

Can Skill-Driven LLM Agents Solve Text2SPARQL through Autonomous Exploration and Iterative Refinement?

Nandana Mihindukulasooriya¹, Niharika Shimona DSouza¹, Satyananda Kashyap¹,
Oshani Seneviratne² and Horst Samulowitz¹

¹IBM, USA

²Rensselaer Polytechnic Institute, USA

Abstract

Knowledge Base Question Answering (KBQA) requires systems to answer natural-language questions on structured knowledge graphs such as Wikidata. In Text2SPARQL settings, this task requires generating an executable SPARQL query that can extract the intended answer. However, formulating the correct SPARQL query often requires multiple steps of exploration and reasoning: linking elements of the natural-language question to the appropriate entities and properties, determining how the relevant facts are modeled in Wikidata, identifying a feasible path to the answer among alternative graph patterns, and interpreting intermediate results to refine the query-generation strategy.

This paper presents an agentic, skill-driven approach to the WikiKGQA Text2SPARQL task. Rather than treating query generation as a single-step translation problem, the proposed system decomposes the process into a sequence of capabilities, including entity and relation search, graph exploration, query construction, endpoint execution, result validation, and iterative refinement. We evaluate the approach on the English WikiKGQA tracks with mention and no-mention annotations (i.e., pre-supplied entity and property links, including superlatives and quantifiers) and analyze the resulting agent trajectories to determine when autonomous exploration contributes to successful query generation.

The proposed approach achieves an F1 score of 0.88 in the with-mentions setting, ranking first in the WikiKGQA: Wiki-Based Knowledge Graph Question Answering Challenge, and an F1 score of 0.81 in the no-mentions setting, ranking second. These results are obtained using a comparatively simple skill-based agentic configuration, without model fine-tuning or complex task-specific pipelines. Our case studies further show that explicit skills and execution-grounded feedback enable a frontier language model to discover Wikidata-specific modeling patterns, revise incorrect assumptions based on graph observations and query results, and generate accurate SPARQL queries for complex KBQA questions.

Source Code: <https://github.com/IBM/agentic-text2sparql/>

Keywords

Knowledge Base Question Answering, Text2SPARQL, Wikidata, LLM agents, SPARQL, WikiKGQA

1. Introduction

Knowledge Base Question Answering (KBQA) makes structured knowledge accessible through natural-language (NL) interactions. The task is particularly challenging for large-scale knowledge graphs such as Wikidata, where relevant evidence may be distributed across entities, properties, statement qualifiers, class hierarchies, and modeling conventions that are not apparent from the surface form of a question. Moreover, the same fact may be represented in multiple equally valid ways, requiring a system to reason over alternative graph patterns and determine which modeling path is most appropriate for answering a given question. A user may pose questions spanning diverse domains, such as films, mountains, awards, organizations, dates, and quantitative comparisons. Answering these questions over Wikidata may require resolving the appropriate entity identifiers, selecting the correct properties and relation directions, traversing statement qualifiers or intermediate nodes, and applying temporal

Proceedings of the Wiki-Based Knowledge Graph Question Answering Challenge (WikiKGQA), co-located with the 25th International Semantic Web Conference (ISWC 2026), Bari, Italy, October 25-29, 2026

✉ nandana@ibm.com (N. Mihindukulasooriya); Niharika.DSouza@ibm.com (N. S. DSouza); satyananda.kashyap@ibm.com (S. Kashyap); senevo@rpi.edu (O. Seneviratne); samulowitz@us.ibm.com (H. Samulowitz)

🆔 0000-0003-1707-4842 (N. Mihindukulasooriya); 0009-0004-7657-3312 (N. S. DSouza); 0000-0002-4624-5690 (S. Kashyap); 0000-0001-8518-917X (O. Seneviratne); 0000-0002-6780-3217 (H. Samulowitz)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

filters, aggregation operators, or other query constraints. The challenge is therefore not merely to generate a query from a question, but to discover, during inference, how the semantics of the question map onto the particular modeling conventions of a given KG.

Text2SPARQL formulates KBQA as the generation of an executable SPARQL query from an NL question. This formulation is attractive because the final output is auditable: a generated query can be executed against the knowledge graph, inspected, and compared with the expected denotation. At the same time, it exposes a central difficulty. Correctness depends not only on language understanding, but also on grounding the question into the ontology and modeling practices of the target graph. A one-shot translation can fail for a variety of reasons, for example, when a plausible relation is used in the wrong direction, when Wikidata represents an event through a reified statement and qualifiers rather than a direct edge, or when the generated query fails to capture semantic constraints that are implicit in the question and therefore returns results that are valid with respect to the graph but broader than the user’s intended answer.

