and 8 execution result errors, indicating that the query does not generally match the user's question. In short, the raw generation does not fail for lack of syntactic or logical SPARQL competence, but because the model cannot reliably select the right identifiers or fully understand the user's question in a single pass.

Upper-bound results confirm that accurate NLEs enable near-flawless code generation. In the *upper-bound* scenario (middle of Table 1), the LLM receives a ground-truth NLE and generates a query in one pass. The resulting queries achieve near-perfect F1 (e.g., 0.977 for `GPT-4o` on `QALD-10`), confirming that accurate explanations are sufficient for correct query generation. Although impractical for routine deployment, this establishes a ceiling and shows that INTERACSPARQL can serve as an effective "handle" for interactive refinement (Section 4.2.2).

4.2.2 Practical iterative Self-Refinement

We implement the self-refinement baseline that autonomously iterates up to five times to refine both the SPARQL query and its NLE based on detected errors. The bottom part of Table 1 shows that our **self-refinement** pipeline achieves substantially higher F1 scores than the baseline raw generation. Notably, no external agent (e.g., a human reviewer) provides feedback in this setup; the model itself identifies potential issues (such as incorrect IRIs or missing filters), invokes tool-based lookups when necessary, and revises the query accordingly. These results demonstrate that even a fully automated loop can meaningfully converge toward the user's intended query. Furthermore, this setup also constitutes a *good starting point for a human-in-the-loop* approach: once the model refines the query to a satisfactory baseline, a human expert (if available) can inspect and fine-tune it further, minimizing the manual workload required.

4.2.3 Comparison with Prior NL2SPARQL Systems

Table 4 compares INTERACSPARQL against prior NL2SPARQL systems using published numbers from the KGQA leaderboard (Perevalov et al., 2022) and QALD-10 proceedings (Usbeck et al., 2023). Despite being a *training-free enhancement layer* rather than a one-turn generator, INTERACSPARQL is competitive: `GPT-5` (F1=0.455) places between Shivashankar et al. (Shivashankar et al., 2022) (0.491) and Baramiia et al. (Baramiia et al., 2022) (0.428) on `QALD-10`. Meanwhile on `QALD-9-DBpedia`, `Claude-3.5-Sonnet` (F1=0.567) exceeds most trained systems. Higher-scoring systems are fine-tuned or use KG-specific pipelines that do not generalize. The output of any prior system can serve as input to INTERACSPARQL for further refinement.

4.2.4 Error Analysis of Unresolved Failures

We analyze queries that did not reach F1=1.0 after refinement (`GPT-4o`, `QALD-10`, $N=5$). Of 280 completed queries, 138 reached F1=1.0 (112 improved, 26 already correct). We classified the remaining 142 failures by comparing refined queries against ground truth along three axes: entity IRIs, property IRIs, and structural features (Table 5).

IRI errors dominate (57.7%), with property errors more prevalent than entity errors (72.5% vs. 64.1% across all 142 failures), confirming that property disambiguation is harder as it requires understanding KG schema conventions. Degradation is rare, as only 5/280 queries (1.8%) had F1 decrease. Examples are in Appendix G.

4.3 Ablation Studies

4.3.1 Impact of NLE Design

To rigorously quantify the contribution of each component in our Hybrid NLE framework, we evaluate four distinct configurations, isolating the effects of structural grounding and few-shot guidance:**(a) Original Design (OD):** The complete pipeline, integrating deterministic rule-based AST extraction with LLM-driven semantic enrichment, guided by structured few-shot examples. **(b) Bare Query Baseline (BQB):** A direct generation baseline where the LLM is prompted to explain the

Table 2: Ablation study on design of NLE on QALD-10 and QALD-9-DBpedia datasets with both closed- and open-sourced LLMs.

Model & Dataset	OD			BQB			NFS			BQFS		
	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
GPT-4o + QALD-10	0.930	0.930	0.930	0.766	0.768	0.767	0.819	0.817	0.818	0.376	0.376	0.376
Qwen-2.5-32B + QALD-10	0.957	0.961	0.959	0.693	0.697	0.693	0.881	0.889	0.883	0.404	0.404	0.403
GPT-4o + QALD-9-DBpedia	0.931	0.931	0.931	0.825	0.841	0.829	0.926	0.928	0.927	0.821	0.835	0.826
Qwen-2.5-32B + QALD-9-DBpedia	0.974	0.974	0.974	0.860	0.875	0.864	0.980	0.980	0.980	0.970	0.970	0.970

Table 3: Ablation study on design of NLE on self-refinement performance on QALD-10 and QALD-9-DBpedia datasets with both closed- and open-sourced LLMs.

Model & Dataset	OD			BQB			NFS			BQFS		
	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
GPT-4o + QALD-10	0.389	0.408	0.393	0.329	0.344	0.332	0.374	0.391	0.378	0.291	0.308	0.294
Qwen-2.5-32B + QALD-10	0.314	0.361	0.325	0.290	0.337	0.302	0.309	0.363	0.322	0.260	0.297	0.268
GPT-4o + QALD-9-DBpedia	0.549	0.551	0.532	0.545	0.533	0.521	0.543	0.538	0.525	0.521	0.515	0.503
Qwen-2.5-32B + QALD-9-DBpedia	0.532	0.530	0.523	0.498	0.501	0.491	0.488	0.486	0.480	0.515	0.514	0.508

raw SPARQL code without any structural scaffolding or few-shot guidance. This serves as a control to measure the model's intrinsic zero-shot capability. (c) **Bare Query with Few-Shots (BQFS):** This setting provides the LLM with structured input/output examples but omits the rule-based AST skeleton. By comparing BQFS with OD, we isolate the specific value of the *intermediate structural representation* in anchoring the generation. **(d) No Few-Shots (NFS):** This configuration retains the full rule-based AST skeleton but removes the few-shot examples. This allows us to assess whether the *schema-enforced prompt* alone is sufficient for high-quality generation.

We assess these configurations across two distinct tasks: upper-bound generation (Table 2) and practical iterative self-refinement (Table 3). The extended discussion of the results is provided in Appendix E.1. In the **upper-bound generation** task, the OD configuration consistently yields the highest F1 scores across datasets (e.g., 0.977 for GPT-4o on QALD-10). Notably, the performance gap between OD and BQFS highlights that few-shot prompting alone is insufficient; the deterministic AST skeleton provides critical "grounding" that prevents the LLM from hallucinating logic not present in the code. While the NFS setting performs competitively in terms of information completeness, it lacks the stylistic consistency required for optimal user comprehension. These outcomes highlight that our structured explanations robustly preserve crucial query semantics and structural details, firmly aligning with insights from the human evaluation discussed in Section 4.4.2.

In the **iterative self-refinement** scenario, the OD approach maintains its advantage. For instance, GPT-4o combined with OD significantly outperforms all baselines on QALD-10. This indicates that our detailed, structured explanations act as effective "scaffolding" for the agentic loop, allowing the model to diagnose errors and formulate corrections more accurately than with unstructured or purely rule-based feedback. We observe that simpler baselines (BQB, BQFS) are only competitive on DBpedia, where semantic-rich identifiers (e.g., dbr:Barack_Obama) naturally reduce the need for structural grounding compared to Wikidata IRIs.

4.3.2 Impact of Designs on Self-Refinement

We further evaluate the necessity of external tools and NLEs, and the impact of refinement steps on self-refinement of INTERACSPARQL.

