From Transport to Grounding: Designing MCP Connectors for Reliable SPARQL Querying by LLMs

Talha Tahmid
University of Texas at Arlington
talhathmd@gmail.com

Humera Sabir
University of Texas at Arlington
humera.sabir@uta.edu

Ashraf Aboulnaga
University of Texas at Arlington
ashraf.aboulnaga@uta.edu

ABSTRACT

SPARQL endpoints over RDF knowledge graphs are a standard way to access semantic data, including when augmenting tabular data with knowledge-graph facts. When the consumer is a Large Language Model, the Model Context Protocol (MCP) provides a natural substrate for connecting the LLM to the endpoint, but the question of how to design an MCP connector for SPARQL has not been studied: the tools to expose and the prompts used to guide the LLM are open design choices that determine whether the LLM produces useful answers. We present GraSP-MCP, an MCP connector for SPARQL endpoints, and an empirical study of how its design affects answer quality. Our central finding is that the choice of tool surface dominates: on the QAWiki benchmark, a minimal MCP connector exposing only a `run_sparql` tool performs worse than no MCP at all, while a connector that decomposes query construction into entity grounding, property grounding, schema verification, and validated execution raises exact-match accuracy by 15 percentage points over the no MCP case. On Rhea, a structurally different knowledge graph without gold answer sets, a pairwise LLM judge prefers GraSP-MCP responses in 86% of 50 cases.

VLDB Workshop Reference Format:
Talha Tahmid, Humera Sabir, and Ashraf Aboulnaga. From Transport to Grounding: Designing MCP Connectors for Reliable SPARQL Querying by LLMs. VLDB 2026 Workshop: Tabular Data Analysis (TaDA).

VLDB Workshop Artifact Availability:
Artifacts available at https://github.com/lids-lab/grasp-mcp.

1 INTRODUCTION

Augmenting tabular data with semantic information from a knowledge graph is a common and useful technique in tabular data analysis. A simple and effective way to access a knowledge graph and retrieve the required parts is to issue a SPARQL query to a web-accessible endpoint of the knowledge graph or to an RDF database that stores it [1]. The query returns a structured result set that can augment an existing table with grounded, auditable facts drawn from the graph.

If the tabular data analysis is performed through a Large Language Model, the LLM should be able to issue these SPARQL queries. One way to give the LLM this capability is through the Model Context Protocol (MCP). MCP is a recent open specification that defines how an LLM client invokes capabilities exposed by an external server. In this paper, we study the question of what capabilities to expose when the server connects the LLM to a SPARQL endpoint of a knowledge graph.[1] We argue that using MCP as a transport protocol that simply executes SPARQL queries and returns results is not sufficient. The capabilities exposed by the MCP connector must enable the LLM to better ground its answers in the facts of the knowledge graph. To our knowledge, no prior work has studied how the design choices for an MCP connector accessing a SPARQL endpoint affect the quality of the LLM's answers.

To illustrate the importance of grounding, consider a question from the QAWiki benchmark for evaluating question answering and SPARQL query generation over Wikidata [14]: "Who played Eleven in Stranger Things?" Answering this question from the LLM's parametric memory is unreliable for a reason that is not obvious from the question itself. The literal string "Eleven" has several plausible referents in Wikidata: the natural number 11, a song by C418, an American e-sports player, and several other entities, in addition to the fictional character the question is asking about. An LLM that proceeds directly to SPARQL is likely to commit to whichever Wikidata identifier the string resembles most in its pretraining data, producing a query that runs cleanly and returns confidently wrong bindings to Wikidata identifiers. A correct answer requires the system to disambiguate the entity before composing the query: "Eleven" resolves to `Q27955792` (the fictional character), "Stranger Things" resolves to `Q19798734` (the TV series), the cast-member relation is `P161`, and the character-played qualifier is `P453`, which links the cast statement to the fictional character. Only after these are grounded in the contents of the knowledge graph does the SPARQL query return the correct answer. The same problem appears whenever an LLM augments a table with knowledge graph facts: each augmentation reduces to a SPARQL query whose entities must be resolved correctly.

In this paper, we present *GraSP-MCP* (for *Grounded SPARQL MCP*), an MCP connector that exposes SPARQL endpoints to LLMs and is designed to assist in grounding the LLM answers in knowledge graph facts. The central design decision is **which tools (i.e., capabilities) to expose to the LLM**. The simplest design exposes a single `run_sparql` tool and leaves the rest to the LLM's reasoning. We show that this performs poorly: the LLM defaults to writing queries with identifiers recalled from pretraining, producing syntactically valid queries that return wrong answers without error. A better design decomposes query construction into separate tools for entity grounding, property grounding, schema verification, and validated execution. The execution tool refuses queries that contain identifiers not produced by a grounding tool in the same session, converting the most common silent-failure mode into an observable

Proceedings of the VLDB Endowment. ISSN 2150-8097.

[1] This paper focuses on connecting an LLM to a SPARQL endpoint, but our techniques apply equally to connecting the LLM to an RDF database.

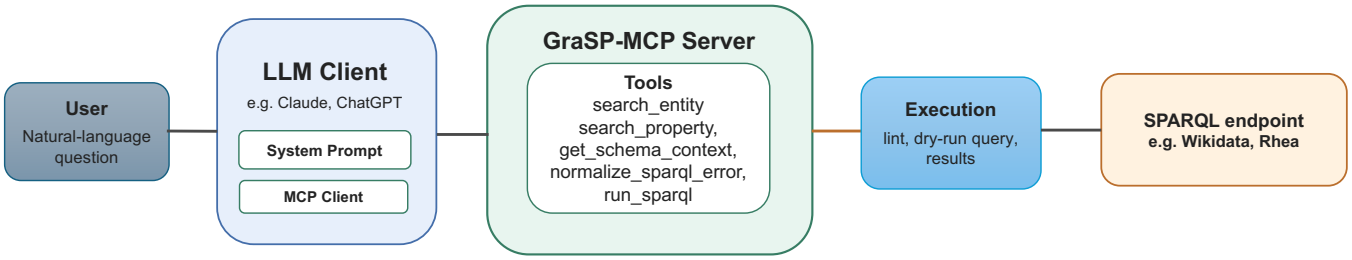

Figure 1: GraSP-MCP architecture. The LLM client contains a system prompt and an MCP client. The MCP client invokes the tools exposed by the GraSP-MCP server (Section 3). The execution step lints the query, dry runs it, and issues it against the SPARQL endpoint, returning results back through the server.

failure. These design principles are instantiated in a tailored MCP connector for each endpoint. Going beyond tool design, **prompt design** contributes additional gains: instructing the LLM to decompose the question, identify relation directions, and treat empty results as repair signals improves answer quality on top of a well-designed tool surface, but does little without one. The GraSP-MCP server is usable independently of the custom client we describe: any MCP-compatible client, including a standard chat interface, can connect to it and invoke its tools for grounding, schema inspection, and validated execution. The custom client adds a smart-prompting workflow that steers the LLM to use these tools in a grounding-first order, and the end-to-end workflow studied in this paper combines the two.

