Journal Title
XX(X):2–37

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Evaluating Large Language Models for RDF Knowledge Graph Related Tasks - The LLM-KG-Bench-Framework 3

Lars-Peter Meyer[1,2], Johannes Frey[1,3], Felix Brei[1], Desiree Heim[4,5], Sabine Gründer-Fahrer[1], Sara Todorovikj[2], Claus Stadler[1,3], Markus Schröder[4], Natanael Arndt[1] and Michael Martin[1,2]

Abstract

Current Large Language Models (LLMs) can work with structured information and even assist developing program code, but can they support working with Knowledge Graphs (KGs) as well? Which LLM is offering the best capabilities in the field of Semantic Web and Knowledge Graph Engineering (KGE)? Is it possible to determine this without manually checking many answers? The LLM-KG-Bench framework is designed to answer these questions. It consists of an extensible set of tasks for which the LLM answers are automatically evaluated, and covers different aspects of working with semantic technologies.

This article gives a description of the LLM-KG-Bench framework, its main concepts, and the tasks implemented. In a benchmark run, a comprehensive dataset has been generated with it, evaluating more than 40 contemporary open and proprietary LLMs with 26 benchmark tasks, resulting in interaction logs and evaluations of roughly 45 000 LLM task dialogues. Finally, this dataset is used for an analysis of the SPARQL-related capabilities of the LLMs tested.

Keywords
Knowledge Graph, Large Language Model, RDF, SPARQL, LLM-Benchmark, LLM-Capabilities, LLM-RDF-Benchmark

1 Introduction

The combination of Large Language Models (LLMs) and Knowledge Graphs (KGs) offers great synergetic potential for various knowledge-driven applications and is still gaining more traction within the research community. However, the ongoing rapid development in the field of LLMs makes it difficult to keep up with the latest developments and put them into context of prior work. Several initiatives for automated benchmarking, like BIG-bench [1], HELM [2], or Chatbot-Arena [3], address the need for systematic evaluation of LLM performance. However, the tasks in scope of these evaluations are very general and do not address topics specifically related to KGs.

With the benchmarking framework *LLM-KG-Bench*, we are particularly focusing on automatically assessing and comparing LLMs regarding their capabilities to cope with Semantic Web technologies. This includes basic capability checks to establish a scientific foundation, as well as more practical tasks. The existing tasks cover already the Knowledge Graph Engineering (KGE) areas, RDF KG serialization, KG construction, SPARQL and KG schema, and the framework is open to support more tasks. Connectors for many open weight and top-ranking commercial LLMs are included.

Our investigation has been guided by the following overarching research questions:

RQ 1. *Can KGE-related capabilities of LLMs be evaluated in an automated way?*

[1]InfAI e.V., Leipzig, Germany
[2]TU Chemnitz, Germany
[3]Uni Leipzig, Germany
[4]DFKI Kaiserslautern, Germany
[5]RPTU University Kaiserslautern-Landau, Germany

Corresponding author:
Lars-Peter Meyer, InfAI e.V. Leipzig, Goerdelerring 9, 04109 Leipzig, Germany
Email: lpmeyer@infai.org

Preprint submitted for review at Semantic Web Journal 2025

Prepared using sagej.cls *[Version: 2017/01/17 v1.20]*

RQ 2. *Should specific LLMs be favored for specific tasks or KG data formats? And if so, to which extent do their capabilities differ and how can it be measured which LLM fits best for which task?*

And specific to SPARQL SELECT queries:

RQ 3. *Can LLMs follow the syntactic rules of* SPARQL SELECT *queries?*

RQ 4. *Can LLMs parse the semantics of* SPARQL SELECT *queries and act accordingly?*

RQ 5. *Can LLMs write semantically correct* SPARQL SELECT *queries for a given question and KG?*

We present here an overview on the current state of the LLM-KG-Bench framework together with an evaluation dataset on more than 40 current LLMs and an evaluation of their *SPARQL-SELECT*-query-related capabilities. The LLM-KG-Bench framework is developed for automated benchmarking of Knowledge-Graph-related capabilities of LLMs.

The main contribution of this article is threefold:

- We provide a description of the current state of the framework, including main concepts, its API, supported model connectors and tasks.
- We present a dataset on the evaluation of more than 40 LLMs. The dataset includes detailed logs, interaction data and the evaluation result. This enables other researchers to extend this dataset, reproduce our analysis or run additional analyses e.g. by updating the framework code and using the framework's reevaluation feature.
- As a demonstration of the frameworks' analytical capabilities and the generated dataset, we evaluate the *SPARQL-SELECT*-query-related capabilities of the selected LLMs.

This is an extended version of previous work by Meyer et al. 2025[4] with a similar use case as the one presented in Meyer et al. 2024[5]. The added contribution of this paper compared to our previous work consists of:

- a unified source of truth for the current state of the LLM-KG-Bench framework, especially based on current conference and workshop articles[4–6]

- a refined description of capabilities that an LLM needs to have in order to solve the benchmark tasks
- the enrichment of the previous result dataset by executing the benchmark framework against several top-ranked LLMs
- grouping and refining of the capability plots together with the computation and analysis of an aggregated score for the SPARQL-related capabilities of the LLMs tested
- an improvement of the analysis of using Turtle(TTL) versus JSON-LD RDF serialization format in tasks by including a per-model as well as a per-task summary.

2 Related Work

2.1 LLM Evaluation

Attempts on exploring and navigating the vast amount of LLMs are made in form of several LLM leaderboards, which provide evaluation services for LLMs in form of selections of benchmarks or workloads. We discuss here the most relevant ones that we discovered during our research.