This paper investigates an agentic alternative. Instead of generating SPARQL in a single pass, a frontier language model such as GPT [1] or Claude [2] is equipped with explicit reusable agent skills for entity and property search, graph exploration, SPARQL generation, endpoint execution, result validation, and iterative refinement. Skills are reusable, versioned units of procedural knowledge: each bundles instructions, scripts, and reference material that encode how to perform a reliable multi-step operation, allowing domain expertise to be injected and revised without retraining the underlying model. Unlike a monolithic application prompt, they can be inspected, tested, and shared; unlike retrieval-augmented generation, they specify how to act rather than merely what facts to retrieve. The model is allowed to interact with Wikidata as an environment: it can search for identifiers, inspect local graph neighbourhoods for entities of interest, test candidate queries, observe empty results or errors, and revise its assumptions before producing the final answer. The research question thus is:

To what extent can a skill-driven language-model agent solve complex Text2SPARQL questions by autonomously exploring Wikidata and using query execution as feedback for iterative refinement?

We study this question in the context of the WikiKGQA challenge, focusing on the English Text2SPARQL with and without mention annotation tasks. The paper makes two contributions:

1. We present an agentic skill-driven approach for Text2SPARQL over Wikidata, in which modular agent Skills organize search, graph exploration, executable query generation, endpoint execution, validation, and refinement.
2. We empirically validate the approach using verified WikiKGQA metrics and qualitative trajectory analysis, showing how execution-grounded exploration helps explain both successful query construction and recurring failure modes.

2. Background and Related Work

Recent Text2SPARQL work has moved beyond treating KBQA as a one-shot sequence-to-sequence translation problem. SPINACH is the closest Wikidata precedent: it frames SPARQL generation as an interactive process in which an LLM agent searches Wikidata, inspects properties and examples, executes candidate queries, and refines them using endpoint feedback [3]. ARUQULA follows a similar ReAct-style pattern but generalizes it from Wikidata-specific APIs to RDF/OWL-oriented graph-exploration utilities, including entity, property, and class search, property examples, and SPARQL execution [4]. Graf von Data likewise uses a ReAct agent with search, describe, and query actions, emphasizing graph-independent deployment with open-weight models and text-search-enabled SPARQL endpoints for organizational knowledge graphs [5]. mKGQAgent employs a modular plan–action–feedback loop together with an experience pool of successful and unsuccessful trajectories, making it particularly relevant to skill and memory-based Text2SPARQL agents [6]. GRASP uses an LLM to interact with a

knowledge graph autonomously by interleaving function calls to search or query the knowledge graph with reasoning about previous and future steps to explore the knowledge and to reach the answer [7]. Like these systems, our approach treats Text2SPARQL as an inspectable search $\rightarrow$ explore $\rightarrow$ execute $\leftrightarrow$ refinement process rather than direct translation. Its main distinction is the use of explicit Agent Skills as reusable units of procedural knowledge. These skills modularize entity and property search, graph exploration, SPARQL generation, endpoint execution, validation, refinement, and example retrieval into inspectable components that can be independently revised using trajectory evidence [8]. Wikidata consequently serves not only as the target KG but also as an online feedback environment in which intermediate actions gather graph evidence, query execution tests system assumptions, and the final output also contains an executable SPARQL denotation rather than a free-form answer.

Other recent systems explore complementary mechanisms for supplying grounding evidence or adapting the generator. Wardenga and Käfer show that automatically extracted ShEx or SHACL data-shape context can improve LLM-based SPARQL generation, particularly for unseen or private graphs [9]. D’Abramo et al. investigate in-context learning and demonstration retrieval for text-to-SPARQL without fine-tuning [10], whereas Soru et al. fine-tune autoregressive code models on a large DBpedia question–SPARQL corpus, providing a supervised contrast to agentic exploration [11]. Additional recent work examines fine-tuned LLMs for SPARQL generation, memorization and knowledge injection, interactive SPARQL refinement, and decomposition-based query generation [12, 13, 14, 15].

