rudofMCP: A Model Context Protocol Server Exposing Knowledge Graph Engineering Capabilities to Large Language Models

Samuel Bustamante-Larriet[1,*], Álvaro García-Fernández[1] and Jose-Emilio Labra-Gayo[1]

[1]University of Oviedo, Spain

Abstract

The convergence between Large Language Models (LLMs) and Knowledge Graphs (KGs) constitutes a promising research frontier, constrained by the limited ability of LLMs to reason directly over the structure of KGs. Existing *Model Context Protocol* (MCP) servers for Knowledge Graph Engineering only partially address this problem, as they specialize in a single capability. This work presents rudofMCP, an MCP server that exposes the main capabilities of the rudof library (reading, writing and converting between RDF formats; node inspection; SPARQL query execution; ShEx, SHACL validation) to any client that supports the protocol, allowing models to delegate Knowledge Graph Engineering operations to a specialized system while leaving an explicit, inspectable audit trail of every invoked operation and its result. The effect of the server on model performance is measured through an adaptation of the LLM-KG-Bench 3.0 *benchmark*. The adaptation adds a new agentic MCP connector and a new ShEx validation task. A single model is evaluated across two arms (without and with access to the MCP server) and the results are analyzed with non-parametric rank tests and Cliff's delta as the effect size. The results show that the effect is task-dependent: large improvements when the task admits delegating the work to a formal operation, deterioration when conciseness is required, and, in all cases, an additional cost in latency and token consumption.

Keywords

Large Language Models, Knowledge Graphs, Model Context Protocol, RDF, SPARQL, ShEx, SHACL

1. Introduction

The convergence between Large Language Models (LLMs) and Knowledge Graphs (KGs) is a promising research area, where the natural language processing capabilities of the former meet the precision and structure of the formal knowledge represented in the latter [1]. This integration, however, faces obstacles: LLMs suffer from hallucinations and struggle to understand the structure of KGs directly, while KGs provide explicit knowledge and interpretable reasoning but are difficult to build, inherently incomplete and lack language-understanding capabilities on their own [1].

Empirical work documents these shortcomings. Systematic studies evaluate forty-one contemporary models with the LLM-KG-Bench benchmark [2, 3, 4, 5, 6]. They reveal that, while syntactic repair of RDF graphs is within reach of most models, elementary semantic tasks (such as counting the incoming relations of a node) produce bimodal distributions in which models tend to either succeed or fail systematically. Regarding SPARQL, the syntax is manageable for almost all of them, but semantic correctness is consistent only in a reduced group of large models. In addition, models exhibit a cross-cutting bias (dependent on model family and task) toward a specific RDF serialization format, without any universally preferred format [2]. To bridge these gaps, recent work materializes the union between LLMs and KGs through the *Model Context Protocol* (MCP) [7], an open protocol developed by Anthropic

ELMKE 2026: The Fourth International Workshop on Evaluation of Language Models in Knowledge Engineering, co-located with ISWC 2026, October 25–29, 2026, Bari, Italy

*Corresponding author.

bustamantesamuel@uniovi.es (S. Bustamante-Larriet); garciafalvaro@uniovi.es (Á. García-Fernández); labra@uniovi.es (J. Labra-Gayo)

https://github.com/samuel-bustamante (S. Bustamante-Larriet); https://github.com/algarfer (Á. García-Fernández); https://labra.weso.es/ (J. Labra-Gayo)

0009-0005-8631-2682 (S. Bustamante-Larriet); 0009-0008-9390-6210 (Á. García-Fernández); 0000-0001-8907-5348 (J. Labra-Gayo)

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

that standardizes the communication between LLMs and external systems [8, 9]. However, these efforts share a common pattern: they specialize in a single capability (SPARQL) rather than covering the broader spectrum of tasks that comprise Knowledge Graph Engineering (RDF, ShEx, SHACL, etc.).

Faced with these limitations, this work contributes with the implementation of rudofMCP, an MCP server that exposes the main capabilities of the rudof library [10, 11]. rudof is an open-source library, written in Rust, that implements the main operations of Knowledge Graph Engineering: reading, writing and converting between RDF formats; node inspection; SPARQL query execution; ShEx, SHACL validation; among others. Using MCP as an interoperability layer, the server allows an LLM to delegate each of those operations to rudof instead of simulating their semantics from the textual form of the graph, thereby externalizing structural reasoning to a specialized system while the model retains its role of orchestrating sequences of operations and interpreting results. This delegation also offers an inspectable audit trail of the external operations behind each answer: the sequence of invoked tools and the results each of them returns are available *a posteriori*. This is not interpretability of the model's internal reasoning, but a concrete trace of the formal operations it relied on.

This work is guided by the following research question: given the same LLM, does the availability of an MCP server that exposes the capabilities of the rudof library appreciably modify the model's performance on tasks related to RDF, SPARQL and ShEx? To answer it, we adapt the LLM-KG-Bench 3.0 benchmark [2], implementing a new connector able to manage the agentic loop that MCP requires and a new validation task against ShEx schemas. The evaluation compares the performance of the same model in two arms (*baseline* without access to rudof and *mcp* with access via the proposed server) and analyzes the results with non-parametric rank tests and the Cliff's delta effect size.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the rudofMCP server. Section 4 presents the experimental design and the statistical analysis plan. Section 5 reports the results, which are interpreted in Section 6. Section 7 concludes and outlines future work.

2. Related Work

S. Pan *et al.* [1] propose a roadmap for the integration of LLMs and KGs through three complementary frameworks: (1) KG-enhanced LLMs, which incorporate graphs during pre-training or inference, or to increase interpretability; (2) LLM-augmented KGs, which use language models for embedding, completion, construction, text generation and question answering; and (3) LLMs and KGs in synergy, where both technologies collaborate on equal footing for the mutual improvement of knowledge representation and bidirectional reasoning. The authors characterize the complementary strengths of each paradigm. LLMs excel at natural language processing, but suffer from hallucinations and limited access to structured factual knowledge, while KGs provide explicit knowledge and interpretable reasoning, though they are difficult to build, inherently incomplete and lack language-understanding capabilities. Among other pending challenges, they identify that of endowing LLMs with the ability to directly understand the structure of KGs, which would enable more effective reasoning over structured knowledge.

L.-P. Meyer *et al.* [2] present LLM-KG-Bench 3.0, a framework for the automatic and systematic evaluation of the capabilities of LLMs in Knowledge Graph Engineering tasks. This third iteration, also presented in condensed form at ESWC 2025 [3], culminates a trajectory initiated in 2023 with the first proposal of the benchmark [6] and extended successively to the creation and understanding of RDF [5] and, later, to SPARQL capabilities [4]. The framework is articulated around a *Prompt-Answer-Evaluate* cycle with dialogued correction rounds, connectors for multiple LLMs and an extensible catalog of tasks grouped into two large families: RDF-related tasks (comprehension, syntactic repair and graph construction in different serialization formats) and SPARQL-related tasks (generation, execution and syntactic repair of SELECT queries). Using this framework, the authors execute twenty-six tasks over forty-one contemporary LLMs, both open (Llama, Qwen, Phi, DeepSeek and OpenCoder families) and proprietary (Claude, GPT and Gemini). Regarding RDF tasks, syntactic repair is within reach of most

models, while elementary semantic tasks (such as counting the incoming relations of a node) produce a bimodal distribution in which models tend to either succeed or fail systematically. Regarding SPARQL, syntax is within reach of almost all models, but only a reduced group of large models achieves semantic correctness consistently. A transversal finding is the presence of biases (dependent on the model family and the task) toward a specific RDF serialization format.