Synergy of Tools and NLE. As shown in Table 8, our ablation study reveals that both external tools and structured explanations are essential for effective refinement. Removing the *Entity/Property Search Tool* causes a drastic performance drop (e.g., GPT-4o F1 on QALD-10 falls from 0.393 to 0.112), confirming that LLMs alone cannot reliably correct opaque IRIs without external grounding. However, removing the NLE also leads to a significant decline (e.g., F1 drops to 0.351). This indicates that while tools provide the necessary data access, the structured NLE provides the critical "scaffolding" that enables the LLM to diagnose when and why to invoke those tools. The highest performance is achieved only when both components work in synergy (see Appendix E.2

Table 4: Comparison with prior NL2SPARQL systems (macro-averaged F1). "Spec." = requires training/specialization on the target KG.

QALD-10 (Wikidata)

System	F1	Spec.
SPINACH (Liu et al., 2024)	0.695	Yes
SPARQL-QA (Borroto et al., 2022)	0.595	Yes
QAnswer (Diefenbach et al., 2019)	0.578	Yes
Shivashankar et al. (Shivashankar et al., 2022)	0.491	Yes
INTERACSPARQL (GPT-5)	**0.455**	No
Baramiia et al. (Baramiia et al., 2022)	0.428	Yes
INTERACSPARQL (GPT-4o)	**0.393**	No

QALD-9-DBpedia

System	F1	Spec.
SGPT (Rony et al., 2022)	0.678	Yes
INTERACSPARQL (Claude-3.5-Sonnet)	**0.567**	No
Purkayastha et al. (Purkayastha et al., 2022)	0.553	Yes
LingTeQA (To and Reformat, 2020)	0.535	Yes
INTERACSPARQL (GPT-4o)	**0.532**	No
qaSQP (Zheng and Zhang, 2019)	0.463	Yes
INTERACSPARQL (GPT-5)	**0.448**	No
gAnswer (Zou et al., 2014)	0.296	Yes
QAnswer (Diefenbach et al., 2019)	0.197	Yes

Table 5: Root cause classification of 142 unresolved queries (GPT-4o, QALD-10, $N = 5$).

Root Cause	Count	%
IRI errors only (correct structure)	82	57.7
Structural mismatch + IRI errors	35	24.6
Triple-pattern count + IRI errors	11	7.7
Structural mismatch only	9	6.3
Subtle logic errors	5	3.5

for detailed analysis).

Refinement Iterations. Regarding the refinement loop structure, Table 9 demonstrates that setting the maximum iterations to $N = 5$ is optimal. While a single iteration ($N = 1$) yields modest gains, extending the loop to $N = 5$ allows the system to converge on correct entities, raising the F1 score for GPT-4o from 0.307 to 0.393. Increasing N further to 10 yields diminishing returns that do not justify the additional computational cost. The extended discussion is provided in Appendix E.3.

4.4 Human Evaluation of NLE Quality

Complementing the benchmark evaluations, we conduct a comprehensive human evaluation to assess the perceived quality of our NLEs directly. This evaluation involves a head-to-head comparative study among the four ablation configurations detailed previously in Section 4.3.1: **OD**, **BQB**, **BQFS**, and **NFS**.

4.4.1 Evaluation Setup

The study focuses on essential dimensions, including dataset complexity and variations in LLMs.

Datasets and Knowledge Graphs & LLM Variants. We evaluate explanations generated for two datasets of differing complexity: QALD-10 (higher complexity) and QALD-9-DBpedia (lower complexity). This allows us to examine how the complexity of datasets and the underlying knowledge graphs (DBpedia versus Wikidata) influence explanation quality. We evaluate with both closed-source models (GPT-4o) and open-source models (Qwen-2.5-32B) to confirm the general applicability and robustness of our NLE approach across different LLM architectures.

Experimental Design. Each participant sees a series of head-to-head comparisons. In each comparison, they are shown two NLEs side by side for the same SPARQL query: one produced by the Original Design (OD) and the other by exactly one of the three ablated configurations (BQB, BQFS, or NFS). The participants are not told which one is from which system. For every pair, participants provide four independent dimension scores on a five-point Likert scale (1 = Very Poor, 5 = Very Good) and then indicate an overall preference (OD wins, OD loses, or Tie). The four dimensions are: **(a) Aesthetics:** Readability, highlighting, formatting, and overall presentation structure; **(b) Clarity:** Use of clear, everyday language, logical flow, and ease of understanding the query's intent; **(c) Completeness:** Coverage of all critical SPARQL components, i.e., variables, graph patterns, filters, subqueries, prefixes, and modifiers. **(d) Usefulness:** Whether the explanation helps a reader unfamiliar with the query to understand, debug, extend, or modify it accurately.

Although participants often rely on their dimension scores when choosing a winner, the overall preference is recorded separately to capture their holistic judgment. A detailed protocol for the human study, given to participants, together with their profiles, can be found in Appendix F.

4.4.2 Results and Analysis

The human evaluation results, illustrated in Table 6, highlight the consistent advantage of our

Table 6: **Aggregated Human Evaluation Results. OD** achieves the highest utility and clarity.

	Avg. Dimension Scores				vs. OD Preference (%)		
Design	**Aes.**	**Clar.**	**Comp.**	**Util.**	**Win**	**Tie**	**Lose**
BQB	3.15	3.49	3.75	3.37	77.1	12.0	10.8
NFS	3.36	3.75	3.96	3.66	73.3	13.6	13.1
BQFS	**4.39**	4.48	4.52	4.42	27.3	54.9	17.8
OD	4.37	**4.57**	**4.73**	**4.52**	—	—	—

OD across all assessed dimensions and comparison settings. A more fine-grained version with different LLMs is included in Table 7 and Figure 3 in Appendix A. Participants consistently rate OD highest for completeness and usefulness across both models and datasets. This clearly underscores the effectiveness of integrating structured semantic extraction from SPARQL queries with carefully designed few-shot examples.

Among the ablation configurations, BQFS was appreciated for its intuitive formatting, yielding high aesthetic and clarity scores. Nonetheless, participants consistently identified crucial semantic details missing from BQFS explanations, underscoring that visual improvements alone cannot substitute for structured semantic extraction. NFS explanations retained comprehensive semantic content but lacked intuitive formatting. While recognized for completeness, these explanations were seen as less readable and somewhat mechanical, reinforcing the necessity of structured examples for user-friendliness. BQB consistently received the lowest ratings due to the absence of both structured content and formatting guidance. Participants frequently noted its lack of clarity, incomplete explanations, and poor utility for practical use. Overall, OD was strongly preferred over both BQB and NFS. Preferences between OD and BQFS were more mixed, with participants acknowledging BQFS's appealing format, yet consistently preferring OD for its comprehensive content coverage, aligning with the conclusion in Section 4.3.1.

These results highlight that effectively combining structured semantic extraction with intuitive, example-based formatting is essential for producing high-quality NLEs for SPARQL queries.

5 Related Works

One-turn NL2SPARQL. Recent one-turn SPARQL generation approaches use LLMs to translate natural language questions into queries in a single pass. These include entity-pretrained GPT models (Bustamante and Takeda, 2024), output-vocabulary adaptation (Banerjee et al., 2023), KG-informed decoding (SPARKLE (Lee and Shin, 2024)), domain-specific fine-tuning (Rangel et al., 2024), dynamic program induction (ArcaneQA (Gu and Su, 2022)), and few-shot learning (Li et al., 2023). While these improve generation accuracy, they remain one-turn systems that do not provide structured explanations or support iterative human-guided correction. ASTs have also been used in NL2SQL work (Yu et al., 2018; Guo et al., 2019; Rubin and Berant, 2021; Scholak et al., 2021), but as *generation-time constraints* that guide decoder output. In contrast, InteracSPARQL parses an *existing* query's AST to produce a grounded explanation for iterative refinement—an *explain-then-refine* paradigm not explored in prior work.