3. Task Description

The WikiKGQA Text2SPARQL task evaluates a system’s ability to answer natural-language questions over Wikidata by generating SPARQL queries whose execution produces the intended answers. The benchmark defines two settings. In the *with-mentions* setting, the system receives the natural-language question together with annotations for the relevant entities and relations, reducing the entity-linking burden and shifting the focus toward Wikidata-specific modeling and query construction. In the more demanding *no-mentions* setting, the system receives only the question and must identify and ground the appropriate entities and properties through search and graph exploration before constructing the SPARQL query. For example, given the question “Which movie released in the last ten years has won the most Academy Awards?”, the *with-mentions* setting additionally supplies linked identifiers such as Q11424 (film) with P31 (instance of), P166 (award received) for “won”, and P577 (publication date) for “released”, whereas the *no-mentions* setting provides the question alone and requires the system to discover these through search. Errors made during these initial grounding steps may propagate through subsequent reasoning and query generation. In both settings, the system is expected to generate an executable SPARQL query and return its answers. Performance is evaluated by comparing the predicted answers with the gold-standard answers using Macro QALD F1 [16].

The benchmark includes questions that require simple entity-relation lookup as well as more complex operations: multi-hop joins, type constraints, temporal filters, aggregation, counting, superlatives, negation or exclusion, and statement-level qualifiers. Because Wikidata is a heterogeneous, collaboratively maintained graph, semantically similar facts may be represented through different modeling patterns. This motivates a system design that can verify graph structure during inference rather than relying only on memorized schema associations.

4. System Overview

We present an autonomous Text2SPARQL agent whose behavior is organized through explicit Agent Skills [8].

4.1. Agent Skills

An Agent Skill is a portable, directory-level capability that provides a general-purpose agent with task-specific procedural knowledge without modifying the underlying model parameters. Following the emerging Agent Skills specification, each skill contains a required `SKILL.md` file with structured metadata and operational instructions, and may also include executable scripts, reference documents, templates, and other reusable resources. The runtime initially exposes only the skill metadata and loads the detailed instructions and resources when the skill is activated.

Unlike a simple prompt, which is typically transient text placed directly in the model context, a skill is a named, modular capability that the runtime can discover and activate only when it is relevant to the task and state. Moreover, a skill can combine instructions with executable code and reusable artefacts such as documentation linked from the main `SKILL.md` text, making its behavior easier to test, maintain, and share independently of any single prompt.

Agent Skills provide a lightweight mechanism for specializing an LLM agent for Wikidata question answering. Unlike fine-tuning, they require neither a representative training corpus nor additional model optimization or retraining as the task, knowledge graph, or underlying model evolves. They also extend conventional prompt engineering by packaging procedures as inspectable, testable, and reusable capabilities rather than embedding them in a monolithic prompt. Skills complement retrieval-augmented generation: retrieval supplies factual evidence, whereas skills specify how the agent should act, including which tools to invoke, which modeling constraints to verify, and how to validate results.

The architecture uses progressive disclosure to limit context overhead. The LLM selects capabilities according to the current task, while the runtime loads their instructions, scripts, and references on demand. This is particularly useful for Text2SPARQL, which may involve identifier resolution, graph exploration, query construction, execution, validation, refinement, and example retrieval, although an individual question may require only a subset of these capabilities.

4.2. Skill-Driven LLM Agents

The agent separates Text2SPARQL into two closely connected phases: evidence acquisition and query construction. Given a natural-language question, it first determines the expected answer type and identifies the principal entities, relations, constraints, and aggregation requirements. It then searches for candidate Wikidata entities, properties, and classes and performs bounded graph exploration to verify how the relevant concepts are represented. These probes may examine relation direction, class membership, property paths, statement qualifiers, temporal constraints, and event-centered modeling.

Only after collecting this evidence does the agent construct a candidate SPARQL query. The query is executed against the endpoint, and the resulting denotation, answer shape, execution errors, and endpoint diagnostics are treated as additional evidence. Wikidata therefore participates directly in the inference process rather than serving only as a passive database queried at the end. Search and exploration actions establish hypotheses, query execution tests them, and validation determines whether the agent should terminate or revise the query.

When a candidate query fails or produces an implausible result, the agent performs localized refinements rather than regenerating the query without diagnosis. Typical refinements include replacing a property, reversing an edge direction, moving from the direct-property layer to the statement and qualifier layers, adjusting a date constraint, correcting an aggregation condition, or revising the expected answer type. The final output consists of the validated SPARQL query and the answer array obtained from its execution.