On the practical implementation side, works exist that materialize this LLM/KG convergence through MCP. For example, A. R. Kinjo *et al.* [8] present TogoMCP, an MCP server oriented to natural-language querying of life-science KGs. TogoMCP guides the LLM through a context file (the *MIE file*) and a workflow that separates entity resolution from SPARQL generation, aiming to improve the precision of the queries generated by the model. In a complementary direction, R. Taelman and J. De Smet [9] propose Comunica MCP SPARQL, a server that provides AI agents with SPARQL access to decentralized KGs, improving the accuracy of their answers by grounding them in verifiable external data. Both works, representative of an expanding ecosystem, share the same pattern: they specialize in a single capability (SPARQL) rather than covering the broader spectrum of tasks that comprise Knowledge Graph Engineering.

3. rudofMCP: System Description

This section describes rudofMCP, the system developed in this work. The server exposes the capabilities of the rudof library to any client that supports MCP, so that an LLM can delegate Knowledge Graph Engineering tasks to it.

3.1. Architecture

The server is distributed as a Rust crate (`rudof_mcp`[1]) and is compiled as part of the main rudof binary (`rudof_cli`[2]), being accessible through the subcommand `rudof mcp`. Internally, `rudof_mcp` relies on two fundamental dependencies: `rudof_lib`[3], the rudof API, which exposes RDF, ShEx, SHACL and SPARQL capabilities, among others, and the official Rust SDK for MCP, `rmcp`[4], which provides the protocol infrastructure.

The `rudof_mcp` crate follows the Single Responsibility Principle at three levels of granularity so that adding a new tool, prompt or resource affects a single point of the code: modules (`server` concerns the transport layer, `service` concerns the response logic), submodules (each covers one transport or one MCP capability), and files (each groups the elements of one functional domain: RDF data management, node inspection, SPARQL query, ShEx, SHACL).

Above this principle, the architecture applies the *Adapter* pattern (`rudof_mcp` adapts `rudof_lib` to the `ServerHandler` trait of `rmcp`), the *Strategy* pattern for the interchangeable transports (Stdio and Streamable HTTP), the *Factory Method* pattern in the HTTP transport (the SDK receives a factory that produces a fresh service instance per MCP session, isolating simultaneous clients — see Section 3.4), and a two-level layered architecture with a unidirectional dependency from transport to service.

3.2. Supported Transports

rudofMCP supports the two transports defined by the current MCP specification (version 2025-11-25). The Stdio transport allows a local client (e.g., Claude Desktop) to launch rudofMCP as a subprocess and exchange JSON-RPC messages through stdin and stdout, delimited by line breaks. The Streamable HTTP transport exposes the server on a configurable port and path; exchanges use POST over HTTP and the server responds, depending on the case, with a single JSON object or with a *Server-Sent Events* (SSE) stream that enables streaming messages and server notifications. Session management is done

[1] https://crates.io/crates/rudof_mcp
[2] https://crates.io/crates/rudof_cli
[3] https://crates.io/crates/rudof_lib
[4] https://github.com/modelcontextprotocol/rust-sdk

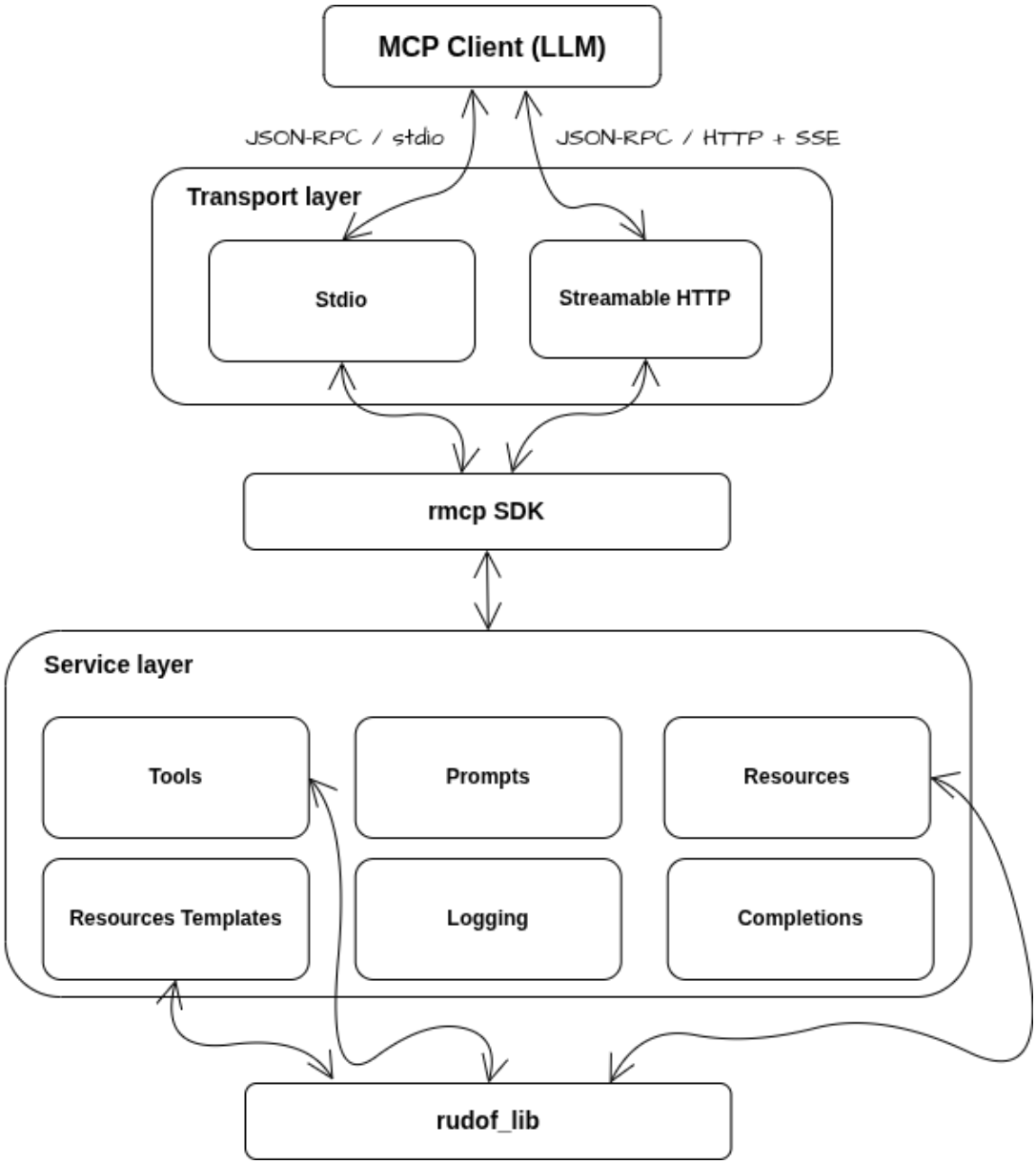

Figure 1: Architecture of rudofMCP. The transport layer selects between Stdio and Streamable HTTP and delivers JSON-RPC messages to the service layer, which dispatches tools, prompts and resources over `rudof_lib`.

through the *MCP-Session-Id* header and clients can explicitly terminate a session by sending a DELETE request to the server path. The Streamable HTTP layer incorporates several defensive mechanisms: those required or recommended by the specification (validation of the *Origin* header to mitigate DNS rebinding attacks, validation of the protocol version declared by the client and binding to *localhost* by default) and an additional configurable allow-list of IP networks in CIDR notation.