This paper makes three contributions: (1) The design of GraSP-MCP and its instantiation for the SPARQL endpoints of two knowledge graphs, Wikidata [18] and Rhea [3]. (2) An empirical study showing that the tool surface is the primary driver of answer quality. (3) Showing on the QAWiki benchmark that GraSP-MCP enables SPARQL that executes successfully on 91.5% of the questions with Claude Opus 4.7 and 94.6% with GPT-5, improving the fraction of questions where the predicted answer exactly matches the reference answer by 15 percentage points for Claude and 17 for GPT-5.

2 OVERVIEW OF MCP

An MCP deployment has two components (Figure 1). An *MCP server* exposes capabilities over a standard JSON-RPC transport. An *MCP client*, embedded in the LLM application, discovers these capabilities at session start and surfaces them to the model during inference. In our setting, an LLM client application connects to the GraSP-MCP server. The client supplies the LLM with a set of prompts that guide its use of tools (Section 4) and an MCP client that discovers and invokes the tools the server exposes. The GraSP-MCP server mediates all communication between the LLM and the underlying SPARQL endpoint, so the LLM does not query the endpoint directly; it interacts only through the tools and structured results the server exposes. The MCP server, on the other hand, is a standalone server application that must be hosted in a runtime environment and requires network connectivity to both the LLM client and the SPARQL endpoint.

The MCP protocol specifies how capabilities are advertised and invoked, and how results are returned. The protocol does not specify what these capabilities should do. What an LLM can reliably accomplish through MCP is determined by the design of these capabilities. The same GraSP-MCP server is used for all LLMs. However, a separate server is instantiated for each endpoint based on our design principles.

Two pieces of MCP terminology are important for this paper. A *tool* is a server-defined function the LLM may call, identified by a name and a JSON schema describing its arguments and return value. The LLM decides when and with what arguments to invoke a tool; the server executes the call and returns a structured result. A tool thus exposes a capability of the server to the LLM. A *resource* is a read-only artifact the server makes available to the LLM, for example a schema document or a set of prefix definitions. GraSP-MCP uses only the tool primitive: schema and prefix information is returned through tool calls rather than registered as resources.

Beyond exposing tools, the GraSP-MCP server applies multiple safety checks before SPARQL queries reach the endpoint, including a static linter and a dry run with limited result size. One advantage of GraSP-MCP sitting between the LLM and the SPARQL endpoint is that the system's behavior is easier to diagnose: when a question is answered incorrectly, the failure can be attributed to a specific layer such as query execution, query syntax, entity grounding, or semantic mismatch.

3 TOOL DESIGN

A SPARQL query is a compact specification of a graph pattern, but writing one correctly requires more than syntactic fluency. A system generating the query must know which entities in the knowledge graph correspond to the natural-language terms in the question, which properties encode the relations of interest, what direction those properties run, and what constraints the endpoint imposes to ensure that queries are well-behaved. Each of these is a separate decision that can fail independently. Current LLMs can proficiently compose syntactically valid SPARQL; the remaining question is whether that SPARQL returns the correct answer against a specific endpoint. That question is the focus of our tool design.

The simplest way to connect an LLM to a SPARQL endpoint is to expose a single MCP tool: `run_sparql`. The LLM composes a query, the tool sends it to the endpoint, and the result is returned to the LLM. In this case, MCP acts as a transport protocol, and the LLM is responsible for all aspects of composing the query: identifier resolution, property selection, query structure, and error recovery. **This minimal design performs poorly, because it asks the LLM to make every decision in a single generation step using only its parametric knowledge of the knowledge graph.**

The failures of the simple MCP design fall into four characteristic modes. First, the LLM produces *syntactically invalid queries* that the endpoint rejects before execution. Second, the LLM produces *queries with hallucinated identifiers*, in which a QID or PID is drawn from pretraining but does not correspond to the entity the question refers to. Third, the LLM produces *queries with incorrect property*

Table 1: Tools provided by the GraSP-MCP server for the Wikidata endpoint.

Tool	Description	Justification
`search_entity`	Resolves a natural-language entity name to Wikidata QIDs through the Wikidata search API.	Without grounding, the LLM uses hallucinated QIDs drawn from pretraining, producing silent failures with queries that are syntactically valid but semantically incorrect.
`search_property`	Resolves a natural-language relation name to Wikidata PIDs through the Wikidata search API.	Without grounding, the LLM confuses adjacent properties (e.g., P31 `instance_of` vs. P279 `subclass_of`) or invents PIDs that do not exist.
`get_schema_context`	Returns labels, descriptions, and datatypes for a given set of grounded QIDs and PIDs.	Verifies that a grounded identifier refers to the intended concept and narrows down the context to what the current question needs.
`run_sparql`	Lints the query, executes it once with `LIMIT 1` to detect errors cheaply, then executes it fully against the endpoint.	Without lint and dry run, syntax errors and unsafe constructs reach the endpoint, causing timeouts and triggering rate limits that affect subsequent queries.
`normalize_sparql_error`	Maps endpoint errors to a structured code (SYNTAX, TIMEOUT, RATE_LIMIT, ENDPOINT_ERROR) with a repair hint.	Without normalization, raw endpoint error strings are inconsistent across failure types, and the LLM cannot reason about how to repair them.

paths, either reversing the direction of a relation or selecting a property that is semantically adjacent but not the one the question requires. Fourth, the LLM produces *over-constrained queries* that execute but return an empty result set, because a filter is more specific than the data supports.