This execution-grounded workflow addresses several common KBQA failure modes. It reduces unsupported schema assumptions by verifying graph structure before query construction, limits hallucination by requiring executable evidence, and supports interpretable refinement by associating failures with specific modeling assumptions. The resulting trajectories can also be inspected after execution and used to refine the corresponding skills.

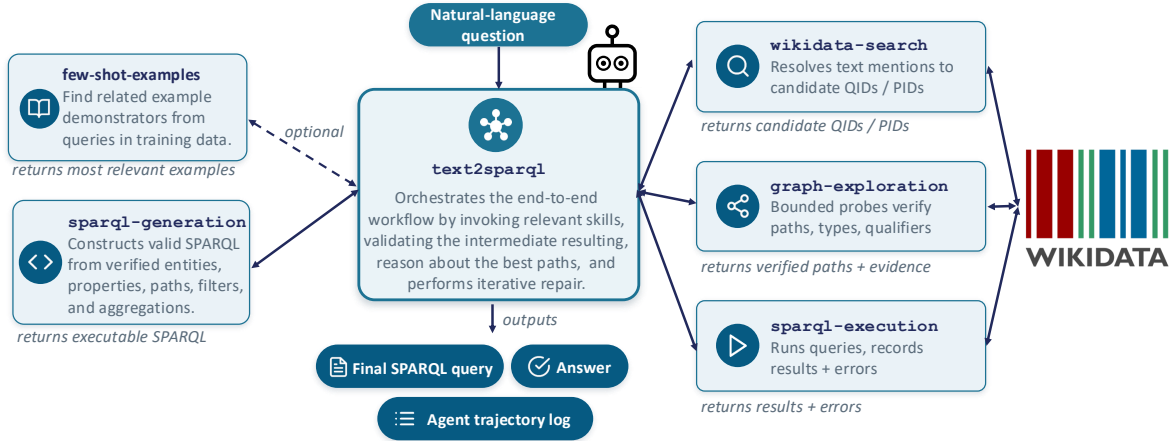


Figure 1: Skill-driven Text2SPARQL workflow for knowledge-base question answering over Wikidata. It shows the six skills used by the system: text2sparql, wikidata-search, graph-exploration, sparql-generation, few-shot-examples, and sparql-execution.

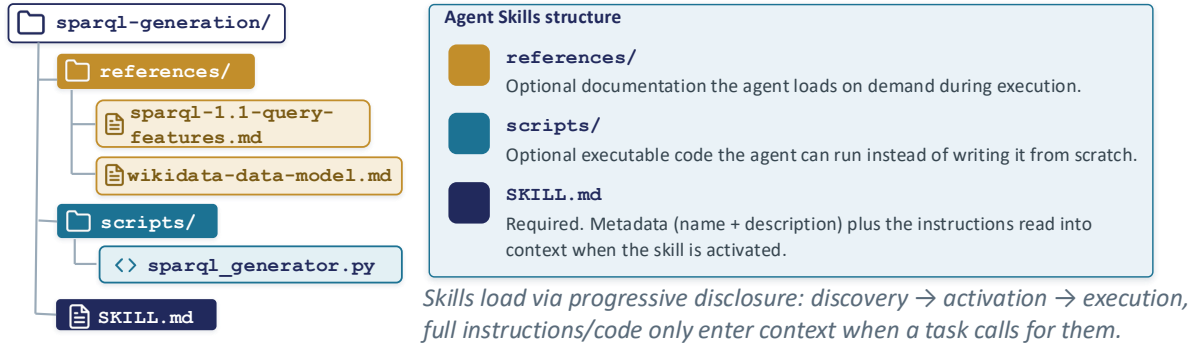


Figure 2: Structure and resources of the sparql-generation skill.

4.3. An Example Skill: sparql-generation

We use the sparql-generation skill to illustrate how the skills are represented as concrete and inspectable task modules rather than informal prompt fragments. Its SKILL.md frontmatter specifies the skill name, activation conditions, and runtime requirements, such as Python and uv. The instruction body defines the required inputs from preceding steps, supported operations, safety constraints for read-only query generation, expected return structures, query templates, generation rules, and common errors to avoid.

The skill directory additionally contains the executable helper scripts/sparql_generator.py. The script supports SPARQL syntax validation, query generation from structured entity and property inputs, and reusable query patterns such as direct and reverse lookup, type filtering, aggregation, qualifier access, temporal filtering, and top-k ordering. These operations expose documented arguments and structured outputs, converting recurring query-construction procedures into a reusable programmatic interface.