3.3. Exposed MCP Capabilities

MCP organizes the interaction between client and server around a small set of capabilities: *tools*, the main entry point for the model to invoke operations over data; *prompts*, guided templates for recurring workflows; *resources* and *resource templates*, which let the model read server state idempotently; *logging*, which sends trace notifications to the client with severity-level filtering per RFC 5424[5]; and *completions*, which suggest values for prompt and resource-template arguments. Table 1 summarizes the ten tools

[5] datatracker.ietf.org/doc/html/rfc5424#section-6.2.1

exposed by the server, grouped functionally into five blocks: RDF data management, node inspection, SPARQL querying, ShEx validation, and SHACL validation. Each tool declares, along with its description, a set of standardized MCP annotations (*read-only*, *destructive*, *idempotent*, *open-world*) that the client can use to reason about the effects of an invocation, as well as JSON input and output schemas.

Table 1
Tools exposed by rudofMCP.

Tool	Description
`load_rdf_data_from_sources`	Loads RDF into server memory from URLs, local files, plain text, or SPARQL endpoints. Loads can be cumulative.
`export_rdf_data`	Serializes the loaded graph in any of the supported formats (Turtle, N-Triples, RDF/XML, JSON-LD, TriG, N-Quads, N3).
`export_plantuml`	Generates a PlantUML representation of the graph.
`export_image`	Renders the graph as an SVG or PNG image from the PlantUML representation. Requires PlantUML to be accessible.
`node_info`	Inspects a node of the graph and returns its outgoing and incoming arcs.
`execute_sparql_query`	Executes SPARQL queries over the loaded graph.
`validate_shex`	Validates the loaded graph against a ShEx schema, using a shape map or a node-shape pair.
`check_shex`	Checks that a ShEx schema is syntactically correct and free of negative cycles.
`show_shex`	Serializes a ShEx schema in any of the supported formats (ShExC, ShExJ) and produces statistics and shape dependency analysis.
`validate_shacl`	Validates the loaded graph against a SHACL schema.

The server also exposes three prompt templates that guide the model through recurring workflows (`explore_rdf_node`, `analyze_rdf_data` and `validation_guide`), a set of resources (the currently loaded graph at `rudof://current-data` and a template `rudof://current-data/{format}` to obtain it in any supported format), and descriptive resources under `rudof://formats/` that enumerate the formats and modes supported by each operation. Logging notifications are emitted through the `notifications/message` mechanism with rate-limiting, and completions provide contextual suggestions for prompt and resource-template arguments.

3.4. State Management

The server maintains a single in-memory RDF store, encapsulated in a shared `rudof` instance across all invocations. The only tool that modifies the state (`load_rdf_data_from_sources`) accumulates the loaded triples, while read-only tools and resources operate over the state in force at the time of the request.

The rudof library is not designed for concurrent use: its operations run sequentially over the engine and assume exclusive access to the graph throughout their duration. This restriction propagates to the server, but its actual scope is bounded by two factors. First, isolation is per session: on the Streamable HTTP transport, each MCP session receives its own independent instance of `rudof`, so several simultaneously connected clients do not compete with each other. On the Stdio transport, where there is a single client per process, isolation is trivial. Sequential execution therefore only manifests within a single session. Second, this scenario is uncommon in practice with an LLM as a client, because the model issues tool invocations one at a time and waits for the result of each before continuing.

To support ephemeral container deployments (as in the *Docker MCP Registry*), the server incorporates an optional persistence mechanism: on state changes it serializes the graph in N-Triples and writes it inside a JSON file in a configurable directory (by default `/app/state/data.json`); on startup, it attempts to recover that state and reconstruct the graph in memory. If the state directory does not exist, the mechanism is transparently disabled.

4. Methodology

This section describes the methodology followed. First, we state the research question, then justify the choice of the evaluation platform (benchmark) and describe its architecture; the central block presents the two-arm experimental design, the adaptations made to the benchmark and the criteria for the choice of tasks and model; finally, we describe the statistical analysis. Operational details of the code are available in the public repository[6].

4.1. Research Question

Given the same LLM, does the availability of an MCP server that exposes the capabilities of the rudof library appreciably modify the model's performance on tasks related to RDF, SPARQL and ShEx?

4.2. Evaluation Platform: LLM-KG-Bench 3.0

The study reuses LLM-KG-Bench 3.0[7] [2, 3] as the evaluation infrastructure. Three considerations support this decision. First, it is an active, open project specifically oriented to measuring the capabilities of LLMs in Knowledge Graph Engineering tasks. Second, it offers a catalog of already implemented tasks, with their metrics and data. And third, contributing an extension to the benchmark enriches its coverage. LLM-KG-Bench 3.0 is articulated into three main components: tasks that encapsulate a capability to evaluate, with a prompt template and a set of metrics; model connectors that translate the abstract interface (deliver a prompt, receive an answer) into concrete calls to a provider; and a runner that orchestrates both, executing a matrix declared in a configuration file. This separation of concerns is what enabled the present study: adding a new task or a new connector reduces to implementing the corresponding interface.

4.3. Experimental Design

The experiment follows a two-arm design that differs in tool access. In the control arm (*baseline*) the model receives the prompt of each task and produces its answer in a single API call, with no possibility of consulting external tools. In the experimental arm (*mcp*) the model has access to a local MCP server that exposes the capabilities of rudof and can invoke them as many times as it deems appropriate in an agentic loop that ends when the model itself decides to answer (by emitting a text-only turn, not on any correctness check), capped at twenty turns. Two operational adjustments accompany tool access and are corollaries of the setup rather than independent manipulations: a system prompt that instructs the model to delegate to the tools, and a larger output-token budget per API call (8192 vs. 4096 in the baseline) to accommodate the multi-turn loop. The comparison is therefore between the deployed MCP configuration and a single-shot no-tools baseline. Everything else (model, catalog of tasks, task parameters and user prompts) is shared across arms.

Adaptations to LLM-KG-Bench 3.0. The benchmark was not ready, in its original state, to answer the research question: it lacked a connector capable of talking to an MCP server, and a graph-validation task (ShEx; SHACL is left for future work). Two adaptations were therefore necessary. The MCP connector (`ModelClaudeMCP`) launches a local instance of the rudof server over stdio, queries its catalog of available tools, injects them into each API call together with a system prompt reminding the model to rely on them instead of reasoning from the graph text, and aggregates the token consumption of all turns and the sequence of invoked tools. The loop terminates when the model itself emits a text-only response (`end_turn`); the decision to stop rests with the model and is not conditioned on the correctness of the answer. A twenty-turn cap bounds the loop; if reached, a final tool-less call forces a textual response. The ShEx validation task (`ShExValidationStatic`) contains twenty items distilled

[6] https://github.com/rudof-project/LLM-KG-Bench-rudof
[7] https://github.com/AKSW/LLM-KG-Bench

from the official ShEx test suite[8], deliberately choosing cases that test the limits of the language; a more detailed description of the task and its items is available in the public repository[9]. The primary metric is a binary *accuracy* per item, coherent with the nature of the problem: ShEx conformance does not admit partial truths.