The first and third modes are at least observable as errors or visibly unhelpful results. The fourth produces an empty result that the LLM can in principle recognize as informative. The second produces no error at all. A query with a hallucinated identifier behaves to all appearances like a successful query: it parses, it executes, and it returns bindings to the knowledge graph. The LLM grounds its final answer in these bindings as if they answered the question. **Reducing this silent-failure mode is the principal design objective of the MCP tools we describe next.**

Three principles guide the design of the tools. The first is *decomposition*: replace one large decision (write a SPARQL query) with a sequence of smaller, verifiable decisions (ground the entities, ground the properties, verify the schema, then write the query). Each smaller decision is one the LLM can either succeed or fail at observably, where the original combined decision could fail silently. The second is *selective exposure*: expose tools and information narrowly. A small tool surface is easier for the LLM to choose from, and context returned for specific grounded identifiers is more useful than a full schema dump that crowds the context window. The third is *repair as a first-class operation*: treat errors and empty results as inputs to the next iteration rather than as final outcomes. The tools return structured feedback that the LLM can act on, so that local mistakes can be corrected without restarting from scratch.

Table 1 lists the tools provided by GraSP-MCP for Wikidata. The first three tools implement decomposition. `search_entity` and `search_property` require the LLM to resolve entity and relation names against the endpoint's actual content before composing a query. `get_schema_context` provides a verification step in which the LLM can confirm that a grounded identifier refers to the concept it intended, and at the same time narrows down the context returned to only the identifiers in use. The pairing of grounding and verification is what suppresses the silent-failure mode that motivated this design. The execution tool, `run_sparql`, enforces this discipline operationally: it refuses queries that contain identifiers not produced by a grounding tool in the same session.

The execution tool also addresses the first failure mode (syntactically invalid queries) and prevents queries from violating the endpoint constraints. Before sending a query to the endpoint, `run_sparql` applies a static linter that checks for blocked constructs, missing `LIMIT` clauses, and unsafe property paths. The query is then executed with `LIMIT 1` as a dry run to expose remaining errors at low cost. Only after both checks pass is the query executed in full. Following our third design principle, when the query fails, the `normalize_sparql_error` tool maps the failure to a structured code with a repair hint, so that the LLM can address the problem (e.g., relax a filter, retry with a corrected property) rather than restart from the original question. Errors and empty results in this design are not terminal, but rather signals for improvement.

The size of this tool surface is itself a design decision. A larger surface would expose more capabilities to the LLM, but it would also make tool selection noisier and increase the chance that the LLM invokes the wrong tool for the task. We deliberately keep the surface small and focused on the operations that directly support graph question answering. Internal server functions are deliberately hidden from the LLM. We show in Section 5 that this tool surface substantially improves answer quality over both the no-MCP baseline and the minimal-MCP configuration that exposes only the execution tool.

The tools in Table 1 expose composable primitives that depend on Wikidata's full-text search API over entity labels. A second knowledge graph used in our experiments, Rhea, does not expose such an API. We discuss in Section 5.4 how we apply our design principles to Rhea.

4 SMART PROMPTING

The availability of a tool through MCP does not determine when, how, or whether the LLM uses it. When we leave these decisions to the LLM, its use of tools is suboptimal in three recurring ways.

- **Skipping grounding and writing SPARQL from memory.** Even with `search_entity` and `search_property` registered as tools, the LLM often composes a query directly from QIDs and PIDs recalled from pretraining. The query is syntactically valid and runs without error, but its identifiers do not match the entity the question refers to: this is the silent-failure mode of Section 3, and the LLM does not invoke the grounding tools unless its context tells it that it must.

- **Selecting the wrong direction of a property.** SPARQL property paths are directional. The natural-language relations that questions use are often ambiguous about direction: "works by an author" and "authors of a work" refer to the same property used in opposite directions. Given a question phrased in either direction, the LLM tends to default to whichever direction it produced more often during pretraining rather than the direction the question requires. The query executes cleanly and returns bindings, but for the wrong side of the relation.
- **Treating empty results and errors as terminal rather than as signals.** An empty result or an execution error is informative: an empty set may mean the query is over-constrained or that the answer genuinely is empty, and an error code identifies what went wrong. Left to itself, the LLM does not distinguish these cases or use them to drive a repair.

To induce the LLM to use the GraSP-MCP tools correctly, we develop an LLM client application that accepts the user's question and answers it by sending a sequence of prompts to the LLM through its API. These prompts steer the LLM around the three failure modes above, guiding it to ground before composing, to make the direction of a relation explicit, and to treat errors and empty results as repair signals. The workflow of smart prompting is shown in Figure 2, and its steps are as follows.

(1) **Decomposition and grounding.** After accepting the user's question, the LLM client sends a prompt that instructs the LLM to decompose the question before composing any SPARQL: identify the entities being asked about, the relation that connects them, and the expected shape of the answer. The entity and relation parts of this decomposition must be resolved through `search_entity` or `search_property` before the LLM proceeds to query composition.

(2) **Direction-explicit pattern guidance.** The LLM client sends a second prompt providing three canonical SPARQL patterns covering the common shapes of relation-direction questions: forward (subject to object), reverse (object to subject), and bidirectional. Each pattern is paired with a short example drawn from a Wikidata question of that shape. Before composing the query, the LLM is asked to identify which direction the question requires and which pattern to specialize. This converts an implicit decision into an explicit one.

(3) **SPARQL execution and repair.** The next prompt sent by the LLM client over the API instructs the LLM to compose the SPARQL query based on the information collected thus far. The prompt explains the constraints enforced by `run_sparql` so that the LLM treats them as part of the task rather than arbitrary restrictions. The expected answer shape, while not operationally enforced, guides the LLM toward the right query form: a SELECT query for entity or literal answers, an ASK query for Boolean answers, and a tolerance for empty results when the question admits such a result.

If the SPARQL query returns an error, another prompt is sent to the LLM to repair the error and re-execute the query. If the SPARQL returns an empty result and the question cannot plausibly have an empty answer, this indicates an over-constrained query, and the LLM is instructed to relax the most specific filter and re-execute. The structured error codes returned by `normalize_sparql_error` carry the same kind of signal for execution failures: a SYNTAX code indicates a problem in the query itself, a TIMEOUT code indicates the query is too broad, and an ENDPOINT_ERROR code indicates a server-side issue not addressable by retry. The prompt instructs the LLM to use these codes to drive targeted repair, limiting the LLM to a small number of repair attempts per question to prevent unbounded loops on questions the system cannot answer.