The skill also bundles reference material under references/. One document summarizes relevant SPARQL 1.1 constructs, including property paths, VALUES, BIND, aggregation, subqueries, negation, and solution modifiers. Another describes Wikidata-specific RDF modeling, including wdt: truthy statements, p:/ps:/pq: statement and qualifier access, ranks, normalized quantities, and datatype handling. These documents are loaded selectively when the agent encounters syntax or modeling uncertainty not covered by the core instructions.

This modular design is useful because many Text2SPARQL failures are procedural rather than purely linguistic. For example, an LLM may correctly recognize a temporal constraint but generate an endpoint-

incompatible literal; identify the appropriate property but incorrectly use the wrong namespace wdt : for a qualifier; or include an auxiliary ordering variable in the projected answer, thereby changing the expected result shape. Encoding such constraints in a skill makes them explicit, auditable, and independently revisable. When trajectory analysis reveals a recurring failure, the corresponding skill can be updated by adding a rule, reference note, template, or validation routine without retraining the underlying language model.

5. Running Example

Consider the question: “Which movie released in the last ten years has won the most Academy Awards?” Although concise, this question requires the system to resolve several distinct challenges: grounding the relevant entities and properties, interpreting the temporal constraint, identifying how Academy Awards are represented in Wikidata, and aggregating the number of awards received by each film.

A naive translation might search for films whose *award received* property points directly to a generic Academy Awards entity. Wikidata, however, does not consistently represent Oscar wins in this way. Films are typically linked through P166 (*award received*) to specific award-category items, such as the Academy Award for Best Picture or the Academy Award for Best Original Screenplay. Moreover, award-related information may be expressed through multiple property patterns. Consequently, the agent must explore the graph rather than assume a single representation.

The agent begins by invoking the wikidata-search skill with terms such as “Academy Awards”, “film”, “award received”, “publication date”, and “winner”. From the returned candidates, it identifies Q19020 as the Academy Awards, Q11424 as film, P166 as *award received*, P577 as *publication date*, and P1346 as *winner*.

Using the graph-exploration skill, the agent then investigates how these entities and properties are used in the graph. Rather than relying exclusively on the generic Academy Awards item, it examines both incoming and outgoing relations associated with specific Oscar categories. It also uses a recent Oscar-winning film, *Everything Everywhere All at Once* (Q83808444), as a probe entity.

This inspection confirms that the probe film has outgoing P166 statements whose values include specific Oscar categories, such as Q102427 (Academy Award for Best Picture) and Q41417 (Academy Award for Best Writing, Original Screenplay). It also evaluates P1346 (*winner*) as an alternative modeling pattern, but this relation produces less complete results for the target question, so the agent discards this pattern.

Based on this exploration of how individual Oscar award items are linked to their parent, it uses the sparql-generation skill to construct a query with the alternative paths feature in the SPARQL 1.1 Property Paths specification combining P31 (*instance of*), P279 (*subclass of*), P361 (*part of*) and P1269 (*facet of*). The query restricts candidate films to those having at least one P577 value within the ten-year interval and ranks them by the number of distinct Oscar-related awards received. The agent uses a declared reference date (i.e., the current system date) to calculate the last ten-year interval.

Listing 1: SPARQL query generated for the question “Which movie released in the last ten years has won the most Academy Awards?”

```
SELECT ?film ?filmLabel (MIN(?releaseDate) AS ?firstRelease) (COUNT(DISTINCT ?award) AS ?awards)
WHERE {
  ?film wdt:P31/wdt:P279* wd:Q11424; wdt:P577 ?releaseDate ; wdt:P166 ?award .
  ?award (wdt:P31|wdt:P279|wdt:P361|wdt:P1269) wd:Q19020 .

  OPTIONAL { ?film rdfs:label ?filmLabel . FILTER(LANG(?filmLabel) = "en") }
}
GROUP BY ?film ?filmLabel
HAVING(MIN(?releaseDate) >= "2016-07-21T00:00:00Z"^^xsd:dateTime)
ORDER BY DESC(?awards)
LIMIT 1
```

Table 1

Performance across the English WikiKGQA evaluation settings. F1 denotes Macro QALD F1. For the blind test sets, only the macro F1 scores were available; per-question outcome counts were not released. All experiments used the GPT-5.5 model.