Catalog of Tasks Evaluated. The benchmark runs over five task families summarized in Table 2: four inherited from LLM-KG-Bench 3.0 and the new *ShExValidationStatic*. The inherited four were picked from the sixteen available in the catalog to cover the capability dimensions rudofMCP most directly targets: RDF comprehension (*RdfConnectionExplainStatic*, *RdfFriendCount*), natural-language question answering over RDF (*Text2AnswerListOrganizational*) and SPARQL execution over RDF (*Sparql2AnswerListOrganizational*). For each family we also fixed *a priori* a single primary metric that guides the confirmatory analysis, while the remaining sub-metrics feed the exploratory analysis of Section 4.4. The *pairing* column records whether the i-th run of each arm resolves the same item across arms (pairing preserves within-item variance) or not.

Table 2
Selected task families, variants, primary metric and pairing regime.

Family	Origin	Variants	Primary metric	Pairing
`RdfConnectionExplainStatic`	inherited	nt, xml	`listF1`	unpaired
`RdfFriendCount`	inherited	turtle-1, turtle-2, jsonld-1, jsonld-2	`f1`	unpaired
`Text2AnswerListOrganizational`	inherited	turtle, jsonld	`f1`	paired
`Sparql2AnswerListOrganizational`	inherited	turtle, jsonld	`f1`	paired
`ShExValidationStatic`	new	default	`accuracy`	paired

- **RdfConnectionExplainStatic.** The prompt presents a static organizational graph (two persons in different departments of the same organization). The model must identify the shortest path between them, ignoring `rdf:type` arcs, and return the IRI sequence one per line. The two variants serialize the same graph in N-Triples and RDF/XML. The graph is the same in every one of the twenty runs of each cell: the model receives an identical prompt each time and the only source of variability is its own randomness, so what this task measures is how consistently the model answers the same question, not how well it generalizes to other path-explanation graphs. Observations are treated as independent (all runs receive the same prompt).
- **RdfFriendCount.** The prompt presents a programmatically generated graph of `foaf:Person` resources connected by `foaf:knows`. Each person points to two others; a special node receives additional arcs and becomes the individual with the most incoming arcs. The model must identify that node and return only its IRI. Four variants combine two serializations (Turtle and JSON-LD) with two difficulty levels via the *specialAddKnowsCount* parameter (1 or 2 extra arcs). Each of the twenty runs regenerates a random graph, so observations are unpaired.
- **Text2AnswerListOrganizational** and **Sparql2AnswerListOrganizational.** Both present the same organizational graph plus, respectively, a natural-language question or a SPARQL SELECT query; the model must return the set of entities that answer it, one per line, using abbreviated prefixes. The two variants serialize the graph in Turtle and JSON-LD. The set of questions/queries is fixed and traversed in the same deterministic order across arms, so observations are paired.
- **ShExValidationStatic.** As described above. The runner iterates the twenty items in a deterministic order via an index, so observations are paired.

Model. This study does not aim to explore the space of models but to serve as a proof of concept of whether the rudofMCP server appreciably affects a model's performance on Knowledge

[8] https://github.com/shexSpec/shexTest/
[9] https://github.com/rudof-project/LLM-KG-Bench-rudof/tree/main/LlmKgBench/tasks/ShExValidationStatic

Graph Engineering tasks. For that purpose the two arms share a single mid-tier Claude 4.x model: `claude-haiku-4-5-20251001`, accessed through the official Anthropic API. The magnitudes reported here are contingent on this capability level: a ceiling-tier model would likely saturate more cells and mask effects, whereas for a smaller model the direction is less clear, the improvements could grow larger, or the deteriorations could dominate. Cost is compatible with repeating each cell $n = 20$ times per arm, the default value of the inherited benchmark, which offers a reasonable balance between statistical power and API cost.

4.4. Statistical Analysis

The analysis is materialized in a Jupyter notebook that orchestrates an internal library of utilities. We summarize its logic below.

Pipeline. The utility library is organized in modules with disjoint responsibilities: file discovery and parsing, pairing assignment, orchestration of the confirmatory and secondary analyses, and figure generation. The confirmatory analysis applies the rank test corresponding to the pairing regime to the primary metric of each cell and reports Cliff's δ with a 95% bootstrap confidence interval. The secondary analysis, of exploratory nature, applies the same statistical machinery to the sub-metrics that each task computes in addition to the one selected as primary, in order to shed light on which specific aspects of performance drive the effect observed on the primary metric.

Non-parametric Rank Tests. Scores are bounded in $[0, 1]$ and the exploratory figures show distributions that are far from Gaussian; the parametric assumptions of the t-test do not hold. The confirmatory analysis is therefore based on non-parametric rank tests: **Wilcoxon signed-rank** for paired cells (Text2Answer, Sparql2Answer, ShExValidationStatic) and **Mann-Whitney** U for unpaired cells (RdfConnectionExplainStatic, RdfFriendCount). The primary test is two-sided.

Effect Size: Cliff's Delta. The rank tests answer whether one arm tends to score above the other, but not by how much. To quantify the magnitude of the effect we use Cliff's delta [12], the probability that a random observation of *mcp* exceeds a random observation of *baseline* minus the reverse probability. The result is bounded in $[-1, +1]$. We classify the magnitude using the thresholds proposed by Romano *et al.* [13]: negligible $|\delta| < 0.147$, small $0.147 \leq |\delta| < 0.33$, medium $0.33 \leq |\delta| < 0.474$ and large $|\delta| \geq 0.474$.

Confidence Intervals. Alongside the point estimate of δ, we report a 95% confidence interval computed by **bootstrap** (ten thousand resamples for the confirmatory analysis). For paired cells the resampling preserves the intra-pair structure by sampling a single index vector applied to both aligned score vectors; for unpaired cells each arm is resampled independently. The percentile method is used to derive the interval, chosen for its robustness in nearly-degenerate cells.

Saturation Detection. When the sample variance of each arm is effectively zero and the difference of medians is also negligible, the cell is marked as *saturated*. In this case the hypothesis test returns $p = 1$ by construction and $\delta = 0$; the statistical machinery is not informative, and no confidence interval is reported.

Multiplicity. No multiple-comparison correction is applied across the eleven confirmatory cells; effects at $|\delta| \approx 1.00$ are robust to any reasonable correction, while the moderate ShEx effect is not.

4.5. Reproducibility

Reproducibility, to the extent compatible with a proprietary API endpoint, has been a design constraint from the start: unambiguous identification of the model via its dated version, explicit random seeds in

the entire statistical pipeline and publication of the code in the aforementioned repository. Both arms use the Anthropic SDK with default sampling parameters (no explicit temperature or top-p override); output-token limits differ (baseline 4096, mcp 8192), which we flag as a limitation of the design. The verbatim system prompt of the mcp arm and the versions of the MCP protocol, the SDK and the connector are documented in the repository.