Focusing mainly on commercial models, the *Chatbot Arena*[1,3] lists scores for *MMLU*[2] and *MT-bench*[7], and also calculates its own arena-score (based on prompts given by users, processed by two models side-by-side and evaluated by the same user voting for his preferred answer). For open models, the *Open-LLM-Leaderboard* [3] provides a list of benchmark results with over 2,000 tested models and several scores plus a carbon dioxide emission estimate. The most exhaustive list is provided by *HELM*[4,2] which includes also domain-specific tests like LegalBench and MedQA. While a comparison of the individual benchmark suites would go beyond the scope of this paper, to the best of our knowledge, none of these general leaderboards addresses Knowledge Graph Engineering (KGE) tasks.

Moreover, what is generally amiss is a benchmark execution framework that helps to deal with the particularities of RDF and KG-related workloads (format parsing, syntax check feedback loops, execution and evaluation of queries towards KGs, etc.). Although early attempts have been made by *BIG-bench*[1], it is falling short of building an efficient base for evaluating LLM capabilities in the context of KGE tasks, due to insufficiencies of the provided Task API, along with the focus on multiple choice tasks and scores based on string or document similarities.

In the face of these issues, our LLM-KG-Bench framework aims to reduce complexity and facilitate creating, executing, evaluating, and analyzing KG-related tasks. Its procedures focus on the syntactically and semantically correct generation of RDF (i.e. in Turtle) and SPARQL.

Current efforts in benchmarking coding capabilities seem more closely related to characteristics of KGE capabilities benchmarking, at a conceptual level (e.g., with respect to output format requirements, instruction complexity, and response evaluation strategies). Several leaderboards, like the *Big Code Models Leaderboard*[5] for Java, Javascript,and CPP, or the *EvalPlus Leaderboard*[6] for Python assess the coding proficiency of LLMs. We included some of the models ranked there into our test set.

2.2 RDF-related evaluation

In the current literature[7] that explores the combination of LLMs and KGs[8–10], several attempts on evaluating the application of LLMs for KG-related tasks are made. However, these LLM evaluations often focus on details for a very selective problem in a specific task area, like Text to RDF (e.g.[11,12]), Knowledge Base Extraction from LLMs (e.g.[13–15]), Knowledge Graph Question Answering (KGQA, e.g.[16]), Text2SPARQL (e.g.[17–19]) or generation of RDF Mapping Language (RML) mappings (e.g.[20]). Unfortunately, many of these evaluations have been conducted manually, which causes scalability issues with respect to more repetitions and including more or newer models. In case an automated evaluation has been performed, the underlying code usually lacks adaptability to encompass new models or task variations. A benchmarking effort that is related to our interest in studying the JSON-LD capabilities, is *StructuredRAG*[21]. It consists of six tasks designed to assess LLM capabilities in following response format instructions according to JSON templates. Table 1 compares several of the mentioned LLM evaluation approaches with respect to the covered LLMs, addressed KGE tasks and their evaluation mode. Only the LLM-KG-Bench framework combines automatic evaluation for many covered LLMs and several KGE topics.

Table 1. Comparison of some of the LLM evaluation approaches mentioned here. Best values are marked with bold font. Only the LLM-KG-Bench framework automatically evaluates several Knowledge Graph Engineering (KGE) tasks and covers many LLMs in a benchmark run.

	LLMs covered	KGE topics	eval. type
BIG-bench [1]	**many**(about 15)	no	automatic
HELM [2]	**many**(about 140)	no	automatic
Chatbot Arena [3]	**many**(about 280)	no	crowd
ChatGPT KG Experiments [9]	some (2)	**several** (Text2Sparql, KG constr., KG exploration)	manual
Text2KgBench [11]	some (2)	Text2KG	automatic
AutoKG [12]	some (4)	Text2KG, Reasoning	manual
LLM-KG-Bench 3	**many**(about 40)	**several** (RDF, KG constr., SPARQL, KG schema)	automatic

2.3 Text2Sparql and KBQA

The interpretation and generation of the query language SPARQL, being a core technology for accessing KGs, is one especially relevant example of integrating Semantic Technology with LLMs.

Rangel et al.[22] introduce a method for fine-tuning OpenLLaMA to generate SPARQL queries for question answering on life science knowledge graphs. Their approach leverages data augmentation techniques, such as the use of meaningful variable names and inline comments, which lead to improved accuracy in SPARQL query generation.

Bustamante and Takeda[23] focus on enhancing the generation of SPARQL queries from natural language questions. They employ a GPT model to identify the most challenging aspects of the Text2SPARQL task, aiming to apply targeted solutions to these difficulties.

Avila et al.[24] evaluate ChatGPT's performance in answering natural language questions on knowledge graphs. Their method, Auto-KGQAGPT, explores translating such questions into SPARQL queries using prompts that include relevant knowledge graph fragments.

Li et al.[25] address the performance drop seen in real-world conditions where high-quality annotated data are lacking. They propose the FlexKBQA framework, which uses template SPARQL queries that are converted into natural language questions via LLMs to produce synthetic training data. This data is used to fine-tune a lightweight SPARQL generator, further refined with real queries to bridge the gap between synthetic and real-world inputs.

Diallo et al.[26] provide a thorough evaluation comparing pre-trained language models, non-pre-trained language models, and LLMs, along with various fine-tuning strategies. Their error analysis reveals that

incorrect URIs, often caused by hallucinations, are a major source of failure in generated SPARQL queries. Hirigoyen et al.[27] demonstrate that incorporating a copy mechanism can help mitigate such hallucinations.

Kovriguina et al.[17] present SPARQLGEN, a one-shot generation technique that uses LLMs to produce SPARQL queries. Their method includes the full context comprising the question, a relevant RDF subgraph, and an example query within a single prompt.