Language	Setting			N	Performance			
	Mentions	Split	Blind		Correct	Partial	Wrong	F1
English	No mentions	Train	No	317	256	27	34	0.8643
English	No mentions	Dev	No	100	83	10	7	0.8940
English	With mentions	Dev	No	100	86	8	6	0.9042
English	No mentions	Test	Yes	75	–	–	–	0.8074
English	With mentions	Test	Yes	75	–	–	–	0.8755

Table 2

Public WikiKGQA Challenge 2026 leaderboard results.^{1 2}

EN-WITH-MENTIONS		EN-NO-MENTIONS	
System	F1	System	F1
ASTER (ours)	0.88	bastiscodabench	0.85
AMA-KBQA	0.87	ASTER (ours)	0.81
fokam_emic	0.65	AMA-KBQA	0.79
melisa_marian	0.63		

Table 3

Execution details for WikiKGQA English test sets.

Metric	EN-WITH MENTIONS	EN-NO MENTIONS
Avg. LLM rounds	5.07	6.31
Avg. Tool calls	7.59	13.07
Avg. SPARQL executions	2.27	2.16
Avg. SPARQL retries	1.28	1.16
Avg. Token footprint	85,013	112,626

Execution reveals an important temporal modeling problem. A film may have multiple P577 values corresponding to its original release, regional releases, festival screenings, or later re-releases. Filtering individual P577 statements can therefore admit an older film whenever one of its later release dates falls within the specified interval. To correct this overbroad interpretation, the agent revises the query to compute the earliest available release date for each film and applies the temporal constraint to that aggregated value. Listing 1 shows the refined query. The refined query ranks *La La Land* (Q20856802) first, with six counted Academy Awards and an earliest recorded release date of 2016-08-31.

This example highlights three desirable properties of the proposed approach. First, the agent does not assume a single schema pattern for the phrase “won Academy Awards”. Instead, it examines multiple candidate representations and evaluates them against the graph. Second, it uses execution feedback to identify and refine an overly permissive interpretation of the temporal constraint. Third, the reasoning trajectory is inspectable: it records which modeling assumptions were tested, which alternatives were rejected, and why the final query was selected. A demonstration of this work is presented in [17].

6. Evaluation and Experiment Results

Evaluation is based on Macro QALD F1 over the returned answer sets. For each question, precision and recall are computed by comparing the predicted answer set with the gold answer set, and the resulting question-level F1 scores are averaged across all questions. Table 1 summarizes the verified results available for the English WikiKGQA settings. For the training and development splits, the table reports the number of correct, partially correct, and incorrect answers in addition to Macro QALD F1. Because the test sets were evaluated blindly, only the aggregate challenge scores were available for those runs.

At the time of submission, the proposed approach achieved an F1 score of 0.88 in the with-mentions setting, ranking first in the WikiKGQA: Wiki-Based Knowledge Graph Question Answering Challenge and in the no-mentions setting, it achieved an F1 score of 0.81, ranking second, as illustrated in Table 2.

The execution statistics provide additional insight into the computational cost of the two settings. In

¹EN-WITH-MENTIONS, **Test** tab: <https://www.codabench.org/competitions/15591/#/results-tab>

²EN-NO-MENTIONS, **Test** tab: <https://www.codabench.org/competitions/15592/#/results-tab>

the with-mentions test set, the run involved 569 tool calls (avg 7.59), 170 SPARQL executions (avg 2.27), and 96 SPARQL retries (avg 1.28). In the no-mentions test set, it required 980 tool calls (avg 13.07), 162 SPARQL executions (avg 2.16), and 87 SPARQL retries (avg 1.16). Although the numbers of SPARQL executions and retries were comparable, the no-mentions setting required 411 additional tool calls, representing an increase of approximately 72.2%. This additional tool usage reflects the greater need for entity search, property discovery, and exploratory graph access when mentions are not supplied. The additional LLM rounds and tool calls also proportionally increase the average token footprint.

These execution statistics warrant further research into optimising Agent Skills to preserve autonomous exploration while reducing unnecessary interactions. Frequently repeated sequences, such as candidate resolution, graph-pattern validation, query construction, and result verification, could be encapsulated in higher-level scripted workflows. Skill instructions could also define clearer activation criteria, stopping conditions, and mechanisms for reusing previously obtained evidence. A hybrid approach could route common “happy-path” questions through deterministic, lower-cost workflows while retaining autonomous exploration and refinement for ambiguous or difficult questions, thereby balancing efficiency, flexibility, and performance.