5 EXPERIMENTS

5.1 Experimental Setup

GraSP-MCP is implemented in Python and is available as open source.[2] For our experiments, the GraSP-MCP server runs in the Render cloud platform and connects to the appropriate SPARQL endpoint.[3] The server is registered with the LLM, which makes all the tools exposed by the server available to the LLM. The LLM runs the MCP client that connects to the server over HTTP as needed. Our experiments use GraSP-MCP connectors for two SPARQL endpoints: Wikidata and Rhea.

Wikidata. The main knowledge graph for our experiments is Wikidata [18], the popular, free, collaboratively edited knowledge graph storing information about a broad set of topics. GraSP-MCP exposes the tools described in Section 3 for Wikidata, and queries it via the Wikidata Query Service (WDQS) SPARQL endpoint.[4] Queries are subject to a 30-second endpoint timeout and 500-row result cap.

We ask the LLM questions from the QAWiki benchmark [14]. QAWiki contains human-authored questions over Wikidata that can be answered with SPARQL queries, paired with the gold SPARQL query for each question. We use the `qawiki-v1-simple` release filtered to the 481 questions whose gold queries execute successfully against the current state of WDQS. When the LLM issues a SPARQL query on Wikidata, the query produces an *answer set*: a set of QIDs, literals, or boolean values returned by the endpoint. Comparing this answer set to the QAWiki reference answer set measures the quality of GraSP-MCP answers. We use four metrics for this comparison. **Exact match** is the fraction of questions for which the predicted answer set is identical to the reference answer set. We also compute the F1 measure between the answer set of each question and the reference answer set, treating each as a multiset of bindings. We report the **mean F1** and **median F1** across all questions. **Execution success** is the fraction of questions for which the generated SPARQL runs to completion without error.

Empty results is the fraction of executed queries returning zero rows. We do not treat an empty result as an execution error: in evaluation, an empty predicted answer set is compared with the reference answer set exactly as any other predicted answer set is. Of the 481 QAWiki questions in our evaluation, 24 have gold SPARQL queries that execute and return zero rows, so an empty prediction is scored correct under exact match precisely when the reference answer is also empty. We report empty results as a separate metric because an *unexpected* empty set is often diagnostic of a semantic problem, such as a wrong entity, a wrong property, a reversed relation direction, or an over-restrictive filter.

[2] https://github.com/lids-lab/grasp-mcp
[3] https://render.com
[4] https://query.wikidata.org

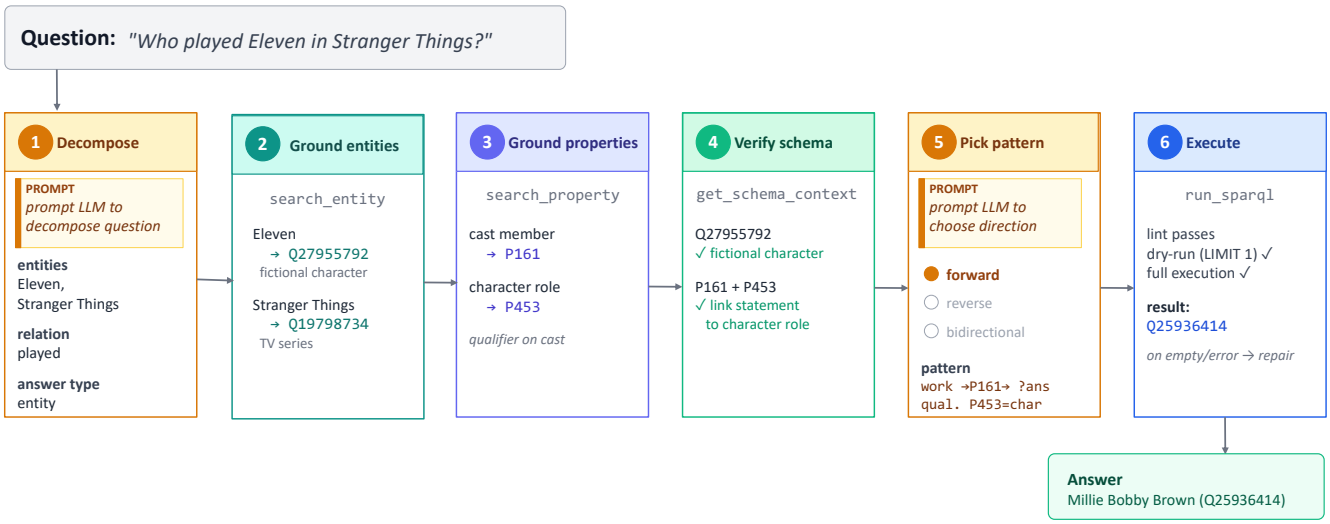

Figure 2: Smart LLM prompting workflow.

Table 2: Effect of MCP design choices on QAWiki answer quality. Results on 481 questions using the Claude Opus 4.7 LLM. Higher is better for Exact Match, Mean F1, Median F1, and Exec. Success. Lower is better for Empty Results. The Execute-only MCP configurations expose only the execution tool (`run_sparql` with lint and dry run) and no grounding tools. The Full MCP configurations use the five tools described in Section 3. Smart Prompting implements the workflow in Section 4.

Configuration	Exact Match ↑	Mean F1 ↑	Median F1 ↑	Exec. Success ↑	Empty Results ↓
Baseline (No MCP)	0.316	0.372	0.000	0.979	0.518
Execute-only MCP, Simple Prompting	0.295	0.343	0.000	0.865	0.555
Execute-only MCP, Smart Prompting	0.304	0.348	0.000	0.825	0.551
Full MCP, Simple Prompting	0.464	0.533	0.857	0.917	0.254
Full MCP, Smart Prompting	**0.468**	**0.550**	**0.920**	0.915	**0.216**

Rhea. We also experiment with the Rhea knowledge graph [3], an expert-curated knowledge graph of biologically relevant chemical and transport reactions. We query Rhea using its public SPARQL endpoint.[5] Queries are subject to a 30-second endpoint timeout and a result cap of 2,000 rows. Rhea is further discussed in Section 5.4.