5. Results

This section presents the figures that synthesize the outcome of the statistical analysis described in Section 4.4. The figures are reproduced without interpretation, which is reserved for Section 6. Additional exploratory figures are omitted for space and are available in the public repository[10].

Figure 2 reproduces the per-iteration distribution of the primary metric of every cell in both arms, as a dot plot with the median overlaid; cells where both arms score at the ceiling of the scale in all twenty iterations are marked as saturated. Figure 3 reproduces the confirmatory forest plot: one row per cell, ordered by increasing Cliff's δ, with the point estimate marked and the 95% bootstrap confidence interval as a horizontal segment. Filled markers denote paired cells (Wilcoxon), hollow markers denote unpaired cells (Mann-Whitney U); the color codes the sign of the effect; rows marked as saturated are drawn in gray at $\delta = 0$ with no interval. Figures 4 and 5 reproduce, respectively, the total tokens billed per iteration in both arms and the cumulative frequency of tool invocations by name in the *mcp* arm.

6. Discussion

6.1. Synthesis of Findings

The confirmatory forest plot (Figure 3) reveals three regimes:

Improvement. The four variants of *RdfFriendCount* reach $\delta = +1.00$ on `f1`: every observation of the *mcp* arm outperforms every observation of *baseline*, the maximum possible unilateral shift. *ShExValidationStatic* obtains $\delta = +0.30$ on `accuracy` with an approximate interval of $[+0.10, +0.50]$.

No measurable effect. The two variants of *Text2AnswerListOrganizational* and the two of *Sparql2AnswerListOrganizational* appear marked as saturated on `f1`: δ is not interpretable, and these cells do not provide evidence either for or against the *mcp* arm.

Deterioration. The two variants of *RdfConnectionExplainStatic* fall on the opposite side on `listF1`, with $\delta = -0.48$ in N-Triples (interval approximately $[-0.80, -0.15]$) and $\delta = -1.00$ in RDF/XML, where no iteration of the *mcp* arm reaches the value that the *baseline* attains in all of them. The mcp arm worsens the model's performance in this task.

6.2. RdfFriendCount

The positive effect in the four variants is dominant. In *baseline* the median of `f1` is around 0.02–0.03, while *mcp* concentrates at the ceiling of the scale. Without access to the tools of rudofMCP, the model does not manage to identify the node with the most incoming `foaf:knows` arcs by reading the graph directly; with SPARQL available through rudofMCP, the task becomes trivial. The secondary analysis precisely locates the origin of the effect: the four *recall* variants are saturated (both arms identify the target node), while *precision*, *brevity* and *noSpecialFriends* reach $\delta = +1.00$ in the twelve non-saturated combinations. That is, *baseline* includes the correct node inside a long list that destroys precision; *mcp* delegates the counting to SPARQL and answers with the exact IRI.

[10] https://github.com/rudof-project/LLM-KG-Bench-rudof

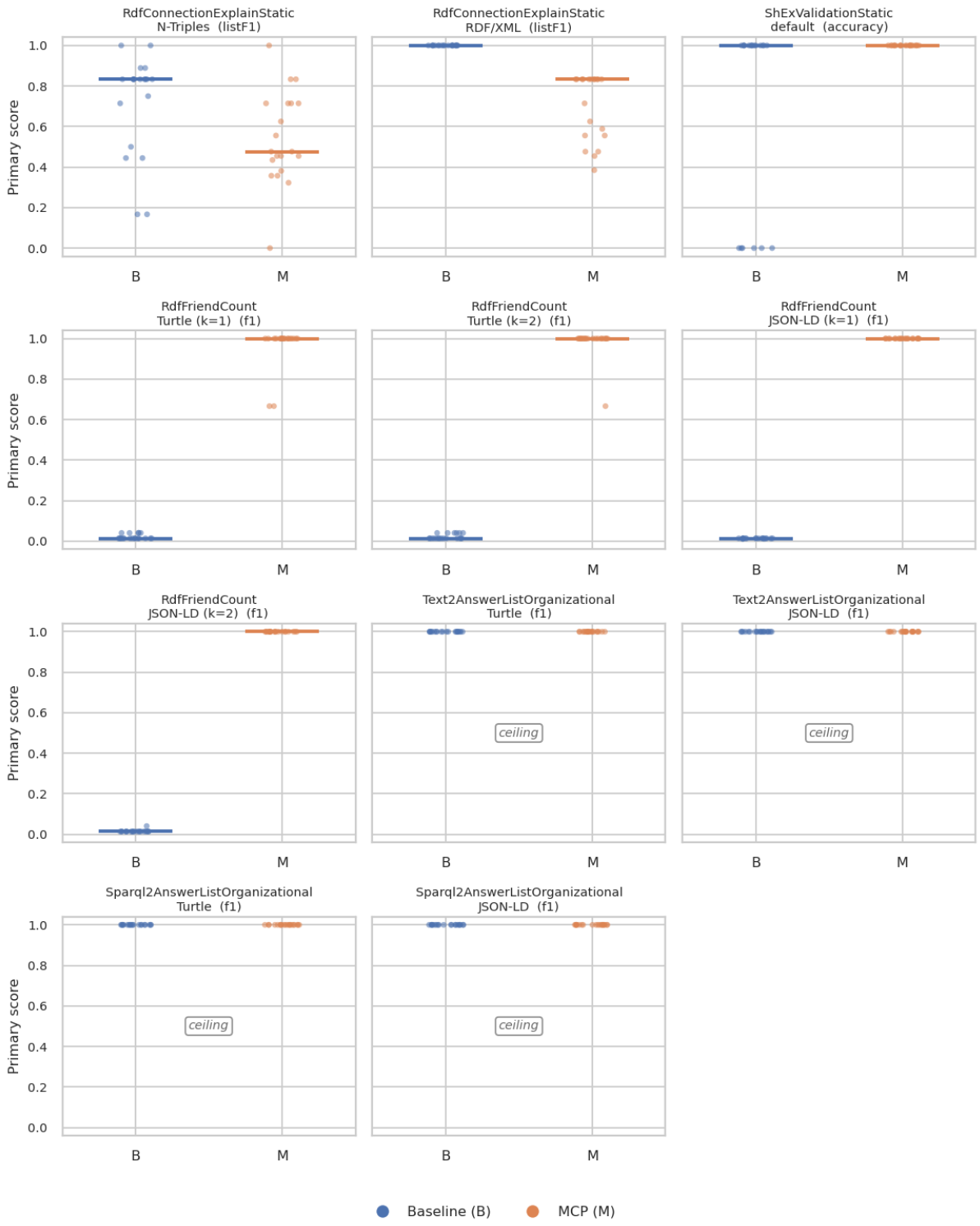

Figure 2: Cross-task distribution of the primary metric per iteration and cell. Each dot is one of the twenty iterations of an (arm, cell) pair; the horizontal bar marks the median. Cells at the ceiling in both arms are marked as saturated.

6.3. ShExValidationStatic

The effect is positive but small (per the Romano thresholds): $\delta = +0.30$ with an approximate interval of $[+0.10, +0.50]$. *baseline* correctly resolves most of the twenty items and fails a few; *mcp* reduces those failures thanks to the availability of `validate_shex` and `check_shex`, which let the model delegate the decision to the rudof validator instead of simulating ShEx's formal semantics.