7. Qualitative Analysis

The logged trajectories suggest that autonomous exploration helps most when a question cannot be solved by surface-form translation alone. We identify four recurring patterns.

Schema discovery rather than schema assumption. Many Wikidata questions require discovering how a concept is represented. In the train question “Which mountains were first ascended in the last eighty years?”, direct search for a simple “first ascent” property is insufficient. The trajectory instead probes a known mountain and discovers that first ascent is modeled as a *significant event* statement with a *point in time* qualifier. This statement-level pattern is then turned into a SPARQL query. Such behavior is difficult for a static pipeline that binds each phrase to a single property without checking graph evidence.

Execution feedback as refinement signal. SPARQL execution provides immediate feedback about whether a hypothesis is syntactically valid and denotationally plausible. Empty results, endpoint errors, unexpected literals, excessive answer counts, or suspicious dates can trigger targeted refinement. In the Oscar example, execution feedback reveals that a naive temporal filter can include older films because Wikidata stores multiple publication dates. The agent can then refine the date logic instead of abandoning the query.

Direction and qualifier verification. Relation direction and statement qualifiers are frequent sources of error in KBQA. Graph exploration allows the agent to check whether the answer entity should be in subject or object position and whether a fact is represented on the direct edge or on a reified statement. This is especially important for questions involving events, awards, memberships, rankings, dates, and counts.

Transparent failure analysis. Even when a final answer is partial or wrong, the trajectory can reveal where the process failed: entity linking, property selection, graph modeling, aggregation, endpoint behavior, or stopping criteria. This supports a research workflow in which skills can be improved from trajectory evidence rather than from aggregate scores alone.

8. Discussion

The results support the view that Text2SPARQL over Wikidata benefits from treating query generation as interactive evidence gathering. The approach combines the interpretability of executable semantic

parsing with the flexibility of agentic graph exploration. It is particularly well suited to questions where the correct query depends on Wikidata-specific modeling choices that are difficult to infer from the question text alone.

At the same time, autonomy introduces trade-offs. Tool use increases latency and cost. Refinements can cause semantic drift if the agent broadens a query to obtain non-empty results. Label-based heuristics may solve individual cases but can be brittle across languages or graph updates. Few-shot retrieval may help complex questions but can also bias the model toward inappropriate query patterns. These trade-offs suggest that future systems should learn policies for when to search, when to explore, when to retrieve examples, and when to stop.

The analysis also points to a broader methodological benefit. Because trajectories are retained, system improvement can be evidence-driven. Rather than only asking whether a final answer is correct, one can ask whether the agent found the right entities, verified the right graph pattern, used execution feedback appropriately, and refined the right assumption. This makes trajectory analysis a practical basis for optimizing skills.

9. Conclusion and Future Work

This paper presented an agentic skill-driven approach to Text2SPARQL for KBQA over Wikidata. The system equips a frontier language model with explicit skills for search, graph exploration, SPARQL generation, execution, validation, and refinement. Verified results on English WikiKGQA settings show strong development performance, and qualitative trajectory analysis illustrates how autonomous exploration can discover schema patterns and correct local errors.

As future work, we plan to mine trajectories systematically to revise skills, learn decision policies for tool use, strengthen ontology-grounded validation, reduce latency and cost, and extend the approach to multilingual and cross-lingual WikiKGQA settings.

Declaration of use of Generative AI

Generative AI tools were used solely to improve the readability of the manuscript and to correct grammatical and stylistic issues. The authors reviewed all generated suggestions and take full responsibility for the final content.

References

- [1] OpenAI, GPT-5.5 System Card, Technical Report, OpenAI, 2026. URL: <https://openai.com/index/gpt-5-5-system-card/>.
- [2] Anthropic, Claude Opus 4.8 System Card, Technical Report, Anthropic, 2026. URL: <https://www.anthropic.com/system-cards>.
- [3] S. Liu, S. Semnani, H. Triedman, J. Xu, I. D. Zhao, M. Lam, SPINACH: SPARQL-based information navigation for challenging real-world questions, in: Y. Al-Onaizan, M. Bansal, Y.-N. Chen (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2024, Association for Computational Linguistics, Miami, Florida, USA, 2024, pp. 15977–16001. URL: <https://aclanthology.org/2024.findings-emnlp.938/>. doi:10.18653/v1/2024.findings-emnlp.938.
- [4] F. Brei, L. Böhmann, J. Frey, D. Gerber, L.-P. Meyer, C. Stadler, K. Bulert, ARUQULA: An LLM-based Text2SPARQL approach using ReAct and knowledge graph exploration utilities, in: Proceedings of the First International TEXT2SPARQL Challenge, volume 4094 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 49–63. URL: <https://ceur-ws.org/Vol-4094/paper4.pdf>.
- [5] R. Dorsch, D. Henselmann, A. Harth, Graf von data: A knowledge graph question answering agent for organisational usage, in: Proceedings of the First International TEXT2SPARQL Challenge, volume 4094 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 32–48. URL: <https://ceur-ws.org/Vol-4094/paper3.pdf>.