Pliukhin et al.[28] propose a similar approach to SPARQLGEN for query generation over the ORKG scholarly knowledge graph, enhanced with a more advanced subgraph extraction method. Zahera et al.[18] extend this by incorporating chain-of-thought prompting and using the GERBIL QA system[16].

Lehmann et al.[29] advocate the use of Controlled Natural Language (CNL) as a user interface due to its proximity to natural language. They show that CNL can be reliably and unambiguously translated into formal query languages like SPARQL, significantly reducing the need for large training datasets.

Diefenbach et al.[30] introduce two datasets for training and benchmarking QA systems on Wikidata: One derived from the SimpleQuestions dataset[31], and another based on user logs and feedback.

Dubey et al.[32] extend the LC-QuAD dataset and introduce LC-QuAD 2.0[33], which is compatible with both Wikidata and DBpedia. The dataset comprises 30,000 questions, including paraphrases and corresponding SPARQL queries.

Banerjee et al.[34] explore various ways to integrate LLMs into standard knowledge graph workflows. Their focus is on models that can be fine-tuned with consumer-grade hardware, in contrast to approaches focusing on large-scale models used without additional training.

Frey et al.[35] demonstrate the long-term value of automated benchmarking through the LLM-KG-Bench platform. They compare multiple LLM iterations with a focus on their ability to handle RDF Turtle syntax, preserving responses for future evaluation. Heim et al.[6] is an example of such a separate evaluation.

Hofer et al.[20], consistent with the findings by Frey et al.[35,36], investigate LLM-driven RML mapping generation in Turtle. They find that while syntactic errors occur, most LLMs can correct these mistakes. Given that

SPARQL is built on Turtle, the authors conduct their experiments as multi-turn dialogues with feedback loops for error correction.

The recently organized "First International Text2Sparql Challenge"[19] shows the potential of best LLM agent-based approaches[37–39] for the Text2Sparql problem and a specific automatic evaluation approach roughly similar to the Text2Sparql evaluation used in LLM-KG-Bench. But it shows as well the open gap, as the best participants reached only scores of about 0.5.

Nevertheless, none of these existing works reach the goal of automatic evaluation on many KGE-related tasks as presented here with the LLM-KG-Bench framework in version 3.

3 The LLM-KG-Bench Framework

Benchmarking LLMs involves significant time, financial costs, organizational effort, and the evaluation process can often be imprecise. LLM-KG-Bench is designed to simplify the creation of KG-related assessments while providing a foundational infrastructure for further development. Its main features are:

- **Modular and Extensible Framework:** Supports automated evaluation tasks using a comprehensive set of KG-extraction and evaluation-related helper methods.
- **Built-in Correction Cycles:** Implements dialogue-based correction cycles, enabling LLMs to revise previous mistakes.
- **Data Security:** Supports encryption of task data to prevent test data leakage into LLM training datasets.
- **Task Management:** Manages task configurations, evaluation orchestration, logging, and result persistence.
- **Result Analysis and Visualization:** Provides built-in tools for analyzing and visualizing evaluation results.
- **Broad Model Support:** Includes connectors for many contemporary LLMs.
- **Open-Source Codebase:** The framework is published as open source and welcomes extensions and community contributions.

In the following sections, we describe the basic concepts and infrastructure of the LLM-KG-Bench framework in greater detail.

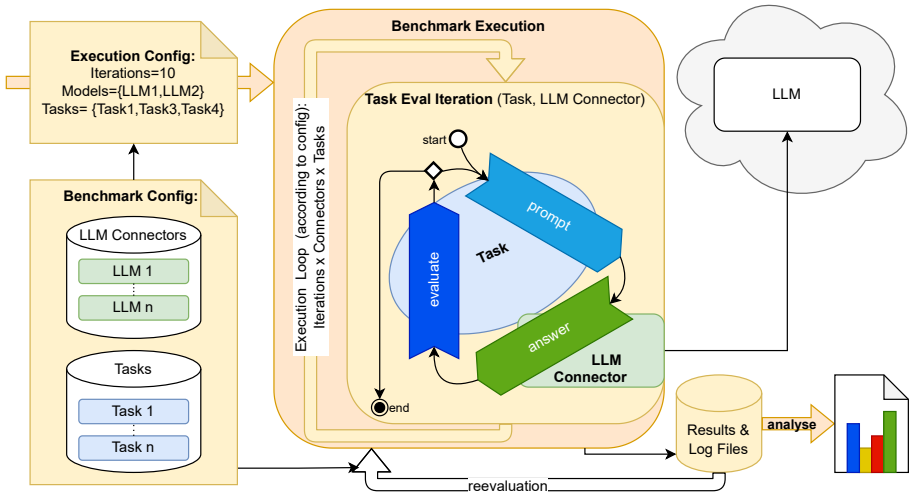


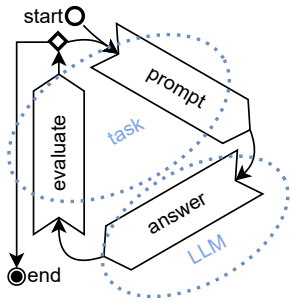
Figure 1. Overview on the execution flow of the LLM-KG-Bench framework. A *execution configuration* defines some settings and selects *LLM connectors* as well as *evaluation tasks* from a *benchmark configuration*. In the *benchmark execution* all given combinations of evaluation tasks and LLM connectors are executed for the number of iterations each as *task evaluation iteration*. In the *task evaluation iteration* the task provides a prompt which is answered by the LLM via the *LLM connector* and the answer is evaluated by the task, maybe resulting in further Prompt-Answer-Evaluate rounds. All results and logs from the *task executions* is stored for later analysis, visualization or *reevaluation*.
Source: LLM-KG-Bench documentation, updated version of Meyer et al.[40].