Knowledge-Based Question-Answering (KBQA). KBQA systems answer questions over knowledge graphs without generating explicit SPARQL. Universal QA platforms (Bouziane et al., 2015; Omar et al., 2023), StructGPT (Jiang et al., 2023), and multi-task frameworks like UnifiedSKG (Xie et al., 2022) enable reasoning over structured data. However, KBQA systems lack transparency regarding the underlying query logic, preventing users from verifying results or correcting errors. InteracSPARQL addresses this gap by providing explicit NLEs that expose the full SPARQL reasoning chain.

Interactive SPARQL Query Refinement. Prior interactive approaches include result-based query refinement (SPARQLIt (Amsterdamer and Callen, 2021)), provenance-guided inference (Abramovitz et al., 2018), dependency-based NL-to-SPARQL translation (PAROT (Ochieng, 2020)), and formal query modification (Jian et al., 2020). These systems do not produce human-friendly explanations systematically. InteracSPARQL fills this gap by integrating structured NLEs that make query logic transparent to both human users and automated feedback loops.

6 Conclusion

We introduced InteracSPARQL, a training-free framework combining a Hybrid NLE module with tool-augmented refinement. By anchoring LLM outputs with deterministic AST parsing and external entity lookups, it mitigates hallucinations in zero-shot scenarios. Evaluations on QALD benchmarks show significant accuracy gains. The human studies confirm the clarity and utility of our NLEs.

Limitations

While INTERACSPARQL demonstrates significant improvements in SPARQL refinement, several limitations remain. First, our current evaluation focuses exclusively on SELECT and ASK queries. Although the framework is theoretically compatible with CONSTRUCT and DESCRIBE forms, we have not rigorously benchmarked performance on these query types, as there is no high-quality benchmark for those types. Second, the multi-step refinement process inherently incurs higher latency and token costs than single-turn generation models, though our ablation studies suggest that limiting the loop to $N = 5$ iterations balances cost and accuracy. Finally, our NLE generation and user studies were conducted primarily in English. Therefore, extending the hybrid explanation module to low-resource languages remains an open challenge for future work.

References

Ibrahim Abdelaziz, Razen Harbi, Zuhair Khayyat, and Panos Kalnis. 2017. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. *Proc. VLDB Endowment*, 10(13):2049–2060.

Efrat Abramovitz, Daniel Deutch, and Amir Gilad. 2018. Interactive inference of sparql queries using provenance. In *Proc. 34th IEEE Int. Conf. on Data Engineering*, pages 581–592.

Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. 2022. A survey of RDF stores & SPARQL engines for querying knowledge graphs. *VLDB J.*, 31(3):1–26.

Yael Amsterdamer and Yehuda Callen. 2021. Sparqlit: Interactive sparql query refinement. In *Proc. 37th IEEE Int. Conf. on Data Engineering*, pages 2649–2652.

Renzo Angles and Claudio Gutierrez. 2008. The expressive power of SPARQL. In *Proc. 7th Int. Semantic Web Conf.*, pages 114–129.

Marcelo Arenas and Jorge Pérez. 2011. Querying semantic web data with SPARQL. In *Proc. 30th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 305–316.

Marcelo Arenas and Martin Ugarte. 2017. Designing a query language for rdf: Marrying open and closed worlds. *ACM Trans. Database Syst.*, 42(4):21:1–21:46.

Debayan Banerjee, Pranav Ajit Nair, Ricardo Usbeck, and Chris Biemann. 2023. The role of output vocabulary in t2t lms for sparql semantic parsing. *Preprint*, arXiv:2305.15108.

Nikita Baramiia, Alina Rogulina, Sergey Petrakov, Valerii Kornilov, and Anton Razzhigaev. 2022. Ranking approach to monolingual question answering over knowledge graphs. In *Proceedings of the 7th Natural Language Interfaces for the Web of Data (NLIWoD) co-located with the 19th European Semantic Web Conference*, Hersonissos, Greece.

Manuel Borroto, Francesco Ricca, Bernardo Cuteri, and Vito Barbara. 2022. SPARQL-QA enters the QALD challenge. In *Proceedings of the 7th Natural Language Interfaces for the Web of Data (NLIWoD) co-located with the 19th European Semantic Web Conference*, Hersonissos, Greece.

Abdelghani Bouziane, Djelloul Bouchiha, Noureddine Doumi, and Mimoun Malki. 2015. Question answering systems: Survey and trends. *Procedia Computer Science*, 73:366 – 375.

Diego Bustamante and Hideaki Takeda. 2024. Sparql generation with entity pre-trained gpt for kg question answering. *Preprint*, arXiv:2402.00969.

Papa Abdou Karim Karou Diallo, Samuel Reyd, and Amal Zouaq. 2024. A comprehensive evaluation of neural sparql query generation from natural language questions. *IEEE Access*, 12:125057–125078.

Gonzalo I. Diaz, Marcelo Arenas, and Michael Benedikt. 2016. Sparqlbye: Querying RDF data by example. *Proc. VLDB Endowment*, 9(13):1533–1536.

Dennis Diefenbach, Pedro Henrique Migliatti, Omar Qawasmeh, Vincent Lully, Kamal Singh, and Pierre Maret. 2019. Qanswer: A question answering prototype bridging the gap between a considerable part of the lod cloud and end-users. In *The World Wide Web Conference*, WWW '19, page 3507–3510, New York, NY, USA. Association for Computing Machinery.

W3C SPARQL Working Group. 2013. SPARQL 1.1 Overview. `https://www.w3.org/TR/sparql11-overview/`. Accessed: 2024-09-04.

Yu Gu and Yu Su. 2022. Arcaneqa: Dynamic program induction and contextualized encoding for knowledge base question answering. *Preprint*, arXiv:2204.08109.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 query language. Accessible at http://www.w3.org/TR/sparql11-query/. Last accessed November 2015.

Olaf Hartig. 2012. SPARQL for a web of linked data: Semantics and computability. In *Proc. 9th Extended Semantic Web Conf.*, pages 8–23.

Olaf Hartig, Christian Bizer, and J.C. Freytag. 2009. Executing SPARQL queries over the web of linked data. In *Proc. 8th Int. Semantic Web Conf.*, pages 293–309.

Xun Jian, Yue Wang, Xiayu Lei, Libin Zheng, and Lei Chen. 2020. SPARQL rewriting: Towards desired results. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1979–1993.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *Preprint*, arXiv:2305.09645.

Jaebok Lee and Hyeonjeong Shin. 2024. Sparkle: Enhancing sparql generation with direct kg integration in decoding. *Preprint*, arXiv:2407.01626.

Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. 2012. SPAM: A SPARQL analysis and manipulation tool. *Proc. VLDB Endowment*, 5(12):1958–1961.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980, Toronto, Canada. Association for Computational Linguistics.

Baozhu Liu, Xin Wang, Pengkai Liu, Sizhuo Li, Qiang Fu, and Yunpeng Chai. 2021. Unikg: A unified interoperable knowledge graph database system. In *Proc. 37th IEEE Int. Conf. on Data Engineering*, pages 2681–2684.

Shicheng Liu, Sina J. Semnani, Harold Triedman, Jialiang Xu, Isaac Dan Zhao, and Monica S. Lam. 2024. Spinach: Sparql-based information navigation for challenging real-world questions. *Preprint*, arXiv:2407.11417.