LLMs. Our focus is on MCP design choices, not on comparing the capabilities of different LLMs. As such, we use the two frontier LLMs that are considered the best at the time of writing. Our primary LLM is Claude Opus 4.7 from Anthropic, and we also use GPT-5 from OpenAI to ensure that our results generalize to multiple LLMs and as a judge for the Rhea experiments. Both LLMs are accessed through their public APIs with default decoding parameters. The same MCP server is used for both without modification.

5.2 Evaluating Design Choices

We compare two options for the design of the MCP server: **Execute-only MCP** exposing only the execution tool (`run_sparql` with lint and dry run) and no grounding tools, and **Full MCP** exposing the five tools in Table 1. We compare two options for prompting: **Simple Prompting** that uses the LLM prompt unchanged, and **Smart Prompting** that implements the workflow in Section 4.

The first question we study in our experiments is the effect of each of these options on answer quality. To answer this question, we compare five configurations on the QAWiki benchmark using the Claude Opus 4.7 LLM. The first configuration is a baseline that does not use MCP: the LLM composes a SPARQL query from the question alone, and the query is sent directly to the Wikidata endpoint without the safety pipeline or grounding tools that GraSP-MCP provides. The remaining four configurations form a 2×2 grid over the two design choices: Execute-only MCP vs. Full MCP, and Simple Prompting vs. Smart Prompting. Table 2 reports the five quality metrics defined in Section 5.1 for all five configurations. Our discussion of the results focuses mainly on the Exact Match metric, but the other metrics convey the same message. The median F1 for the first three rows is zero because most of the results are empty. Three observations stand out:

- **The no-MCP baseline is weak on relation-heavy questions.** Without grounding tools to resolve entities and properties against Wikidata, the LLM composes SPARQL using identifiers recalled from pretraining. The resulting query is sent directly to the endpoint and often executes cleanly, but the identifiers it contains do not match what the question refers to. The baseline performs acceptably on questions whose answers are broadly available in pretraining text, such as well-known entities and common facts, but produces unreliable answers on questions that require specific identifiers or precise relations. The baseline's exact-match score of 0.316 reflects this limitation: without grounding, even a capable model cannot achieve competitive accuracy on QAWiki.
- **Adding MCP without careful design does not help.** Both Execute-only MCP configurations experience a small drop in exact match compared to the baseline, to 0.295 with Simple Prompting and 0.304 with Smart Prompting. This is not a large drop, so MCP does not severely degrade performance, but it certainly does not help. This is a central observation: **access to a SPARQL**

[5] https://sparql.rhea-db.org/sparql

Table 3: Baseline and full GraSP-MCP results for GPT-5.

Configuration	Exact Match ↑	Mean F1 ↑	Median F1 ↑	Exec. Success ↑	Empty Results ↓
Baseline (No MCP)	0.200	0.228	0.000	0.942	0.663
Full MCP, Smart Prompting	**0.374**	**0.434**	0.000	0.946	**0.281**

endpoint through a minimal MCP interface does not, on its own, improve answer quality. The Execute-only MCP adds the lint and dry-run safety checks but no grounding tools.

- **The Full MCP configuration delivers a substantial improvement.** With the five-tool Full MCP configuration, the LLM can ground its results in the Wikidata knowledge graph. Exact match rises by 15 percentage points compared to the baseline, reaching 0.464 with Simple Prompting and 0.468 with Smart Prompting. Smart Prompting contributes only a small additional gain on top of a well-designed tool surface.

The highest execution success in Table 2 belongs to the no-MCP baseline (0.979), and every MCP configuration scores lower. This should not be read as higher answer quality. Execution success measures only whether the generated SPARQL runs to completion, not whether it answers the question. As noted above, the baseline's syntactically valid queries built on hallucinated identifiers execute cleanly and count as successes while returning confidently wrong bindings. The MCP configurations route every query through the GraSP-MCP safety pipeline of linting and dry-run execution, and the full configuration additionally enforces the grounding discipline; these checks stop some queries the baseline would have run, converting silent failures into observable ones. The lower execution success of the MCP configurations is therefore a consequence of stricter validation, and the exact-match and F1 metrics show the opposite trend: the full tool surface substantially improves answer quality. Fully separating the effect of the safety pipeline from the absence of grounding tools would require an additional transport-only configuration without linting, which we leave to future work.

In summary, this experiment shows that exposing a set of MCP tools for grounding and validation substantially improves performance, whereas the ability to execute SPARQL alone, even with safety checks, is insufficient.

5.3 Results on the QAWiki Benchmark

Having established that the full GraSP-MCP design (rich tool surface plus smart prompting) substantially outperforms the no-MCP baseline, we now examine the results of the QAWiki benchmark in this configuration in more detail.

Generalization across models. The first question we ask is whether GraSP-MCP yields good results with frontier LLMs other than Opus 4.7. To answer this question, we run the same experiment as in Table 2 but using GPT-5 as the LLM and only two configurations: baseline and full GraSP-MCP. Table 3 shows the results. Without MCP, GPT-5 achieves 0.200 exact match. Full MCP with Smart Prompting adds 17 percentage points, reaching 0.374. This improvement is of similar magnitude to Opus 4.7's 15 percentage points and shows that the benefits of GraSP-MCP are not specific to a particular model. The accuracy numbers for GPT-5 are lower than those for Opus 4.7, indicating that Opus 4.7 is stronger in this particular case and validating our decision to use it as the default model.

Discrepancy between execution success and answering correctly. Focusing on the last row in Tables 2 and 3, and looking beyond the headline result of 15 to 17 percentage points improvement in accuracy, we see that both LLMs exhibit substantial divergence between execution success and answer correctness. Opus 4.7 produces SPARQL that runs successfully for 91.5% of questions but matches the gold answer on only 46.8%. GPT-5 shows the same pattern: 94.6% execution success versus 37.4% exact match. **This gap between executing successfully and answering correctly is an important empirical observation of this paper.** GraSP-MCP substantially reduces the surface-level failures (syntax errors, malformed queries, unsafe constructs) that historically dominate text-to-SPARQL evaluations. What remains is semantic grounding: choosing the correct entities, property paths, and filters, which the connector supports but does not by itself solve.

Opus 4.7 shows lower execution success than GPT-5 but substantially higher answer quality on every other metric (exact match, mean F1, and median F1). The pattern of lower execution success but higher correctness is consistent with Opus 4.7 being more conservative in composing queries: when it commits to a query, the query is more likely to ground the correct entities and relations.