3.1 Architecture and Main Concepts

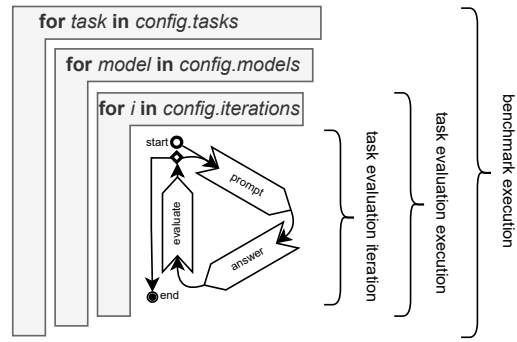
Figure 1 gives an overview on the LLM-KG-Bench framework and its execution flow. It is build around some main concepts we want to describe here. We start with the general concept of the Prompt-Answer-Evaluate loop before going further to LLM connectors, tasks, execution scopes, configuration, and scores.

Prompt-Answer-Evaluate Loop: The evaluation of LLMs is based on dialogues, consisting of prompts and answers. This enables tasks to make use of the chat capability of modern LLMs and their bigger supported context size to get the answer closer to the correct one in an iterative way.

The Prompt-Answer-Evaluate loop starts with the generation of an initial prompt that is sent to an LLM. In the next step, the produced answer is evaluated. Based on the evaluation result, the framework can decide to start a new Prompt-Answer-Evaluate round or stop the dialogue. The



(a) The Prompt-Answer-Evaluate loop for the task — LLM interaction as organized by the framework. Prompting and evaluation is covered by the task, the answer is generated by the LLM.



(b) Different execution scopes: task evaluation iterations (includes one to many cycles of the Prompt-Answer-Evaluate loop), task evaluation execution (includes all iterations), benchmark execution (includes all task executions for all combinations of tasks and LLM models defined for all iterations).

Figure 2. Overview of the evaluation workflow and execution scopes.

process from initial prompt to the stop of the dialogue is named a *task evaluation iteration*.

The structure of these loops is shown in fig. 2a and an example dialogue is given in fig. 4. The prompt and evaluate parts are implemented in *evaluation tasks* and the answer is organized with the help of *LLM connectors*.

LLM Connectors and LLM Connector Classes: The *LLM connectors*(or *model connectors*) offer a consistent abstraction layer to interact with a specific *LLM* (or *model*). In the *benchmark configuration* the *LLM conncectors* are defined by parametrizing *LLM connector classes* for specific *LLMs*. Several *LLM connector classes* classes are implemented in the LLM-KG-Bench framework as described in section 3.3, adding support for many different *LLMs*.

Evaluation Tasks, Task Classes and Parametrized Tasks: The *evaluation tasks* (or *benchmark tasks*) are the main building block of a benchmark and automatically evaluate the LLM answers. For the *Prompt-Answer-Evaluate loop* the tasks provide the prompt and evaluation functionality.

Task Classes and Parametrized Tasks: The *evaluation tasks* can be defined either direct in *task classes* or as *parametrized task classes* derived in the benchmark configuration from a *task class* with a *task parameters* definition.

Task Parameters: Several *task classes* support as *task parameter* a *targeted task size* or a serialization format parameter. The *targeted task size task parameter* defines the count of characters expected to be exchanged with the *LLM connector* in one *task iteration*.

List Tasks and Task Case Entries: *List tasks* have a list of *task case entries*, where each entry defines a distinct exercise resulting in a specific prompt and expected answer. For each *task iteration* one *task case entry* is selected from this list. All *task case entries* are evaluated by the same *list task*.

Task Evaluation Iterations: We name one task evaluation loop consisting of one or more Prompt-Answer-Evaluate rounds a *task evaluation iteration* (or a *dialog*), see also fig. 2b.

Task Evaluation Executions: Since LLM answers are generated probabilistically, a configurable number of *task evaluation iterations* is executed, collectively forming a *task evaluation execution* for a specific task and a particular LLM, see also fig. 2b.

Benchmark Executions: A *benchmark execution* consists of all task executions for all combinations of tasks and models defined in the *execution configuration*, see also fig. 2b.

Result Reevaluation LLM answers recorded in a *benchmark execution* can get reevaluated by repeating the task evaluation code with the *LLM connector* interactions substituted with the stored interactions. When looking at the Prompt-Answer-Evaluate loop given in fig. 2a, the evaluate code is reexecuted but prompt and answer is taken from the recorded interaction. This makes it possible to reproduce evaluation results and even offers the chance to run updated evaluation code on given answers without the need of new maybe costly or differing LLM interactions.

Benchmark Configuration and Execution Configuration: A *benchmark configuration* specifies the tasks and models to be included in a benchmark run together with the number of iterations per task execution. A *benchmark configuration* can be executed as a whole or with a selection of tasks and models defined by command line parameters.