Aisha Mohamed, Ghadeer Abuoda, Abdurrahman Ghanem, Zoi Kaoudi, and Ashraf Aboulnaga. 2022. RDFFrames: Knowledge graph access for machine learning tools. *VLDB J.*, 31(2):321–346.

Peter Ochieng. 2020. Parot: Translating natural language to sparql. *Expert Systems with Applications: X*, 5:100024.

Reham Omar, Ishika Dhall, Panos Kalnis, and Essam Mansour. 2023. A universal question-answering platform for knowledge graphs. *Proc. ACM Manag. Data*, 1(1).

Aleksandr Perevalov, Xi Yan, Liubov Kovriguina, Longquan Jiang, Andreas Both, and Ricardo Usbeck. 2022. Knowledge graph question answering leaderboard: A community resource to prevent a replication crisis. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2998–3007, Marseille, France. European Language Resources Association.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):1–45.

Sukannya Purkayastha, Saswati Dana, Dinesh Garg, Dinesh Khandelwal, and G.P Shrivatsa Bhargav. 2022. A deep neural approach to KGQA via SPARQL silhouette generation. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Julio C. Rangel, Tarcisio Mendes de Farias, Ana Claudia Sima, and Norio Kobayashi. 2024. Sparql generation: an analysis on fine-tuning openllama for question answering over a life science knowledge graph. *Preprint*, arXiv:2402.04627.

Md Rashad Al Hasan Rony, Uttam Kumar, Roman Teucher, Liubov Kovriguina, and Jens Lehmann. 2022. Sgpt: A generative approach for sparql query generation from natural language questions. *IEEE Access*, 10:70712–70723.

Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, Online. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

X. Shen, L. Zou, M. T. Özsu, L. Chen, Y. Li, S. Han, and D. Zhao. 2015. A graph-based RDF triple store. In *Proc. 31st IEEE Int. Conf. on Data Engineering*, pages 1508–1511. System demonstration paper.

Kanchan Shivashankar, Khaoula Benmaarouf, and Nadine Steinmetz. 2022. From graph to graph: AMR to SPARQL. In *Proceedings of the 7th Natural Language Interfaces for the Web of Data (NLIWoD) co-located with the 19th European Semantic Web Conference*, Hersonissos, Greece.

Nhuan D. To and Marek Reformat. 2020. Question-answering system with linguistic terms over rdf knowledge graphs. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4236–4243.

Ricardo Usbeck, Xi Yan, Aleksandr Perevalov, Longquan Jiang, Julius Schulz, Angelie Kraft, Cedric Moeller, Junbo Huang, Jan Reineke, Axel-Cyrille Ngonga Ngomo, Muhammad Saleem, and Andreas Both. 2023. Qald-10 — the 10th challenge on question answering over linked data. *Semantic Web – Interoperability, Usability, Applicability*.

W3C. 2006. SPARQL query language for RDF – Formal definitions. Accessible at http://www.w3.org/2001/sw/DataAccess/rq23/sparql-defns.html#defn_GroupGraphPattern. Last accessed December 2015.

Marcin Wylot, Manfred Hauswirth, Philippe Cudré-Mauroux, and Sherif Sakr. 2018. Rdf data storage and query processing schemes: A survey. *ACM Comput. Surv.*, 51(4):84:1–84:36.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, and 4 others. 2022. Unified-SKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 602–631, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

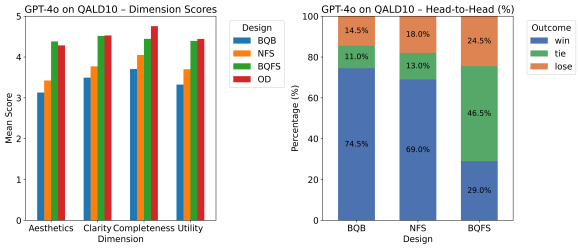
Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023. DecAF: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

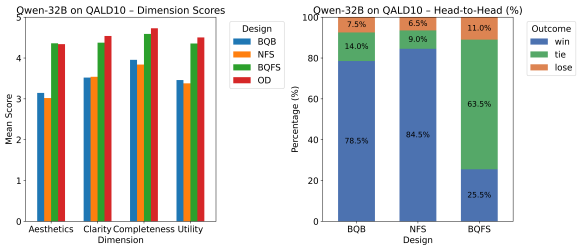
Weiguo Zheng and Mei Zhang. 2019. Question answering over knowledge graphs via structural query patterns. arXiv 1910.09760. *Preprint*, arXiv:1910.09760.

Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over rdf: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 313–324, New York, NY, USA. Association for Computing Machinery.

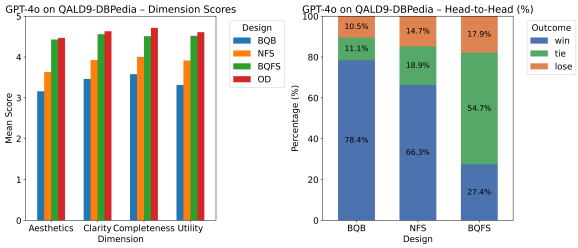
A Detailed Human Evaluation Result in Section 4.4.2



(a) GPT-4o on QALD-10



(b) Qwen-2.5-32B on QALD-10



(c) GPT-4o on QALD-9-DBpedia

Figure 3: Human evaluation results for the three conditions. Each subfigure shows mean dimension scores (left) and head-to-head percentages (right) for OD, BQB, NFS and BQFS.

B Detailed Background and Formal Definitions

This appendix provides the formal definitions and extended background on RDF and SPARQL to supplement the concise summary provided in the main text.

B.1 Resource Description Framework (RDF)

RDF and SPARQL are foundational technologies of the Semantic Web, a framework that aims to create a more intelligent, interconnected web. RDF, introduced by the World Wide Web Consortium (W3C), is a standard model for data representation and interchange on the web, facilitating the integration and sharing of information across different domains (Ali et al., 2022; Wylot et al., 2018; Abdelaziz et al., 2017; Angles and Gutierrez, 2008; Arenas and Pérez, 2011; Shen et al., 2015; Liu et al.,

Table 7: Human evaluation results comparing NLE designs. **Dimension Scores** represent mean ratings on a Likert scale (1–5). **Head-to-Head** columns show the percentage of participants who preferred the **Original Design (OD)** over the specific baseline, judged it a Tie, or preferred the baseline.

		Dimension Scores (1–5)				Head-to-Head Preference (%)		
Model & Dataset	**Design**	**Aesth.**	**Clarity**	**Compl.**	**Utility**	**OD Wins**	**Tie**	**OD Loses**
GPT-4o (QALD-10)	BQB	3.13	3.50	3.71	3.33	74.5	11.0	14.5
	NFS	3.42	3.77	4.05	3.70	69.0	13.0	18.0
	BQFS	**4.38**	4.51	4.45	4.39	29.0	46.5	24.5
	OD (Ours)	4.29	**4.53**	**4.76**	**4.44**	— Reference —		
Qwen-2.5-32B (QALD-10)	BQB	3.15	3.52	3.96	3.46	78.5	14.0	7.5
	NFS	3.02	3.54	3.84	3.38	84.5	9.0	6.5
	BQFS	**4.36**	4.38	4.59	4.36	25.5	63.5	11.0
	OD (Ours)	4.34	**4.54**	**4.73**	**4.50**	— Reference —		
GPT-4o (QALD-9-DBpedia)	BQB	3.16	3.46	3.58	3.32	78.4	11.1	10.5
	NFS	3.64	3.93	4.00	3.91	66.3	18.9	14.7
	BQFS	4.43	4.56	4.51	4.52	27.4	54.7	17.9
	OD (Ours)	**4.47**	**4.63**	**4.71**	**4.61**	— Reference —		

2021). RDF represents data as triples, consisting of a subject, predicate, and object (usually represented as (s, p, o)), which together form a graph structure. This simple yet flexible model allows for the expression of complex data relationships and supports interoperability between heterogeneous data sources.