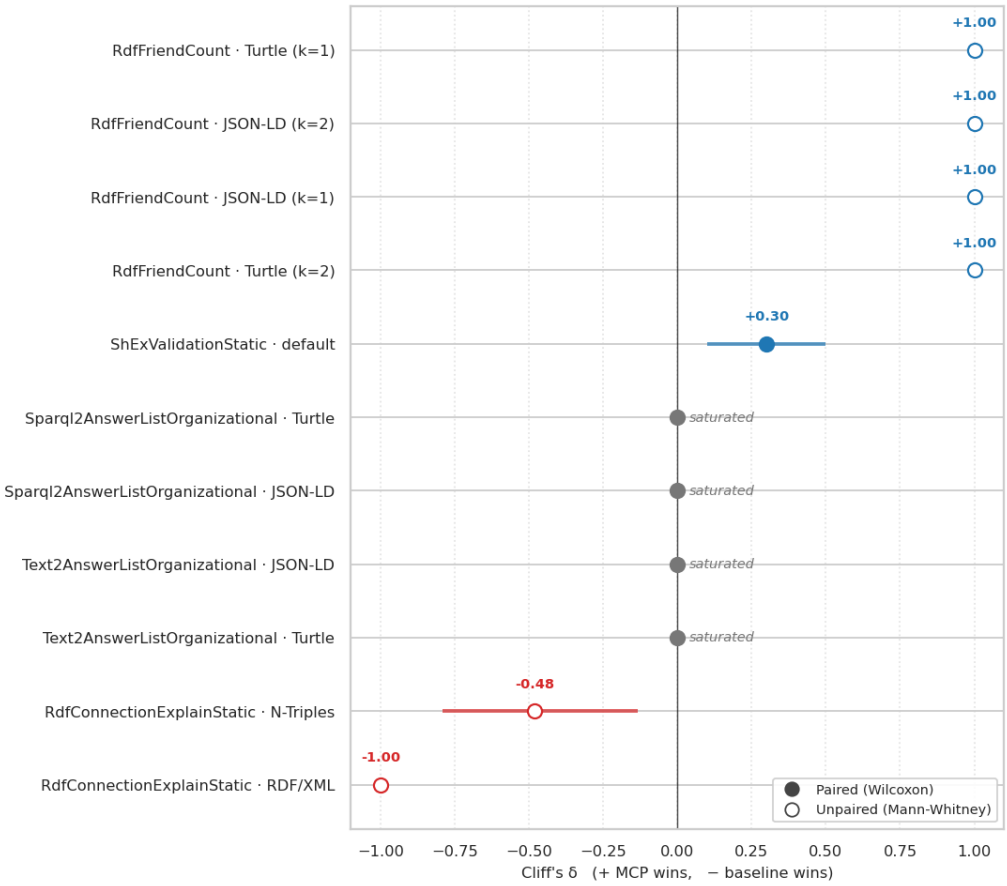

Figure 3: Confirmatory forest plot: point estimate and 95% bootstrap CI of Cliff's δ per cell. Positive δ favors *mcp* over *baseline*; negative δ favors *baseline*. Saturated cells are drawn in gray at $\delta = 0$.

6.4. Text2AnswerListOrganizational and Sparql2AnswerListOrganizational

The four cells are saturated: both arms obtain `f1` = 1 in all twenty iterations. The saturation extends to the three secondary metrics evaluated on each task (*combinedF1*, *relaxedF1*, *brevity*). The experiment cannot pronounce itself in these cells, not because MCP is not useful, but because the tasks are too easy for Haiku 4.5 in their current form: the organizational graph (five entities around an organization with two departments) fits comfortably in the model's context. The saturation nonetheless allows a useful observation: adding rudofMCP does not break what the model already does well.

6.5. RdfConnectionExplainStatic

This task is where the deterioration occurs. The cause is not that the assisted model does not find the answer. The secondary analysis decomposes the deterioration into five sub-metrics that must be read together: *listRecall* rises in N-Triples ($\delta = +0.29$) and is saturated in RDF/XML: the *mcp* arm identifies the nodes of the path as well as or better than the *baseline*. *listPrecision* falls to $\delta = -0.57$ in N-Triples and $\delta = -1.00$ in RDF/XML: the model finds the correct path but delivers it surrounded by extra elements: intermediate blank nodes (`_:n...`) that the task explicitly asks to exclude, numbered lists with explanatory arrows, and even duplicate copies of the final list. *lineBrevity* drops in parallel to *listPrecision* ($\delta = -0.50$ and -1.00). *textHttpF1* drops slightly in both variants ($\delta = -0.14$ and -0.15) because some iterations intersperse IRIs in narrative sentences or embed them in numbered lists. *textShortF1*, which admits short aliases, barely moves. The pattern is clear: all sub-metrics that penalize size or noise fall, and those that only require coverage rise or stay flat. The tool-usage figures point to the origin of that contamination: these two cells concentrate the highest invocation profiles, consistent with an agentic loop that explores the graph node by node, repeats SPARQL queries seeking