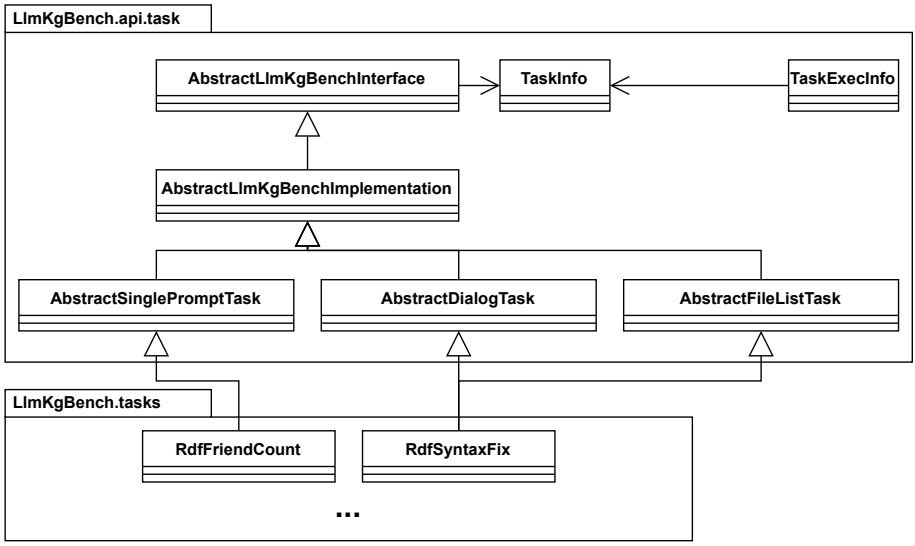


Figure 3. UML class diagram of the Task API and its reference by some example evaluation tasks. All evaluation tasks implement the *AbstractLlmKgBenchInterface* via an inheritance connection. A task can be described with a *TaskInfo* object. A *TaskExecutionInfo* references this *TaskInfo* object for the documentation.

Evaluation Scores and Infos: An *evaluation task* should compute for each *task evaluation iteration* a key value dictionary containing one or more *evaluation scores* with values in the range of $[0..1]$. Here 1 should be the score for an optimal answer. In addition to *evaluation scores* an *evaluation task* can return *evaluation infos* containing more information in the form of a key value dictionary, where the values could be anything usefull about the evaluation of the answers.

3.2 Tasks API

Evaluation tasks are implemented following the Task API as a common interface between evaluation tasks and the framework. In LLM-KG-Bench framework Version 3, a major update of the Task API was introduced together with new helper classes. Figure 3 shows a UML class diagram of the new Task API.

All evaluation tasks in LLM-KG-Bench implement the interface *AbstractLlmKgBenchTaskInterface*. It defines the interface between

evaluation tasks and the framework, including the method signatures for Evaluate and Prompt steps as well as task serialization and deserialization.

The following methods defined in the *AbstractLlmKgBenchTaskInterface* are especially important:

getNextPrompt: combines an evaluation and prompting step. If no new prompt is generated the Prompt-Answer-Evaluate loop ends.

finalizeEvaluation: is called at the end of the Prompt-Answer-Evaluate loop and creates a final evaluation result for this task evaluation iteration.

condenseTaskData: creates a serializable representation of this concrete task case entry. This offers the possibility for later continuation or reevaluation.

createTaskFromCondensedData: initializes a task from the representation given by *condenseTaskData*

The abstract implementation *AbstractLlmKgBenchTaskClass* helps to reduce redundant code and eases the concrete task implementation. For the two main variations of tasks, single-prompt tasks and dialogue-tasks, specialized abstract classes are provided. The *AbstractDialogTask* class helps with handling the score and info values produced by each Prompt-Answer-Evaluate round and computes mean and max aggregate values for scores at the end of a task evaluation iteration. Tasks that store their task data in an encrypted file can benefit from the abstract class *AbstractFileListTaskImplementation*.

This task API is inspired by the BIG-bench [1] task classes but has a more detailed abstraction to let the framework orchestrate the tasks Prompt-Answer-Evaluate loop shown in fig. 2a. Nonetheless the task API should be similar enough with the common generic benchmarking frameworks to keep the path of combining them open for the future. The new task API gives the central framework logic more flexibility in the orchestration, reduces error possibilities and reduces repeated code in tasks.

3.3 LLM Connectors and Supported Models

LLM connectors are responsible for offering standardized APIs to LLMs. They are defined similar to the BIG bench model class. The main method

offered is `generate_text(inputs, ...)->str`, taking a single prompt or dialogue and returning the LLM's answer.

The LLM-KG-Bench framework offers several model connector classes:

OpenAI / ChatGPT: Connector class for OpenAI-compatible LLMs like GPT-3.5, GPT-4, GPT-4t, GPT-4o and GPT-o1 via the OpenAI python library[8] and REST API[9]. Many other LLM providers offer a compatible REST endpoint. They can be integrated with this connector as well.

Google / Gemini: Connector class for LLMs from Google like Gemini 1.5 or Gemini 2.0 via the Google python library[10] and REST API[11].

Anthropic / Claude: Connector class for LLMs from Anthropic from Claude 1.0 to Claude 3.5. The connector uses the Anthropic REST API[12] using the offered Python library[13].

vLLM : Runtime for self-hosted LLMs [14,41]. This library is compatible to many open LLMs and enables serving and inferencing with them.

3.4 Evaluation Tasks and Scores

Several tasks are implemented in the LLM-KG-Bench framework as described in several articles[4,5,35,36,40]. The tasks range from basic RDF capability checks to establish a clean scientific foundation for further experiments, but include more practical tasks as well.

Here we list all task classes implemented at the moment together with the most important scores. More details are given in the referenced papers and in the tasks folder of the code repository [15]. Table 2 shows the relationship between the *SPARQL-SELECT*-query-related tasks and the research questions 3 to 5.

RDF-related Tasks:

RdfConnectionExplainStatic: This task asks the LLM to find the shortest connection between two nodes in a small RDF graph. The shortest path shall be output as a list of IRIs. Four variations present the graph in different serialization formats: JSON-LD, N-Triples, Turtle, and RDF/XML. See also Frey et al.[36] and Meyer et al.[4].