Formally, an RDF dataset and an RDF graph are defined as follows:

Definition 1. *Let $\mathcal{I}, \mathcal{B}, \mathcal{L}$, and $\mathcal{V}$ denote the sets of all URIs, blank nodes, literals, and variables, respectively. A triple $(s, p, o) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is an RDF triple, where s, p and o are called subject, property (or predicate) and object. A set of RDF triples form a RDF dataset.*

Definition 2. *A* RDF graph *$G = \langle V, L_V, f_V, E, L_E, f_E \rangle$ is a six-tuple, where:*

1. *$V = V_r \cup V_l$ is a collection of vertices that correspond to all subjects and objects in RDF data, where V_r and V_l are collections of resource vertices and literal vertices, respectively.*
2. *L_V is a collection of vertex labels.*
3. *A* vertex labeling function *$f_V : V \to L_V$ is a bijective function that assigns to each vertex a label. The label of a vertex $u \in V_l$ is its literal value, and the label of a vertex $u \in V_r$ is its corresponding URI or the blank node identifier.*
4. *$E = \{\overrightarrow{u_1, u_2}\}$ is a collection of directed edges that connect the corresponding subjects and objects.*
5. *L_E is a collection of edge labels.*
6. *An* edge labeling function *$f_E : E \to L_E$ is a bijective function that assigns to each edge a label. The label of an edge $e \in E$ is its corresponding predicate (or called property).*

An edge $\overrightarrow{u_1, u_2}$ is an attribute property *edge if $u_2 \in V_l$; otherwise, it is a* link *edge.*

B.2 SPARQL Query Language

SPARQL, also standardized by the W3C (W3C, 2006; Angles and Gutierrez, 2008; Pérez et al., 2009; Arenas and Pérez, 2011; Hartig et al., 2009; Harris and Seaborne, 2013; Hartig, 2012), is a powerful query language designed to retrieve and manipulate RDF data. An intuitive definition of SPARQL is given below.

Definition 3. *A SPARQL query typically includes five parts:*

1. *The prefix declarations are used to simplify and shorten the URIs (Uniform Resource Identifiers) that are commonly used in RDF data. Prefix declarations allow the user to define a shorthand notation (a prefix) for a namespace URI, making the query more readable and easier to write.*
2. *The output part of a SPARQL can be in the form of a table of values of variables (`SELECT`), or in a RDF graph specified by a graph template substituting for the variables by each query solution in the graph template (`CONSTRUCT`), or testing whether or not a query pattern has a solution (`ASK`).*

3. *The dataset definition refers to the specification of the RDF dataset that a query operates on and is identified in the* `FROM` *clause.*

4. *The graph pattern matching part is specified in the* `WHERE` *clause and includes a set of triple patterns to be matched as well as* `OPTIONAL`, `UNION` *and* `FILTER` *operators. The data source to be matched is specified by* `FROM` *in this part.*

5. *The solution modifier part includes projection, distinct, order and limit operators defined over the graph pattern matching results.*

If the graph pattern matching part consists of only triple patterns (no `OPTIONAL`, `UNION` or `FILTER`), this is called a *basic graph pattern* (BGP) query. SPARQL has undergone significant enhancements since its original release in 2008. The current version, SPARQL 1.1 (Group, 2013), includes support for updates, property paths, aggregates, subqueries, negation, and nested queries, among other features.

Semantics and Example. The semantics of SPARQL are based on graph pattern matching using homomorphism, where the query engine searches for graph patterns specified in the `WHERE` clause against the RDF data. If the pattern matches, the variables in the query are bound to corresponding values from the RDF graph, and the query result is constructed accordingly. SPARQL supports various forms of graph pattern matching, including BGPs, optional patterns, and union patterns.

To illustrate the syntax and semantics of SPARQL, consider the query used as a running example in the main paper (Example 1). This query retrieves the answer to the natural language question "*What is the TV-show that starred Rowan Atkinson, had 4 seasons and started in 1983?*". This is formulated in a SPARQL query where the answer is bound to the variable `?tvShow` that satisfies four conditions:

- It must be **an instance of** (`wdt:P31`) the **television series** (`wd:Q5398426`) via the triple pattern `?tvShow wdt:P31 wd:Q5398426`;
- It must feature **Rowan Atkinson** (`wd:Q23760`) as a **cast member** (`wdt:P161`) via `?tvShow wdt:P161 wd:Q23760`;
- It must have its **number of seasons** (`wdt:P2437`) bound to `?seasons` via `?tvShow wdt:P2437 ?seasons`;
- It must have its **start date** (`wdt:P580`) bound to `?startDate` via `?tvShow wdt:P580 ?startDate`.

The first `FILTER` clause `FILTER(?seasons = 4)` ensures only shows with exactly four seasons are considered, while the second `FILTER` clause `FILTER(YEAR(?startDate) = 1983)` restricts results to those that began in the year 1983. Finally, the `SELECT` clause returns each matching `?tvShow`.

C Extended Methodology

This section presents a comprehensive elaboration of INTERACSPARQL methodology, providing a granular, step-by-step decomposition of the high-level pipeline introduced in Figure 2 of the main content. To demonstrate the practical utility and internal mechanics of our framework, we employ the same example in Figure 2 again, i.e., querying for *"What is the TV-show that starred Rowan Atkinson, had 4 seasons and started in 1983?"*. over WikiData. The following details trace the data transformation process through four dedicated visualizations (Figures 4–7), illustrating precisely how raw erroneous SPARQL code is parsed, grounded, and refined into a correct one.

C.1 AST Parsing

The conversion of a raw SPARQL query into a format suitable for NLE generation and iterative refinement relies on a rigorous parsing stage. We begin by ingesting the query string and transforming it into a structured intermediate representation.

Parsing Workflow. Figure 4 illustrates this transformation process. The pipeline accepts an initial SPARQL query (Input) from any source, whether drafted by a human or generated by a model. While functionally valid, raw queries (e.g., lines 1–6 in the figure, also the input query in Figure 2) rely on opaque identifiers that are difficult to debug without domain expertise.