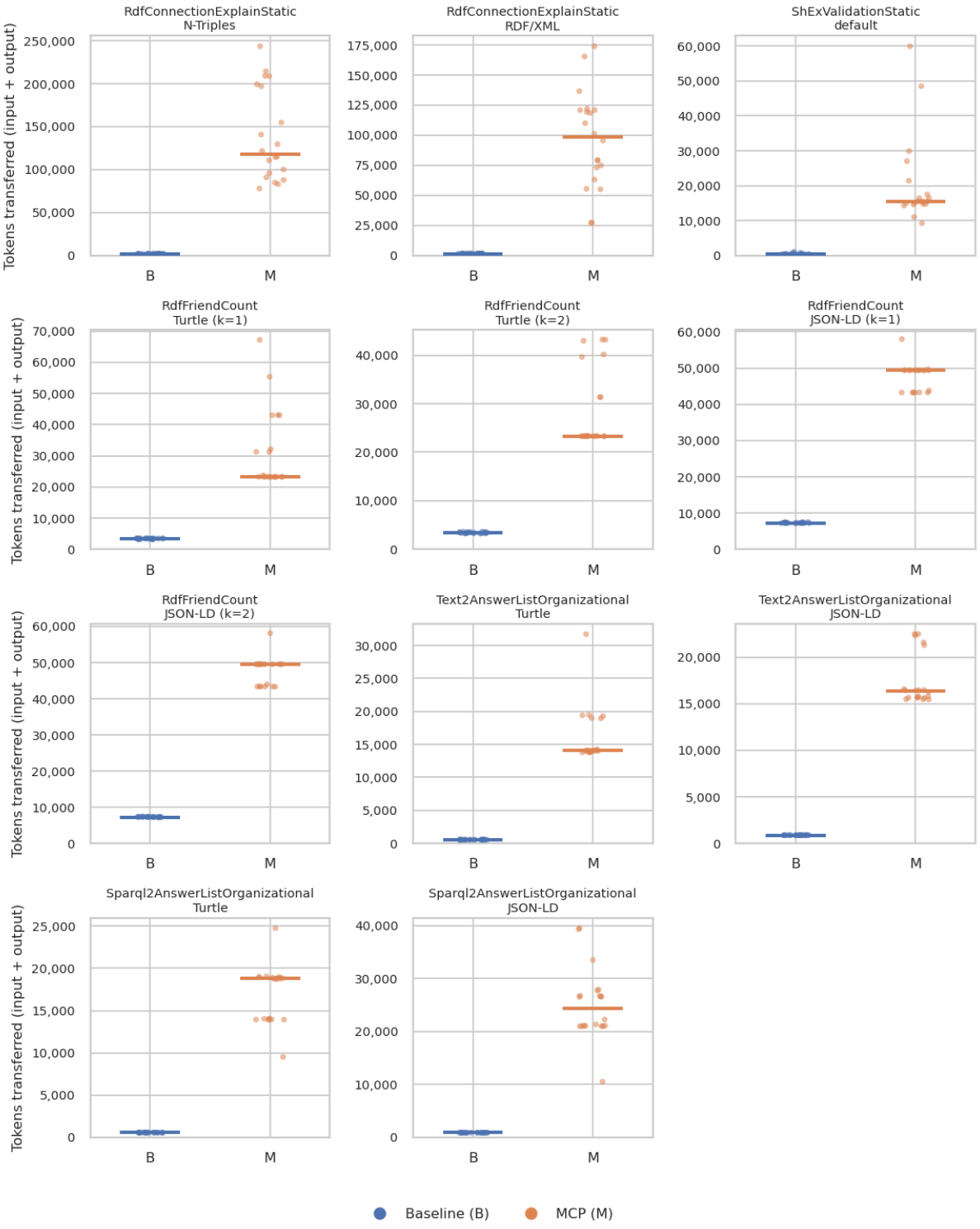

Figure 4: Cross-task distribution of the total tokens billed per iteration (input plus output) for the two arms.

information, and ends up answering with the context accumulated during exploration instead of the minimal answer required.

6.6. Operational Cost of the MCP Server

The overhead of the *mcp* arm is systematic. Baseline medians per iteration are around 1–2 s across the static tasks and 7 s in *RdfFriendCount*, while the mcp arm ranges from around 16 s (*ShExValida-*

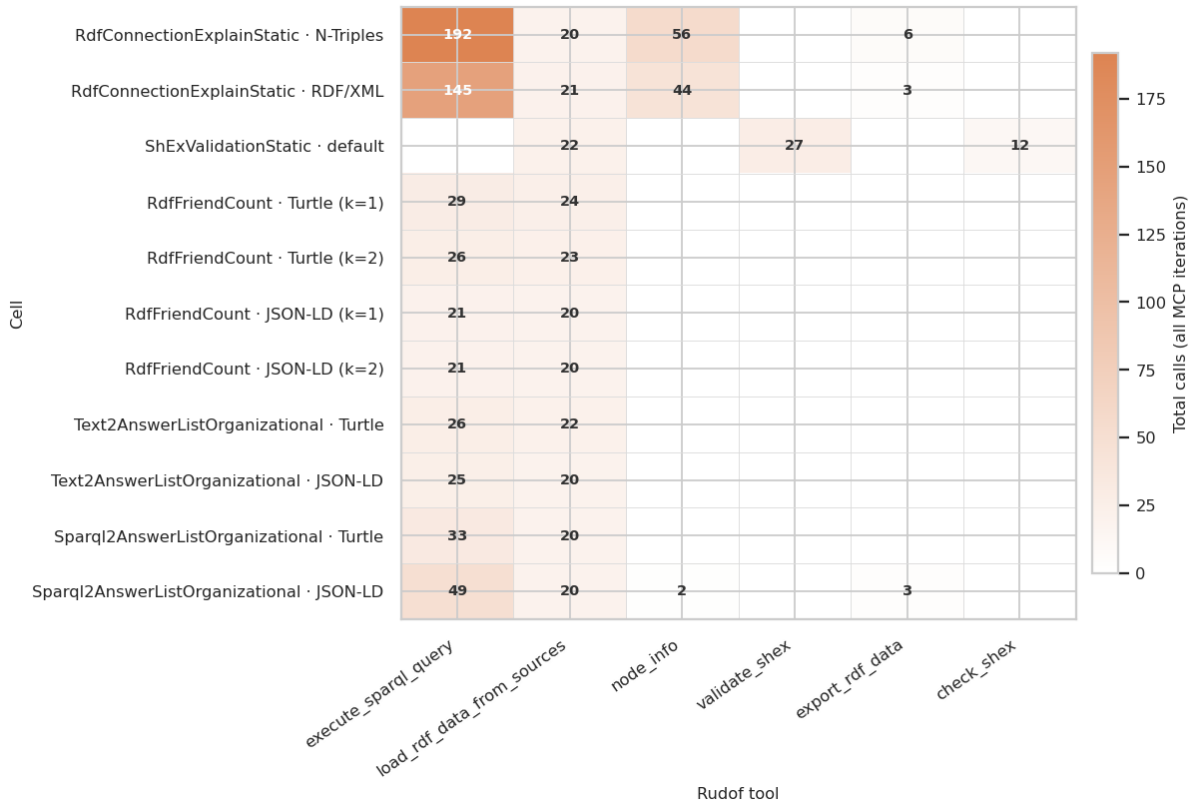

Figure 5: Heatmap of the cumulative frequency of tool invocations by name, aggregated over the twenty iterations of each cell of the *mcp* arm. Rows are cells; columns are tools of rudofMCP; darker cells indicate a larger number of invocations.

tionStatic, *Text2Answer/Sparql2Answer*) to 26–51 s (*RdfFriendCount*) and up to 112–134 s in the two variants of *RdfConnectionExplainStatic*; the full per-cell distribution is available in the public repository. Token consumption follows the same pattern (Figure 4). The most expensive cells are the two of *RdfConnectionExplainStatic*, which record medians of 12–13 tool calls per iteration split between `execute_sparql_query` and `node_info`. The four cells of *RdfFriendCount* are also costly, though for a different reason: even though the median number of invocations is low (two per iteration, concentrated on `execute_sparql_query` and `load_rdf_data_from_sources`), the SPARQL responses over the graph dump an appreciable volume of tokens into the context at each turn. *ShExValidationStatic* is the only cell that invokes `validate_shex` and `check_shex`, with a low median call count.

7. Conclusions and Future Work

In light of the results, the answer to the research question posed in Section 4.1 is a nuanced *yes*: the effect exists and is statistically detectable in non-saturated cells, but its sign and magnitude depend on the task family.

rudofMCP improves performance when the task admits delegating the work to a formal operation that rudof resolves better than the model itself (maximum effect in the four *RdfFriendCount* variants at $\delta = +1.00$; positive though more moderate in *ShExValidationStatic* at $\delta = +0.30$), and deteriorates it when the task rewards concise answers and the availability of tools induces over-exploration of the graph (the two variants of *RdfConnectionExplainStatic*). Crucially, the deterioration is not one of *correctness* (recall rises or saturates) but of *conciseness*: the model finds the correct path but delivers it surrounded by prose, blank nodes and duplicates, which `listF1` records as a loss. The four saturated cells of *Text2Answer/Sparql2AnswerListOrganizational* do not allow a verdict but show that rudofMCP does not break what the model already resolves. On top of this balance, a systematic operational cost in

latency and tokens must be paid.