Most important score `listTrimF1`: F1 measure calculated by comparison of trimmed given and expected lines.

RdfFriendCount: For a simple RDF graph with, respectively, only one node and one edge type, the LLM is asked to identify the node with the most incoming edges. There are several variations covering on the one hand the different serialization formats JSON-LD, N-Triples, Turtle and RDF/XML and on the other hand using different values for the other parameters `size`(targeted task size), `knowsCount`(number of incoming edges for normal nodes), and `specialAddKnowsCount` (number of additional incoming edges for the one special node). See also Frey et al.[36] and Meyer et al.[4].

Most important score `f1`: F1 measure when comparing the given persons with the correct answer.

RdfSyntaxFixList: The LLM is instructed to correct a syntactically invalid RDF graph. The variations cover the serialization formats JSON-LD, N-Triples, and Turtle. For more details, see Meyer et al.[4].

A former version is the **TurtleErrorsStatic** task[36,40], which is limited to a single turtle test case but implements an evaluation method optimized for Turtle serialization format.

Most important scores `0_combined` and `max_combined`: Score combining for the first (0) or best (max) answer in the dialog a string similarity measure score, a parsable syntax score, and a f1 measure on the KG content. The combined scores are calculated as $0.1 \cdot strSimilarity + 0.2 \cdot parsableSyntax + 0.7 \cdot contentF1$.

RDF-related Tasks on KG-Construction:

FactExtractStatic: The LLM is instructed to extract facts from a given textual fact sheet and create a corresponding Turtle KG. See also Meyer et al.[40] and Frey et al.[36].

Most important scores `norm_f-measure`: F1 measure comparing triples of the normalized graphs given and expected.

TurtleSampleGeneration: In this task, the LLM shall generate small Turtle knowledge graphs satisfying given requirements. See also Meyer et al.[40] and Frey et al.[36].

Most important score `persons_relative_error`: Comparison of the number of persons in the generated graph with the expected number.

SPARQL-related Task Classes (for specific tasks see table 3):

Sparql2AnswerList (S2A): Given a small KG and a SPARQL SELECT query, the LLM shall return the respective result set for the query. Here, the KG is given in the JSON-LD or Turtle format. See also Meyer et al.[5].

Most important score `combinedF1Score`: Combination of several F1 measures comparing the expected and given answer.

Text2AnswerList (T2A): In this task, the LLM is instructed to return the result set answering a given textual question on a given KG. While this task is similar to KBQA, we are here mainly interested scientifically in the variation of the Sparql2Answer task. We use the same evaluation logic, similar prompts, and similar task case entries, but replace the *SPARQL SELECT* query by the corresponding textual question. Again, the variations introduce the KG in either the JSON-LD or Turtle format. For more details, see Meyer et al.[5].

Most important score `combinedF1Score`: See explanation above.

Text2SparqlList (T2S): Given a KG and its description, the LLM shall construct a SPARQL SELECT query corresponding to a given natural language query. The KG is either given completely or partly as a subgraph. Alternatively, some variations provide the KG schema instead of the graph itself, thus testing basic schema read capabilities. The variations are described in table 3 and cover seven different KG datasets. See also table 3 and Meyer et al.[5].

Most important scores `0_combined`, `max_combined`, `0_answerParse` and `max_answerParse`: The answer parse scores evaluate if the first (0) or best (max) given *SPARQL SELECT* query parses syntactically correct. The combined scores for the first (0) or best (max) given *SPARQL SELECT* query are based on the answer parse score and an F1 score

Bench: Please fix all syntax errors of the following RDF in turtle syntax. Try to stick with the original formatting of the RDF given and only change as few characters as necessary. To support automated parsing, please answer with just one markdown fenced code block (start and end with ```) containing the rdf, no other text.

```turtle
...
:anne a foaf:Person ; foaf:firstName "Anne"
...
```

Parsing error message: `at line 7 of <>: Bad syntax (expected '.' or '}' or ']' at end of statement) at ^ in:...`

LLM: A dot (.) is missing

Bench: Please correct your answer following the expected structure(exactly one fenced code block with the RDF, no other text).

LLM:

```turtle
...
:anne a foaf:Person ; foaf:firstName "Anne" .
...
```

Figure 4. Fictive example dialogue for the RdfSyntaxFixList task with a missing dot in Turtle syntax. Some text left out is marked with "..." . The LLM's first answer is missing the expected code block with the fixed Turtle which is corrected in the second answer.

comparing the result sets of given and optimal query. The value is computed as $0.2 \cdot answerParse + 0.8 \cdot f1measure$.

SparqlSyntaxFixingList (SSF): Similar to the RdfSyntaxFixList task, the LLM gets a SPARQL SELECT query with syntax errors and shall return a corrected query. See also Meyer et al.[5].

Most important scores `0_combined`, `max_combined`, `0_answerParse` and `max_answerParse`: See explanation above

The prompts used by the tasks are designed in a way that keeps ambiguity as low as possible. All requirements that we expect the LLM to respect are stated explicitly, e.g., `stick with the original formatting`, or `answer with just one markdown fenced code block ... no other text`. At the same time, we avoid LLM-specific prompt optimization for a fair comparison across different models. An example dialog is shown in fig. 4.