The median F1 is particularly informative. A median of 0.920 against a mean of 0.550 indicates a strongly bimodal distribution: the typical question is answered with nearly the entire gold answer set recovered, while a long tail of failures pulls the mean down. This is consistent with the workflow the tool surface enforces. The grounding-then-querying workflow either succeeds (in which case the answer is largely correct) or fails (in which case the answer is largely wrong). There is little middle ground in which the system produces a partially correct answer through partial grounding.

The execution-failure distribution further supports the design argument. Of the approximately 41 questions on which Claude failed to produce executable SPARQL (the 8.5% complement of the 91.5% success rate in Table 2), 35 were blocked by the safety linter for containing identifiers not produced by a grounding tool in the same session. Without the linter, those 35 queries would have been sent to the endpoint with hallucinated QIDs and would either have returned wrong answers silently or have failed opaquely. The linter converts what would have been silent failures into observable refusals. The following result captures the operational value of the safety layer: roughly 7% of all Claude attempts on QAWiki include an ungrounded identifier even when the prompt explicitly instructs grounding-first, and every one of those attempts is caught.

In summary, GraSP-MCP enables the LLM to answer relation-heavy questions over Wikidata at meaningful quality, with failures concentrated in semantic grounding rather than query execution. Eliminating the remaining failures requires semantic improvements that go beyond connector design.

5.4 Results on the Rhea Database

Rhea is a curated database of biochemical reactions maintained by the Swiss Institute of Bioinformatics [3]. It contains expert-annotated descriptions of approximately 16,000 reactions, each

Table 4: Tools provided by the GraSP-MCP server for the Rhea endpoint. Each tool represents a query template.

Tool	Description	Justification
`reactions_by_ec`	Returns reactions associated with a given Enzyme Commission number.	Encapsulates a recurring Rhea query pattern; the LLM does not need to know how Rhea encodes the EC-reaction relation.
`reactions_producing_product_from_substrate_names`	Finds reactions connecting a named substrate to a named product.	Encodes the correct sides for the join, which an unconstrained LLM frequently writes in the wrong direction.
`find_reaction_by_equation_text`	Searches reactions by free-text equation.	Provides a search-by-content path when no identifier is available to ground.
`children_of_reaction`	Returns child reactions in the Rhea reaction hierarchy.	Encodes the hierarchy traversal pattern, which depends on Rhea-specific predicates.
`run_sparql`	Executes freeform SPARQL against the Rhea endpoint with the same lint and dry-run discipline applied to Wikidata.	Provides an escape hatch for queries not covered by templates.

Table 5: Three examples from the Rhea evaluation, illustrating the qualitative difference between baseline and MCP responses.

Question	Baseline Response	MCP Response
EC-number lookup. "Which reactions link to EC 1.11.1.6 (catalase)?"	Plausible catalase explanation with an extra reaction included from parametric memory.	Returns the actual approved Rhea mapping: `RHEA:20309`, `2 H2O2 = O2 + 2 H2O`.
Substrate-to-product. "Which reactions interconvert pyruvate and lactate?"	Lists the main lactate dehydrogenase reactions from general knowledge.	Returns a broader set of actual Rhea reactions, including NAD/NADP, cytochrome, quinone, and FAD-coupled variants.
Cross-reference traversal. "Which reactions produce citrate and have a Reactome cross-reference?"	Says it cannot reliably confirm without querying Rhea and offers likely candidates.	Returns three Rhea reactions with Reactome cross-references: `RHEA:16845`, `RHEA:33183`, and `RHEA:72483`.

linked to participating chemical entities through ChEBI [10], to catalyzing enzymes through Enzyme Commission numbers and UniProt, and to broader biochemical pathways through Rhea's own hierarchy of related reactions. Rhea exposes its contents through a public SPARQL endpoint.[6] We evaluate GraSP-MCP on Rhea for three reasons that together stress-test the design assumptions made for the Wikidata experiments.

- **Schema complexity.** A Rhea reaction is not a single triple. Each reaction has sides (substrate versus product), directionality (forward versus reverse for transport reactions), enzyme mappings via Enzyme Commission numbers, and cross-references to ChEBI, UniProt, and KEGG. A correct query must navigate these dimensions explicitly. Wikidata often requires single-triple-pattern queries; Rhea rarely does. Thus, the LLM cannot succeed by guessing a simple shape.
- **Out-of-distribution vocabulary.** Since Wikidata stores general knowledge about a broad set of domains, LLMs encounter Wikidata entities frequently during pretraining and develop reasonably accurate parametric associations between common names and QIDs. Rhea reaction identifiers (for example, `RHEA:11628`) and ChEBI chemical identifiers do not occur as frequently in pretraining data. The LLM cannot rely on memorization to answer Rhea questions. It must use the tools or fail.
- **Absence of a searchable label index.** Wikidata exposes a full-text search API over entity labels through the `wbsearchentities` endpoint, which makes name-based grounding tractable: a tool can take "Albert Einstein" and return `Q937`. Rhea exposes no analogous search-by-label index at the endpoint level. The composable-primitives design used for Wikidata (Section 3) is therefore not applicable to Rhea, and a different tool surface is required.

[6] https://sparql.rhea-db.org/sparql

The tool surface for Rhea exposes parameterized templates rather than composable primitives. Where the Wikidata tools require the LLM to assemble a SPARQL query from grounded identifiers, the Rhea tools each correspond to a recurring query pattern parameterized by surface text. The LLM selects a template by name and supplies its arguments. Table 4 lists the Rhea tools.

The general design principle that distinguishes the two tool surfaces is the following: **when graph entities are reachable by surface-form search, expose composable primitives and let the LLM assemble; when they are not, expose templates parameterized by surface text.** The correct level of abstraction is the level at which the LLM's parametric knowledge can ground reliably. For Wikidata, that level is the identifier (the LLM can name the entity in English, the tool returns the QID, and the LLM composes from there). For Rhea, that level is the query pattern (the LLM can name the kind of question being asked, the tool runs the correct SPARQL, and the LLM interprets the results).