Structure of the AST. To resolve this opacity, we parse the raw string into a JSON-based Abstract Syntax Tree (AST). As shown in the Output of Figure 4, this AST explicitly delineates the query's logical components. It breaks down the `WHERE` clause into distinct `BGP` objects (corresponding to lines 2–5) and `FILTER` expressions (line 6), while mapping every IRI (`NamedNode`) and literal to its specific term type. This structured form enables the subsequent rule-based module to systematically

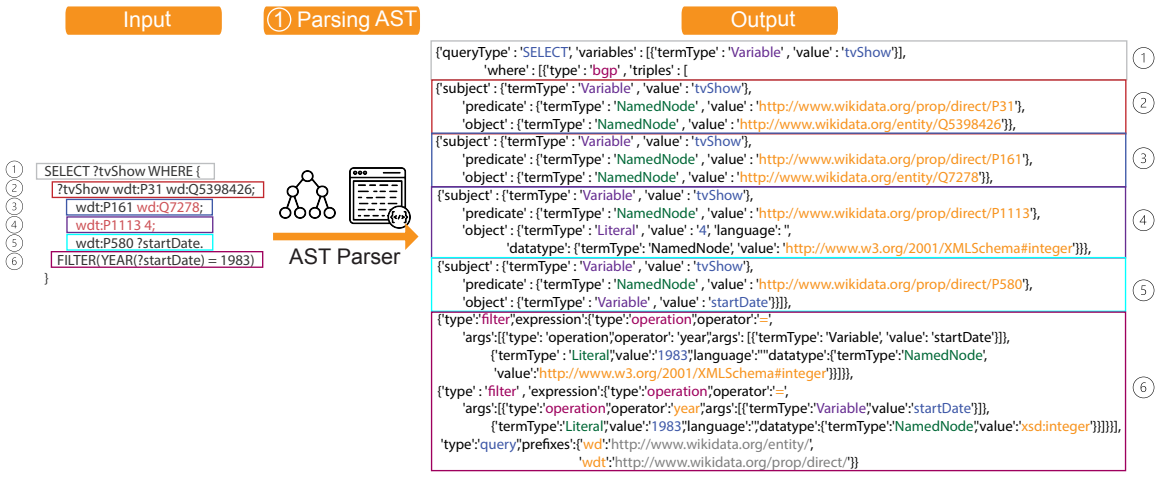


Figure 4: The AST parsing process (**Step ①**) in Figure 2. The raw SPARQL query string (left) is parsed into a structured JSON-based Abstract Syntax Tree (right), which explicitly identifies variable types, predicates, and filter operations for downstream processing.

isolate specific clauses, pinpoint variables of interest, and generate accurate explanations.

C.2 Hybrid 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 C.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 C.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.

C.2.1 Rule-Based NLE with Grounding

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. This workflow is visually depicted in Figure 5. The generation of the NLE given the query in Example 1 is provided in Example 2.

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 2) to enrich original entity identifiers with labels. As shown in the center block of Figure 5, this transformation minimizes cognitive overhead by turning opaque references (e.g., `wd:Q5398426`) into meaningful names like *"television series"*.

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

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.

Strategic Advantages. While rule-based generation is technically straightforward, its role in our framework is foundational for three reasons:

- **Hallucination Constraints:** By providing the LLM (in Stage 2) with a pre-generated, factually correct skeleton, we reduce the generation task to *paraphrasing* rather than *reasoning*. This significantly lowers the hallucination rate compared to end-to-end generation.
- **Deterministic Anchoring:** The rule-based output serves as a stable reference point. If the LLM's fluent explanation diverges from these rules, the system can flag the discrepancy as a potential error.
- **Computational Efficiency:** Performing label lookups and structural parsing via code is orders of magnitude faster and cheaper than relying on an LLM to "guess" IRI labels or parse syntax.

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.

C.2.2 LLM-refined NLE with Semantic Enrichment via Constrained Generation

Building upon the concise but sometimes brief explanations generated by the rule-based approach (Section C.2.1), we employ LLMs to transform these systematically derived outputs into a more fluent, context-rich narrative. This process is visually summarized in Figure 6. 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 linguis-

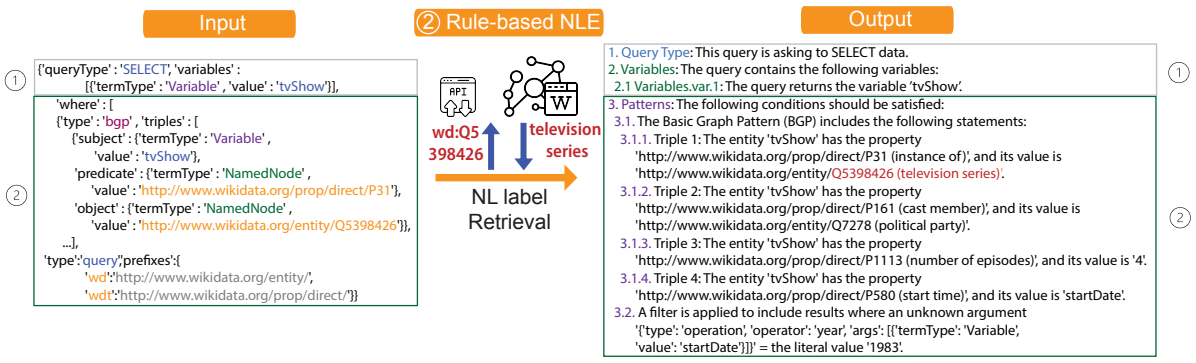


Figure 5: The Rule-Based Extraction process (**Step ②**) in Figure 2. The system traverses the AST (left) from **Step ①**, performs API lookups to resolve opaque IRIs like `wd:Q5398426` into labels like "television series" (middle), and generates a deterministic textual skeleton as rule-based NLE (right).

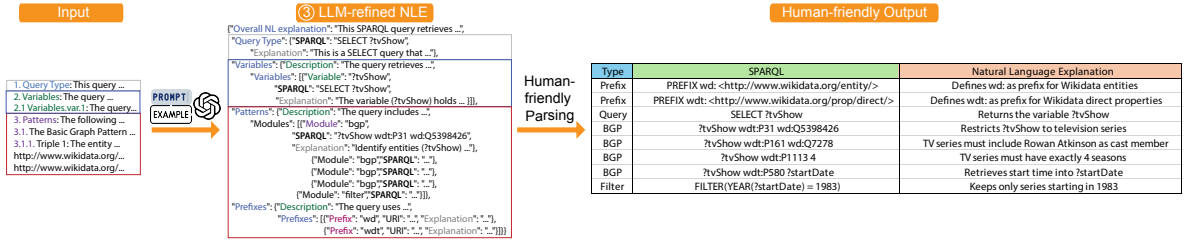


Figure 6: The LLM Semantic Enrichment process (**Step ③**) in Figure 2. The LLM takes the rule-based NLE in Step ② and a constrained instruction with examples, transforming it into a fluent, JSON-formatted narrative that explains the query's intent and logic.

tic 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 Example 3, 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.

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

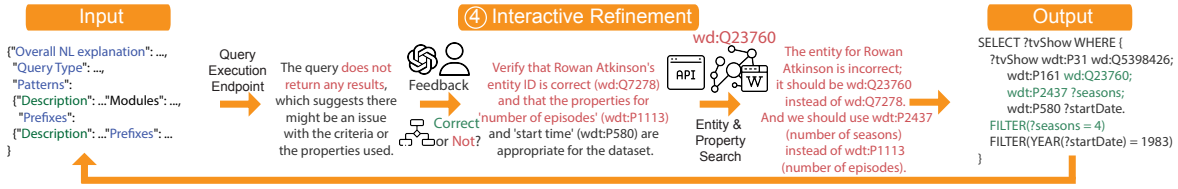


Figure 7: The Interactive Refinement process (**Step ④**) in Figure 2. The system executes the initial input raw query (Input), detects an empty result (Execution), generates specific feedback regarding the entity "Rowan Atkinson" (Feedback), invokes the entity search tool to find the correct IRI `wd:Q23760` and the property search tool to find `wdt:2437` (Tool Search), and produces the corrected SPARQL query (Output).

Example 2: 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'`.

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.

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.

C.3 Tool-Augmented Interactive Refinement

Extending the foundation established by our NLE framework (Section C.2.2), we employ an *iterative refinement* procedure, called INTERACSPARQL, that keeps SPARQL queries aligned with the user's original question. This process is visually detailed in Figure 7. 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.

C.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.

C.3.2 Execution Trace

We illustrate the workflow of the refinement loop using the running example shown in Figure 7

[1]For example, `<www.wikidata.org/entity/Q4346375>` in WikiData, whose label is "Association for Computational Linguistics".