Table 2. Mapping of the task types to the different task aspects and research questions covered. Adopted from Meyer et al.[5]

Research Question:	RQ 3	RQ 3	RQ 4	RQ 5
Task Aspect:	Syntax Read	Syntax Create	Semantic Read	Semantic Create
SPARQL Syntax Fixing (SSF)	x	x	-	-
SPARQL to Answer (S2A)	x	-	x	-
Text to SPARQL (T2S)	-	x	-	x
Text to Answer (T2A)	-	-	-	-

3.5 Benchmark Datasets and KGs Used for SPARQL Task Implementations

There are several benchmark datasets available which contain pairs of *SPARQL SELECT* queries and textual natural language questions for specific knowledge graphs. We selected and implemented benchmark tasks for a couple of current datasets for smaller and bigger knowledge graphs. Only English textual questions were used as we focus on *SPARQL* here and not language capabilities. A total of five tuples, each consisting of a question and corresponding SPARQL query, was manually selected from each dataset. This allows to rerun the tasks more often to reduce the random noise in the results.

Organizational dataset and KG The smallest KG used is an organizational KG[9]. We use it here together with a corresponding dataset[16] of question and SPARQL pairs created by Brei et al.[42].

Triples: 29. Identifiers: human readable.

Organizational Numeric We created an additional variant of the organizational dataset and KG with numerical IRIs (first 3 digits of hash) and same questions.

Triples: 29. Identifiers: numeric.

CoyPu-Mini dataset and KG Brei et al.[42] published another dataset based on a small subset of the CoyPu KG[17]. This sub graph is small enough to fit into the context size of LLMs evaluated here. We added lists of relevant IRIs and schema information.

Triples: 283. Identifiers: human readable.

Beastiary dataset and KG The Beastiary dataset and KG[18] was presented by Kovriguina et al.[17]. It offers for each question a relevant

Table 3. Overview on the different SPARQL-related tasks implemented within LLM-KG-Bench framework

Task Type	Dataset Subset	Task	KG Info Type	KG Info Format
SSF	Syntax-Errors	SSF-LC-QuAD	not needed here	not needed here
S2A	Organizational	S2A-Orga	full KG	Turtle or JSON-LD
T2S	Organizational	T2S-Orga	full KG	Turtle
	Organizational-Numeric	T2S-OrgaNum	full KG + IRIs + Labels	Turtle + table
	Beastiary	T2S-Beast-Graph	KG subset	Turtle
		T2S-Beast-Schema	schema	Turtle
		T2S-Beast-Subschema	schema subset	Turtle
		T2S-Beast-Iris	IRIs	list
	CoyPu-Mini	T2S-CoyPu-Graph	full KG	Turtle or JSON-LD
		T2S-Coypu-Schema	schema	Turtle or JSON-LD
		T2S-Coypu-Iris	IRIs	list
T2A	Organizational	T2A-Orga	full KG	Turtle or JSON-LD

sub graph and a list of IRIs. We derived relevant schema information from these IRI lists.

Triples: 98070. Identifiers: human readable.

SPARQL SELECT Query Syntax Errors We took one question and SPARQL pair from LC-QuAD and derived 5 tests from it by inserting different kinds of syntax errors, one error per test.

Triples: $\sim$ 1billion. Identifiers: numeric.

The list of SPARQL-related benchmark tasks created based on these resources in the *LLM-KG-Bench* framework is given in table 3.

3.6 General Task Characterization

For a structured task characterization, we draw from frameworks from cognitive psychology and educational research. Bloom's Taxonomy[43] was originally developed to classify educational objectives based on the level of cognitive complexity required. It is based on behavioral observations of learning processes and was later revised to better fit modern views of cognitive psychology[44]. Part of that revision includes the separation between what is known from what is done with it, leading to a Knowledge Dimensions framework[44]. It distinguishes between types of knowledge required for different tasks. Relational Complexity Theory[45] was originally developed in developmental and comparative psychology and draws the notion of relational arity from formal systems in logic

Table 4. Task characterization according to Bloom's Taxonomy[43], the Knowledge Dimensions framework[44] and Relational Complexity Theory[45].

Task	Cognitive Process	Knowledge Dimension	Relational Complexity
RDF-related:			
FactExtractStatic	Understand, Create	Conceptual, Procedural	Medium
RdfConnectionExplainState	Understand, Analyze	Conceptual	Medium
RdfFriendCount	Apply	Procedural	Low
RdfSyntaxFixList	Understand, Apply	Factual, Procedural	Low
TurtleSampleGeneration	Understand, Create	Conceptual, Procedural	Medium
SPARQL-related:			
Sparql2AnswerList	Understand, Apply	Conceptual, Procedural	Low
Text2AnswerList	Understand, Apply	Conceptual, Procedural	Low
Text2SparqlList	Understand, Create	Conceptual, Procedural	Low
SparqlSyntaxFixingList	Understand, Apply	Factual, Procedural	Low

and computer science, including relational database theory. It models task difficulty as a function of the number of simultaneous relations that must be processed. Together, these models provide a basis for analyzing the cognitive-inspired and structural demands that the benchmark tasks impose.

The values in the characterization are assigned based on the minimal cognitive and structural requirements expected for successful task completion. Cognitive processes such as *Understand* are assigned when tasks require interpreting given information, while *Apply* corresponds to the execution of known procedures. *Create* is used when tasks involve generating new content and *Analyze* when tasks require decomposing and interpreting relations between elements. For knowledge dimensions, *Factual* knowledge is assigned when specific syntactic or terminological information is needed, *Conceptual* when structural or relational understanding is necessary and *Procedural* when correct application of methods is required. Relational complexity is rated as *Low* when tasks involve isolated or simple binary relations, and as *Medium* when multiple entities and relations must be coordinated simultaneously. Note that not all categories across the frameworks are represented, as the current set of tasks does not span the full theoretical space. The characterization is shown in table 4.