- [6] A. Perevalov, A. Both, Text-to-SPARQL goes beyond english: Multilingual question answering over knowledge graphs through human-inspired reasoning, in: Proceedings of the First International TEXT2SPARQL Challenge, volume 4094 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 77–93. URL: <https://ceur-ws.org/Vol-4094/paper6.pdf>.
- [7] S. Walter, H. Bast, GRASP: Generic Reasoning And SPARQL Generation Across Knowledge Graphs, in: The Semantic Web – ISWC 2025, 2026, pp. 271–289.
- [8] Agent Skills Contributors, Agent Skills: Specification and documentation, 2026. URL: <https://agentskills.io/home>, online specification, accessed 14 July 2026.
- [9] J. G. Wardenga, T. Käfer, Leveraging data shapes in large language model contexts for question answering on public and private knowledge graphs, in: Proceedings of the First International TEXT2SPARQL Challenge, volume 4094 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 1–19. URL: <https://ceur-ws.org/Vol-4094/paper1.pdf>.
- [10] J. D’Abramo, A. Zugarini, P. Torroni, Investigating large language models for text-to-SPARQL generation, in: Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing, Association for Computational Linguistics, 2025, pp. 66–80. doi:10.18653/v1/2025.knowledgenlp-1.5.
- [11] T. Soru, S. Joshi, S. Tiwari, M. Shahinmoghdam, A. Panchbhai, Question answering over DBpedia with fine-tuned autoregressive models, in: Proceedings of the First International TEXT2SPARQL Challenge, volume 4094 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 20–31. URL: <https://ceur-ws.org/Vol-4094/paper2.pdf>.
- [12] T. Mecharnia, M. d’Aquin, Performance and limitations of fine-tuned LLMs in SPARQL query generation, in: Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK), International Committee on Computational Linguistics, Abu Dhabi, UAE, 2025, pp. 69–77. URL: <https://aclanthology.org/2025.genaik-1.8/>.
- [13] A. Gashkov, A. Perevalov, M. Eltsova, A. Both, SPARQL Query Generation with LLMs: Measuring the Impact of Training Data Memorization and Knowledge Injection, in: Web Engineering: 25th International Conference, ICWE 2025, Delft, The Netherlands, June 30 – July 3, 2025, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2025, p. 177–192. URL: https://doi.org/10.1007/978-3-031-97207-2_14. doi:10.1007/978-3-031-97207-2_14.
- [14] X. Jian, Z. Dong, M. T. Özsu, InteracSPARQL: An interactive system for SPARQL query refinement using natural language explanations, in: Findings of the Association for Computational Linguistics: ACL 2026, Association for Computational Linguistics, 2026, pp. 29599–29624. doi:10.18653/v1/2026.findings-acl.1480.
- [15] P. A. K. K. Diallo, A. Sharma, N. E. Fard, A. Zouaq, DeSQ: Decomposition-based SPARQL query generation, arXiv preprint arXiv:2606.00203 (2026).
- [16] R. Usbeck, X. Yan, A. Perevalov, L. Jiang, J. Schulz, A. Kraft, C. Möller, J. Huang, J. Reineke, A.-C. Ngonga Ngomo, et al., Qald-10—the 10th challenge on question answering over linked data: Shifting from dbpedia to wikidata as a kg for kgqa, *Semantic Web 15* (2024) 2193–2207.
- [17] N. Mihindukulasooriya, N. DSouza, S. Kashyap, O. Seneviratne, H. Samulowitz, ASTER: Agentic Text-to-SPARQL with Task Skills, Autonomous Exploration, and Refinement, in: ISWC 2026 Companion Volume, October 25–29, 2026, Bari, Italy, 2026.