(Querying for *"What is the TV-show that starred Rowan Atkinson, had 4 seasons and started in 1983?"*.).

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. In the specific case shown in Figure 7, the endpoint returns **no results**, prompting the system to investigate the discrepancy.

2. **Evaluate and Provide *Feedback*.** Should the query's output prove not to accurately reflect the user's intention, a feedback mechanism (either a human user or an LLM) identifies possible reasons for the mismatch. As shown in the figure, the feedback specifically targets the entity IRI: *"Verify that Rowan Atkinson's entity ID is correct (wd:Q7278)... and that properties... are appropriate."* This feedback delineates which segments of the query demand revision.

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 invokes a tool to perform an on-demand search. *Tool Action:* The search for "Rowan Atkinson" returns the correct IRI `wd:Q23760`, revealing that the previous IRI `wd:Q7278` was incorrect. The refined query modifies only this problematic reference, preserving previously validated logic.

4. **Repeat if Necessary.** The system executes the refined query anew, generating updated results and a refreshed NLE. 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.

C.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.

C.3.4 Self-Refinement Baseline

Under normal circumstances, Phase 2 (line 7) of Algorithm 1(i.e., "feedback") 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 Phase 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 Phase 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.

D Extended Experimental Setup

D.1 Implementation Details

The INTERACSPARQL framework is implemented in Python and is designed to be adaptable across any operating system supporting Python 3.8 or higher. The complete codebase comprises approximately 6,000 lines of code, organized into modular components handling data processing, NLE generation and refinement, the interactive refinement loop, and evaluation metrics.

In terms of computational resources, the core logic is lightweight and compatible with standard personal computing hardware. The primary resource requirement dictates either API access for proprietary models or a GPU for hosting open-source variants. For the latter, we found that a single NVIDIA A100 GPU is sufficient to serve models up to 32 billion parameters. Regarding operational costs, the full end-to-end refinement workflow averages $0.03 USD per query when using `GPT-4o`; this cost can be reduced by a factor of 10–15 when utilizing smaller efficient models like `GPT-4o-mini`, ensuring the framework remains economically viable for larger-scale deployments.

D.2 Datasets and Query Scope

We utilize a comprehensive set of SPARQL query benchmarks from the Question Answering over Linked Data (`QALD`) series. Specifically, we employ `QALD-10` and `QALD-9-Wikidata` for assessing performance on the Wikidata knowledge graph, and `QALD-9-DBpedia` for DBpedia. `QALD-9` offers broad coverage of SPARQL queries across both knowledge graphs, while `QALD-10` advances this by increasing the dataset size and complexity, offering a more challenging benchmark for Wikidata systems (Usbeck et al., 2023). We deliberately selected these human-annotated datasets to ensure a robust assessment of natural language understanding in practical usage scenarios. Our evaluation focuses strictly on `SELECT` and `ASK` queries, as these are the most frequently used command types and the only ones consistently contained across all human-labeled datasets.

D.3 Base Models and Query Engines

Our experimental framework is designed to be model-agnostic, allowing for the seamless integration of various LLMs with differing capabilities. We evaluate the performance of five specific LLMs: `GPT-4o` (version `gpt-4o-2024-08-06`), `GPT-4o-mini` (version `gpt-4o-mini-2024-07-18`), and `Claude-3.5-Sonnet` (version `claude-3-5-sonnet-20241022`). Among open-source models, we employ `Qwen-2.5-32B` and `Qwen-2.5-14B`. This selection allows us to verify that our pipeline remains robust across both proprietary and open-source architectures. To execute the SPARQL queries and retrieve data, we rely on the official public endpoints: the DBpedia SPARQL endpoint (`http://dbpedia.org/sparql`) and the Wikidata SPARQL service (`https://query.wikidata.org/`).

E Ablation Studies

E.1 Design Choice of NLE

To evaluate the impact of our NLE design, we conduct two complementary experiments: a direct ablation study on the quality of generated queries (Table 2) and a separate evaluation focusing on the effectiveness of the NLE during iterative self-refinement (Table 3). Both experiments compare four distinct configurations: **(a) Original Design (OD):** Incorporates structured semantic and hierarchical information extracted through rule-based methods from the AST, complemented by linguistically polished, LLM-refined narratives. Carefully designed few-shot examples in an accessible, structured format further guide the refinement; **(b) Bare Query Baseline (BQB):** Presents the original SPARQL query directly to the LLM, accompanied only by a generic instruction ("explain this query in natural language"), without structured guidance or few-shot examples; **(c) Bare Query with Few-Shots (BQFS):** Provides the original SPARQL query alongside structured few-shot examples derived from our methodology but omits explicit structured guidance from AST-derived rule-based NLEs, highlighting the influence of few-shot examples alone. **(d) No Few-Shots (NFS):** Employs the complete structured prompt but excludes few-shot examples, isolating the impact of explicit few-shot guidance.

As shown in Table 2, our original NLE design provides superior results across both `QALD-10` and `QALD-9-DBpedia` datasets when generating SPARQL queries directly from explanations. The OD approach achieves outstanding performance, reaching near-optimal F1 scores (0.977 for `GPT-4o` with `QALD-10`, 0.959 for `Qwen-2.5-32B` with `QALD-10`, and 0.974 for `Qwen-2.5-32B` with

QALD-9-DBpedia). These outcomes highlight that our structured explanations robustly preserve crucial query semantics and structural details, strongly aligning with insights from the human evaluation discussed in Section 4.4.2.

The performance trend remains consistent in the iterative self-refinement scenario (Table 3). Although the absolute F1 scores are lower due to the complexity of iterative refinement without ground truth NLE, the OD approach continues to exhibit notable advantages. Specifically, GPT-4o combined with OD significantly outperforms all baseline configurations on both datasets. This indicates that our detailed and structured explanations provide crucial scaffolding for the LLM-driven self-refinement process, enabling more accurate and meaningful iterative improvements.

An additional noteworthy observation is that the NFS setting, despite omitting explicit structured few-shot guidance, retains complete query information, enabling LLMs to effectively interpret query details even when provided in less structured formats. Nevertheless, the structured few-shot examples significantly enhance interpretability and clarity, an effect further substantiated by human evaluation results in Sec. 4.4.

Furthermore, simpler approaches such as BQB and BQFS occasionally achieve competitive performance, especially with the DBpedia dataset, benefiting from its semantic-rich identifiers and simpler query structures. Nevertheless, even in these favorable contexts, OD consistently surpasses other configurations, highlighting its robustness and consistent capability in capturing detailed query semantics.

Collectively, these experimental outcomes underscore the pivotal role of our structured and guided NLE design, both in direct SPARQL generation and iterative refinement, reinforcing its effectiveness and robustness.

E.2 Design Choice Ablation Study for Self-Refinement

To quantify the impact of our NLE design and external tool integration within the self-refinement loop (cf. Section C.3.4), we perform an ablation study on both QALD-10 and QALD-9-DBpedia datasets. Table 8 summarizes the performance of representative models—GPT-4o, Qwen-2.5-14B, and Qwen-2.5-32B—under four distinct configurations: **(a) OD:** Implements the full self-refinement pipeline, integrating the two-step NLE (rule-based extraction followed by LLM refinement) and external entity/property search tools; **(b) w/o NLE:** Omits the NLE module, thus relying solely on bare feedback of the external tools without structured explanation; **(c) w/o External Tools:** Retains NLE guidance but excludes external tool calls for entity/property resolution and query execution; **(d) w/o Both:** Removes both the NLE and external tool components, forcing iterative refinement with minimal guidance.