Evaluation protocol. We evaluate the Rhea tool surface on a set of 50 questions written to cover the main Rhea query patterns: reaction lookup by EC number, substrate-to-product navigation, equation-text search, cross-reference traversal, and reaction-hierarchy traversal. Rhea does not provide gold answer sets for these questions, so a fully quantitative evaluation against ground truth is not possible. To approximate one, we use a pairwise judge protocol: for each question, the LLM (Opus 4.7) produces two responses, one with no MCP connection and one with GraSP-MCP connected to the Rhea endpoint. Another LLM, GPT-5, is used as an independent judge that reads both responses blind, in randomized A/B order, and selects the better response based on four criteria: factual accuracy (correct reaction, direction, participants), specificity (equations, stoichiometry, sides), verifiability of Rhea identifiers, and usefulness for someone querying the Rhea database. The judge can also return "both bad" when neither response is satisfactory. Table 5 shows

Table 6: Proxy metrics for quality on the 50 Rhea questions.

Metric	Baseline	MCP	Ratio
Mean Rhea IDs per answer	6.72	112.58	16.75×
Median Rhea IDs per answer	5.0	14.5	2.90×
Mean answer length (chars)	1,223	3,758	3.1×

three examples from the Rhea evaluation questions, illustrating the qualitative difference between baseline and MCP responses.

Findings. The judge preferred the MCP-connected response in 43 of 50 cases (86%), the no-MCP baseline in 6 cases (12%), and rated both responses unsatisfactory in 1 case (2%). The base model produces fluent, plausible-sounding biochemistry that is rarely specific to Rhea: it discusses reaction types in general terms, names chemical entities by common names, and avoids committing to specific identifiers. When it does commit to a Rhea or ChEBI identifier, the identifier is often invented and does not resolve. The MCP-connected model produces responses that name specific Rhea reactions by their actual identifiers, quote reaction equations from the database, and cite the cross-references recorded by Rhea. The judge-preference pattern is not uniform across question types. On open lookups, where the question asks about a reaction or class without specifying a particular Rhea identifier, the MCP-connected response is preferred almost universally. On identifier-specific questions, where the question asks about a given Rhea ID, the split is closer to 56% MCP / 44% baseline. The MCP failures in this category are factual errors: the tool path occasionally retrieves a related reaction with the wrong directionality or returns more identifiers than asked, and the judge correctly penalizes these errors. Parametric memory is competitive when the LLM is given the identifier and just needs to describe the reaction accurately. Table 6 reports objective metrics that are computed automatically and serve as proxy metrics for quality, complementing the subjective evaluation by the LLM judge. These metrics measure the number of Rhea IDs in the returned answer and the size of this answer in characters, under the assumption that more IDs or a larger answer are better, since the answer with more IDs is likely to contain the correct result. The MCP-connected response includes roughly 17 times as many verifiable Rhea identifiers on average, and produces answers roughly 3 times longer. These proxy metrics are objective measurements; they do not establish correctness on their own (the judge data above shows that more identifiers is not the same as a correct answer), but they confirm that the MCP path produces responses that are quantitatively more specific to Rhea content.

The judge-based evaluation reported here uses an LLM as a proxy for human assessment and should be treated as an approximation rather than ground truth, but the headline result confirms that the template-based tool surface enables GraSP-MCP to operate on a knowledge graph for which the composable-primitives design would not work.

6 RELATED WORK

MCP [13] is a recent open specification, and work on using it with SPARQL is still nascent. One relevant system is TogoMCP [11], which provides MCP-based SPARQL access to life-science RDF databases. TogoMCP places the design weight on curated context: each target database is paired with a hand-curated YAML file supplying structural and semantic information at query time. We place the design weight on the connector itself, studying how the choice of tool surface and prompt design affects answer quality across two structurally different knowledge graphs. Agentic SPARQL [6] uses MCP for federated querying across multiple endpoints, and a recent systematization [7] treats MCP primitives as a threat surface, complementary to our treatment of them as a design surface.

Knowledge graphs and SPARQL have a long history in tabular data analysis. The SemTab Challenge [8, 9] has, since 2019, benchmarked systems that match table cells to Wikidata entities and types; its 2024 and 2025 editions introduced LLM-specific tracks where the most successful systems combine LLM judgment with structured retrieval against the target knowledge graph. Our work targets the underlying capability such systems require: a reliable way for an LLM to query SPARQL endpoints on demand.

A substantial body of work studies translating natural-language questions directly into SPARQL queries [2, 4, 12, 16]. These approaches treat SPARQL generation as a translation problem to be solved with better prompting and retrieved examples. We begin from a different assumption: even with good prompting, a single-step text-to-SPARQL pipeline cannot reliably resolve entity identifiers against an endpoint whose contents the LLM has only partial parametric knowledge of. Recent work on protoknowledge [15] characterizes precisely the parametric-knowledge gaps that motivate our grounding-first design. We evaluate on QAWiki [14], a recent multilingual benchmark of hand-curated question-query pairs over Wikidata with answer-type annotations. The WikiKGQA challenge [19] builds on QAWiki and represents an emerging community effort in the same area. Older benchmarks include SimpleQuestionsWikidata [5] and the QALD series (e.g., QALD-10 [17]).

7 CONCLUSION

We presented GraSP-MCP, an MCP connector that enables an LLM to query SPARQL endpoints. Our main finding is that the exposed MCP tools must be carefully designed for the connector to be useful to an LLM. The tools must go beyond the transport layer and help the LLM in result grounding. Specifically, the tool surface must decompose query construction into grounding, schema inspection, and validated execution rather than exposing a single `run_sparql` primitive. In addition to carefully designing the MCP tools, we find that prompting the LLM in a smart way to guide it through the workflow of correctly using these tools adds secondary benefit. Our experiments with questions from the QAWiki benchmark show that Claude Opus 4.7 using GraSP-MCP to connect to Wikidata produces SPARQL that executes successfully for 91.5% of the questions and matches the gold answer exactly for 46.8% of them. The gap between these two numbers points to several directions for future work. First, a per-component error analysis, measuring the accuracy of entity grounding, property grounding, and relation-direction selection separately, would show where in the pipeline the remaining failures concentrate. Second, the grounding-and-synthesis workflow could be made iterative, with the LLM, or several LLMs, critiquing and refining a query across multiple passes rather than committing in one. Third, the Wikidata and Rhea connectors apply the same principles through generic and domain-specific tool surfaces, respectively; understanding the tradeoffs between these two styles, and whether hybrids of them help, would guide the design of connectors for new knowledge graphs.

REFERENCES

[1] Wiem Baazouzi, Marouen Kachroudi, and Sami Faïz. 2023. Kepler-aSI at SemTab 2023. In *Proc. Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab 2023)*. 85–91.