Beyond the quantitative verdict, there is a qualitative reading worth underscoring. Delegating Knowledge Graph Engineering to rudof through an MCP server leaves an inspectable audit trail of the external operations behind each answer: the sequence of invoked tools and the results each returns are available *a posteriori* in the logs of the server and the connector. This is not interpretability of the model's internal reasoning, but a concrete trace of the formal operations the model relied on: a partial but useful mitigation of the opacity that the literature identifies at the LLM/KG intersection [1]. Above all, there is an implication tied to the very nature of the protocol: MCP is a Context Engineering technology that operates at inference time, so every time a Semantic Web standard evolves (for example, a new version of RDF or SPARQL [14, 15, 16]) it is enough to update rudof for any model that uses the server to operate with the up-to-date standard, without retraining or fine-tuning cost.

Limitations. The scope of the study is bounded by a set of limitations. The benchmark has been run over a single model, which prevents generalizing the findings to other models. From the sixteen task families available in the inherited catalog, four have been executed, complemented by the new `ShExValidationStatic` task; the five families cover qualitatively distinct capabilities, but do not exhaust the space of tasks the benchmark could pose nor include, for example, SHACL validation, another capability exposed by rudofMCP. Four of the eleven cells were saturated in both arms because the organizational graph on which they operate is trivial for Haiku 4.5. Finally, each cell has been repeated twenty times per arm, amply sufficient to detect the large effects that dominate the results, but insufficient to detect small or moderate effects with adequate power.

Future work. Several fronts open up. On the artifact side, broadening rudofMCP's tool catalog to capabilities not yet covered (LPG or DCTAP integration, additional SHACL modes) and refining the existing ones so that models invoke them more accurately. On the evaluation side, redesigning the saturated tasks so that they become discriminative again and adding tasks that exercise the capabilities not yet covered (starting with SHACL validation). On the protocol side, exposing the same functionality via alternative interfaces (in particular Anthropic *skills*) and running a comparative study between them.

Acknowledgments

This work has been partially funded by the project *SHAKIGRA: Shaping Knowledge and Interoperable Graphs*, code: *NAC-ES-PUB-ASV-2025 PID2024-157010OBI00*, from the Spanish Research Agency, and the regional project with code *SEK25-GRU-GIC-24-089*.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude in order to: Grammar and spelling check, Improve writing style, Paraphrase and reword. The same tool was also used as a coding assistant in the implementation of the accompanying software artifacts (the rudofMCP server, the `ModelClaudeMCP` connector, the `ShExValidationStatic` task and the analysis notebook), whose source code is published in the referenced repositories. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying large language models and knowledge graphs: A roadmap, IEEE Transactions on Knowledge and Data Engineering 36 (2024) 3580–3599. doi:10.1109/TKDE.2024.3352100.

[2] L.-P. Meyer, J. Frey, F. Brei, D. Heim, S. Gründer-Fahrer, S. Todorovikj, C. Stadler, M. Schröder, N. Arndt, M. Martin, Evaluating large language models for RDF knowledge graph related tasks – the LLM-KG-Bench framework 3, Zenodo, 2025. doi:10.5281/zenodo.16779482, preprint submitted to the Semantic Web Journal, under review.
[3] L.-P. Meyer, J. Frey, D. Heim, F. Brei, C. Stadler, K. Junghanns, M. Martin, LLM-KG-Bench 3.0: A compass for semantic technology capabilities in the ocean of LLMs, in: E. Curry, et al. (Eds.), The Semantic Web, Springer Nature Switzerland, Cham, 2025, pp. 280–296.
[4] L.-P. Meyer, J. Frey, F. Brei, N. Arndt, Assessing SPARQL capabilities of large language models, in: Proc. of the 3rd Int. Workshop on Natural Language Processing for Knowledge Graph Creation co-located with SEMANTiCS 2024, 2024. URL: https://ceur-ws.org/Vol-3874/paper3.pdf.
[5] J. Frey, L.-P. Meyer, N. Arndt, F. Brei, K. Bulert, Benchmarking the abilities of large language models for RDF knowledge graph creation and comprehension: How well do LLMs speak turtle?, in: Proc. of Workshop Deep Learning for Knowledge Graphs (DL4KG) @ ISWC23, 2023. doi:10.48550/ARXIV.2309.17122.
[6] L.-P. Meyer, J. Frey, K. Junghanns, F. Brei, K. Bulert, S. Gründer-Fahrer, M. Martin, Developing a scalable benchmark for assessing large language models in knowledge graph engineering, in: Proc. of Poster Track of Semantics 2023, 2023. doi:10.48550/ARXIV.2308.16622.
[7] Anthropic, Model Context Protocol Specification, https://modelcontextprotocol.io/specification/2025-11-25, 2025. Accessed: 2026-06-24.
[8] A. R. Kinjo, Y. Yamamoto, S. Bustamante-Larriet, J.-E. Labra-Gayo, T. Fujisawa, TogoMCP: Natural language querying of life-science knowledge graphs via schema-guided LLMs and the Model Context Protocol, bioRxiv, 2026. doi:10.64898/2026.03.19.713030, preprint, not peer-reviewed.
[9] R. Taelman, J. De Smet, Comunica MCP SPARQL: Improving the accuracy of AI agents using SPARQL-based access to decentralized knowledge graphs, in: Proc. of the 23rd Extended Semantic Web Conference: Posters and Demos (ESWC 2026), 2026. URL: https://rubensworks.github.io/article-eswc2026-demo-comunica-mcp/.
[10] J.-E. Labra-Gayo, A. Iglesias-Préstamo, D. Martín-Fernández, M.-A. Arnaud, rudof: A Rust library for handling RDF data models and Shapes, in: L. Etcheverry, V. Lopez Garcia, F. Osborne, R. Pernisch (Eds.), Posters, Demos, and Industry Tracks at ISWC 2024, volume 3828 of *CEUR Workshop Proceedings*, 2024. URL: https://ceur-ws.org/Vol-3828/paper32.pdf.
[11] rudof project, rudof: Rust library for RDF data models and shapes, GitHub repository, 2026. URL: https://github.com/rudof-project/rudof, accessed: 2026-06-24.
[12] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions, Psychological Bulletin 114 (1993) 494–509. doi:10.1037/0033-2909.114.3.494.
[13] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys?, in: Annual meeting of the Florida Association of Institutional Research, 2006, pp. 1–33.
[14] G. Kellogg, O. Hartig, P.-A. Champin, A. Seaborne, RDF 1.2 Concepts and Abstract Syntax, W3C Candidate Recommendation, W3C, 2026. URL: https://www.w3.org/TR/rdf12-concepts/.
[15] O. Hartig, A. Seaborne, R. Taelman, G. Williams, T. Pellissier Tanon, SPARQL 1.2 Query Language, W3C Working Draft, W3C, 2026. URL: https://www.w3.org/TR/sparql12-query/.
[16] H. Knublauch, T. Bergwinkl, Y. Taghzouti, J. Wright, Shapes Constraint Language (SHACL) 1.2 Core, W3C Working Draft, W3C, 2026. URL: https://www.w3.org/TR/shacl12-core/.