For GPT-4o, the comprehensive original configuration achieves an F1 score of 0.393 on QALD-10, which declines moderately to 0.351 without the NLE component. The absence of external tools results in a drastic performance drop to 0.112, and removing both elements further diminishes performance to an F1 score of 0.086. A similar trend emerges with Qwen-2.5-14B on QALD-10 and GPT-4o on QALD-10 datasets. These findings clearly highlight: **(a) critical role of NLE:** Structured explanations significantly enhance iterative refinement by providing interpretable query logic. The removal of NLE moderately decreases performance, indicating its vital role, especially in conjunction with external tools; **(b) essential external tools:** External entity/property resolution tools are crucial for accurately identifying IRIs and verifying query executions, evidenced by substantial performance drops upon their removal; **(c) synergistic interaction:** Optimal outcomes occur when both NLE and external tools are integrated, confirming the structured NLE's role in guiding external tool invocation and ensuring robust query refinement.

E.3 Design Choice of Maximum Refinement Iteration in Self-Refinement

To quantify how the maximum number of self-refinement iterations (N in Algorithm 1) affects final query accuracy, we evaluate GPT-4o and Qwen-2.5-32B on QALD-10 with $N \in \{1, 5, 10\}$ ($N = 5$ is our default). At each setting, the loop halts early if the generated SPARQL matches ground truth; otherwise, it proceeds until reaching N. We report precision, recall, and macro-averaged F1 score of the final query.

As Table 9 shows, allowing only a single iteration ($N = 1$) yields modest improvements: GPT-4o achieves an F1 of 0.307, and Qwen-2.5-32B reaches 0.225, since one round of self-refinement can correct simple errors but often leaves deeper misalignments (e.g., incorrect prop-

Table 8: Ablation study on design of self-refinement with both closed- and open-sourced LLMs. Macro-averaged F1 scores are applied here.

Model	Original			w/o NLE			w/o External Tools			w/o Both		
	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
GPT-4o + QALD-10	0.389	0.408	0.393	0.347	0.370	0.351	0.112	0.123	0.112	0.086	0.098	0.086
Qwen-2.5-14B + QALD-10	0.259	0.291	0.266	0.252	0.288	0.260	0.082	0.098	0.084	0.062	0.073	0.063
Qwen-2.5-32B + QALD-10	0.314	0.361	0.325	0.289	0.336	0.300	0.080	0.092	0.083	0.044	0.047	0.045
GPT-4o + QALD-9-DBpedia	0.549	0.551	0.532	0.536	0.536	0.518	0.496	0.506	0.490	0.501	0.514	0.497

Table 9: Ablation on maximum number of self-refinement iterations (N) for QALD-10.

Model	N	**Precision**	**Recall**	**F1**
GPT-4o	1	0.303	0.326	0.307
	5	0.389	0.408	0.393
	10	0.410	0.442	0.417
Qwen-2.5-32B	1	0.220	0.245	0.225
	5	0.314	0.361	0.325
	10	0.340	0.388	0.352

erty IRIs) unresolved. When $N = 5$, both models experience a substantial boost raising GPT-4o's F1 to 0.393 and Qwen-2.5-32B's to 0.325, as a result of the fact that most queries converge within five rounds of lookup and minor structural edits. Extending to $N = 10$ provides only marginal gains, with GPT-4o's F1 moving to 0.417 and Qwen-2.5-32B's to 0.352, because the majority of self-refinements have already converged by iteration five. These results confirm that while increasing N from 1 to 5 is crucial for achieving high accuracy, doubling from 5 to 10 incurs diminishing returns and typically does not justify the extra computational cost.

F Human Evaluation Details

F.1 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.

F.2 Participant Criteria

We recruit graduate-level participants with basic familiarity with knowledge graphs and databases but varied expertise with SPARQL to ensure broad, representative feedback. The experiment involves 20 participants with a gender distribution of 8 females and 12 males. The participants are native to 7 countries. Upon successful completion of the study, each participant received a reimbursement of 10 Canadian dollars.

F.3 Ethics Statement

This human study was conducted in accordance with the Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans (TCPS2, 2022) and reviewed and received ethics clearance through the University of Waterloo Research Ethics Board (REB #47266). Informed consent was obtained from all participants prior to their participation in the study. And we collect no personally identifiable information or offensive content in the study.

G Error Analysis Details

This appendix provides concrete examples of failure patterns from the error analysis in Section 4.2.4, based on the `GPT-4o` / `QALD-10` / $N = 5$ setting.

Property disambiguation failure. *"How many studio albums has Lana Del Rey have?"* The ground truth uses `wdt:P7937` (form of creative work) to filter for studio albums, but the model used `wdt:P31` (instance of). The entity IRIs (`wd:Q208569`, `wd:Q37150`) are correct; only the property is wrong. The tool search for "studio album" did not return `P7937` because this property requires knowledge that Wikidata models discography types via the specific "form of creative work" property rather than the more generic "instance of" relation.

Inverse-relation confusion. *"How many cities are part of the Pearl River Delta?"* The ground truth uses `wdt:P527` (has part) on the delta entity, querying it for its constituent cities. The model instead used `wdt:P361` (part of) on each city, the inverse relation applied to the wrong subject. Both properties are semantically plausible from the question text, but only one matches the ground truth. Entity IRIs are correct in both queries.

Structural mismatch. *"How many children (including adopted ones) does Jeff Bezos have with his ex-wife?"* The ground truth uses 5 triple patterns with the Wikidata statement-qualifier pattern `p:P26/ps:P26/pq:P582` (encoding "spouse" with an "end time" qualifier to identify the ex-wife), plus `wdt:P22/wdt:P25` for paternal/maternal relations. The model produced a simpler 2-triple query using `wdt:P26` (spouse, direct) and `wdt:P40` (child), missing the qualifier-based substructure entirely.

Future directions. These failure patterns suggest two concrete directions: (1) *schema-aware disambiguation*: augmenting the search tools with knowledge graph schema information (property domains, ranges, and usage statistics) to improve retrieval for ambiguous properties; and (2) *structural template guidance*: providing the LLM with query skeleton templates for common patterns (e.g., qualifier-based temporal filtering, multi-hop property paths) to help it produce the correct structure before IRI refinement begins.

H Refinement Cost Analysis

To quantify the cost-performance trade-off of our self-refinement pipeline, we re-ran the `GPT-4o` self-refinement on `QALD-10` with per-query token tracking enabled. Table 10 combines these measurements with the iteration ablation from Table 9.

Table 10: Single-turn vs. multi-step cost-performance trade-off (`GPT-4o`, `QALD-10`). Token counts and costs at $N = 5$ are measured; others are estimated from per-iteration averages.

Setting	**Tokens/Query**	**Cost/Query**	**F1**
Raw (single-turn)	0	\$0	0.136
Self-refine $N = 1$	$\sim$2,000	$\sim$\$0.006	0.307
Self-refine $N = 5$	6,205	\$0.020	0.393
Self-refine $N = 10$	$\sim$12,000	$\sim$\$0.040	0.417

The trade-off is favorable: for \$0.020 per query on average, self-refinement nearly triples the F1 score (0.136 to 0.393). The first iteration captures most of the improvement (0.171 F1 gain for $\sim$\$0.006), while going from $N = 5$ to $N = 10$ yields diminishing returns, showing our early-stopping mechanism is effective. With open-source models (e.g., `Qwen-2.5-32B` on a single A100 GPU), the API cost is zero.

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