[2] Debayan Banerjee, Pranav Ajit Nair, Jivat Neet Kaur, Ricardo Usbeck, and Chris Biemann. 2022. Modern Baselines for SPARQL Semantic Parsing. In *Proc. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*. 2260–2265. https://doi.org/10.1145/3477495.3531841

[3] Parit Bansal, Anne Morgat, Kristian B. Axelsen, Venkatesh Muthukrishnan, Elisabeth Coudert, Lucila Aimo, Nevila Hyka-Nouspikel, Elisabeth Gasteiger, Arnaud Kerhornou, Teresa Batista Neto, Monica Pozzato, Marie-Claude Blatter, Alex Ignatchenko, Nicole Redaschi, and Alan Bridge. 2022. Rhea, the reaction knowledgebase in 2022. *Nucleic Acids Research* 50, D1 (2022), D693–D700. https://doi.org/10.1093/nar/gkab1016

[4] Jacopo D'Abramo, Andrea Zugarini, and Paolo Torroni. 2025. 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, Albuquerque, New Mexico, USA, 66–80. https://doi.org/10.18653/v1/2025.knowledgenlp-1.5

[5] Dennis Diefenbach, Thomas Pellissier Tanon, Kamal Singh, and Pierre Maret. 2017. Question Answering Benchmarks for Wikidata. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks*. https://ceur-ws.org/Vol-1963/paper555.pdf

[6] Daniel Dobriy, Frederik Bauer, Amr Azzam, Debayan Banerjee, and Axel Polleres. 2026. Agentic SPARQL: Evaluating SPARQL-MCP-powered Intelligent Agents on the Federated KGQA Benchmark. *arXiv preprint arXiv:2603.06582* (2026). arXiv:2603.06582 https://arxiv.org/abs/2603.06582

[7] Shiva Gaire, Srijan Gyawali, Saroj Mishra, Suman Niroula, Dilip Thakur, and Umesh Yadav. 2025. Systematization of Knowledge: Security and Safety in the Model Context Protocol Ecosystem. *arXiv preprint arXiv:2512.08290* (2025). arXiv:2512.08290 https://arxiv.org/abs/2512.08290

[8] Oktie Hassanzadeh, Nora Abdelmageed, Marco Cremaschi, Vincenzo Cutrona, Fabio D'Adda, Vasilis Efthymiou, Benno Kruit, Elita Lobo, Nandana Mihindukulasooriya, and Nhan H. Pham. 2024. Results of SemTab 2024. In *Proc. Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab 2024)*. 1–11. https://ceur-ws.org/Vol-3889/paper0.pdf

[9] Oktie Hassanzadeh, Marco Cremaschi, Fabio D'Adda, Fidel Jiomekong Azanzi, Jean Petit Bikim, and Ernesto Jiménez-Ruiz. 2025. Results of SemTab 2025. In *Proceedings of the 20th International Workshop on Ontology Matching co-located with the 24th International Semantic Web Conference (ISWC 2025)*. 216–220. https://ceur-ws.org/Vol-4144/om2025-SemTab-paper0.pdf

[10] Janna Hastings, Gareth Owen, Adriano Dekker, Marcus Ennis, Namrata Kale, Venkatesh Muthukrishnan, Steve Turner, Neil Swainston, Pedro Mendes, and Christoph Steinbeck. 2016. ChEBI in 2016: Improved services and an expanding collection of metabolites. *Nucleic Acids Research* 44, D1 (2016), D1214–D1219. https://doi.org/10.1093/nar/gkv1031

[11] Akira R Kinjo, Yasunori Yamamoto, Samuel Bustamante-Larriet, Jose-Emilio Labra-Gayo, and Takatomo Fujisawa. 2026. TogoMCP: Natural Language Querying of Life-Science Knowledge Graphs via Schema-Guided LLMs and the Model Context Protocol. *bioRxiv* (2026). https://doi.org/10.64898/2026.03.19.713030

[12] Liubov Kovriguina, Roman Teucher, Daniil Radyush, and Dmitry Mouromtsev. 2023. SPARQLGEN: One-Shot Prompt-based Approach for SPARQL Query Generation. In *Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems co-located with 19th International Conference on Semantic Systems (SEMANTiCS 2023)*. https://ceur-ws.org/Vol-3526/paper-08.pdf

[13] Model Context Protocol Contributors. 2025. Model Context Protocol Specification. https://modelcontextprotocol.io/specification/2025-11-25 Accessed: 2026-05-13.

[14] Alberto Moya Loustaunau and Aidan Hogan. 2025. QAWiki: A Knowledge Graph Question Answering & SPARQL Query Generation Dataset for Wikidata. In *Proceedings of the Wikidata Workshop 2025*. https://ceur-ws.org/Vol-4108/paper12.pdf

[15] Federico Ranaldi, Andrea Zugarini, Leonardo Ranaldi, and Fabio Massimo Zanzotto. 2025. Protoknowledge Shapes Behaviour of LLMs in Downstream Tasks: Memorization and Generalization with Knowledge Graphs. *CoRR* abs/2505.15501 (2025). https://doi.org/10.48550/ARXIV.2505.15501 arXiv:2505.15501

[16] Tilahun Abedissa Taffa and Ricardo Usbeck. 2023. Leveraging LLMs in Scholarly Knowledge Graph Question Answering. *CoRR* abs/2311.09841 (2023). https://doi.org/10.48550/ARXIV.2311.09841 arXiv:2311.09841

[17] Ricardo Usbeck, Xi Yan, Aleksandr Perevalov, Longquan Jiang, Julius Schulz, Angelie Kraft, Cedric Möller, Junbo Huang, Jan Reineke, Axel-Cyrille Ngonga Ngomo, Muhammad Saleem, and Andreas Both. 2024. QALD-10 – The 10th Challenge on Question Answering over Linked Data: Shifting from DBpedia to Wikidata as a KG for KGQA. *Semantic Web* 15, 6 (2024), 2193–2207.

[18] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledgebase. *Comm. of the ACM* 57, 10 (2014), 78–85.

[19] WikiKGQA Challenge Organizers. 2026. WikiKGQA: Wiki-Based Knowledge Graph Question Answering Challenge. https://wikikgqa.org/ Co-located with the 25th International Semantic Web Conference (ISWC 2026); accessed 2026-05-13.