4 Experiment

We utilized the LLM-KG-Bench framework to evaluate a list of 41 LLMs with a selection of 26 tasks. The results containing the full dialog and evaluation data for all this about 45 000 LLM task dialogues(*task*

Evaluation Iterations) is documented and publicly available for further analysis including reevaluation. Whenever new LLMs are of interest, their data can get added to the initial dataset[4]. An example for a statistical analysis on existing data generated is presented by Heim et al.[6].

In the following section we outline the experiment setup and some analysis results from this evaluation.

4.1 Experiment Setup

We adopted the default configuration to define the tasks and models included in this evaluation. As a trade-off between resource usage and confidence, we have decided to conduct 20 iterations for the proprietary LLMs and 50 iterations for the open LLMs. For the 41 LLMs selected and 26 selected evaluation tasks this get multiplied to 45 500 task evaluation iterations.

4.1.1 Selected Tasks The benchmark was executed on the following 26 tasks that are described in section 3.4 and in the code repository:

- RdfSyntaxFixList (3 variations): For Turtle, JSON-LD and N-Triples as graph format
- RdfConnectionExplainStatic (4 variations): For Turtle, JSON-LD, RDF/XML and N-Triples as graph format
- RdfFriendCount (8 variations): For Turtle, JSON-LD, RDF/XML and N-Triples as graph format; 1 and 2 as additional link count
- SparqlSyntaxFixingList (1 variation): For LC-QuAD / Wikidata
- Sparql2AnswerList (1 variation): For Organizational graph
- Text2SparqlList (9 variations): For Organizational, Organizational-Numeric, Coypu-Mini and Beastiary

4.1.2 Selection of Proprietary LLMs To get an overview on the current state-of-the-art proprietary models we selected three long-term high-ranked model families from the Chatbot Arena Leaderboard: OpenAI GPT, Google Gemini and Anthropic Claude. From these families, we selected the current models in various sizes and also included the latest GPT-3.5 for comparability with other results. The selected models together with their context size are shown in table 5.

4.1.3 Selection of Open LLMs We based our selection of state-of-the-art open LLMs on the Open LLM Leaderboard[46] and used the average score over all included benchmarks as our reference value. The selection

criteria were that the model is instruction-finetuned, as required by the task construction of the LLM-KG-Bench framework, has less than 80B parameters because of a limited amount of available hardware resources and is a base model, i.e., not a fine-tuned version of another model. With the latter requirement, we wanted to stick to mature and popularly used LLMs that are not just optimized to achieve a slightly higher score on one or few benchmarks than a base model.

The models fulfilling all our criteria and were among the TOP-4 models based on the average benchmark scores, disregarding models of the same family with lower scores, were Qwen2-72B-Instruct, Meta-Llama-3.1-70B-Instruct, solar-pro-preview-instruct and Phi-3.5-MoE-instruct. Here, we excluded solar-pro-preview-instruct from our selection since it only supports a context length of up to 4096 Tokens and not all prompts of tasks included in the run fitted within this limit. For the remaining three models, we also included all models of their larger model families that matched our requirements, i.e., we also tested all models of the Llama3[47], Qwen2[48,49], and Phi3[50] families fulfilling our requirements.

In addition, we wanted to test open LLMs that are fine-tuned or explicitly optimized on code since they could potentially better understand and produce structured data as required for the tasks included in the LLM-KG-Bench. Here, we consulted the EvalPlus Leaderboard[51] and used the reported models' Mostly Basic Python Programming (MBPP) Benchmark score as our reference value to assess the code-producing quality of the models. Again, we excluded models that were only finetuned versions of a code-finetuned or -optimized base model, had more than 80B parameters or were not instruction-finetuned. Moreover, we only searched for models that are fine-tuned or explicitly optimized on code. Finally, we included the Top-3 models satisfying the criteria in our runs, namely Qwen2.5-Coder-32B-Instruct[48], DeepSeek-Coder-33B-Instruct[52] and OpenCoder-8B-Instruct[53].

After this selection of 37 open LLMs and a first evaluation round end of 2024, we extended the list by the following 4 high ranking LLMs from this families with later release dates until April 2025: deepseek-R1, deepseek-chat-v3-0324, Llama-4.0-Maveric and Qwen3.0-235b-a22b.

Table 5. Details for the models selected for the experiment presented here and the iterations evaluated per model and task combination. The parameter count of proprietary models is not documented and marked with a question mark (?) here. Proprietary model families are OpenAI-GPT, Google-Gemini and Anthropic-Claude.

Family	Model	Parameter Count	Context	Iterations
OpenAI-GPT	ChatGPT 3.5 turbo	?	16k	20
	ChatGPT 4o	?	128k	20
	ChatGPT 4o-mini	?	128k	20
	ChatGPT o1	?	128k	20
	ChatGPT o1-mini	?	128k	20
Google-Gemini	Gemini 2.0 Flash	?	128k–1M	20
	Gemini 1.5 Pro	?	128k–2M	20
	Gemini 1.5 Flash	?	128k–1M	20
Anthropic-Claude	Claude 3.5 Sonnet	?	200k	20
	Claude 3.5 Haiku	?	200k	20
Qwen [48,49]	Qwen2-0.5B-Instruct	0.5B	32k	50
	Qwen2-1.5B-Instruct	1.5B	32k	50
	Qwen2-7B-Instruct	7B	128k	50
	Qwen2-57B-A14B-Instruct	57B (active: 14B)	64k	50
	Qwen2-72B-Instruct	72B	128k	50
	Qwen2.5-0.5B-Instruct	0.5B	32k	50
	Qwen2.5-1.5B-Instruct	1.5B	32k	50
	Qwen2.5-3B-Instruct	3B	32k	50
	Qwen2.5-7B-Instruct	7B	128k	50
	Qwen2.5-14B-Instruct	14B	128k	50
	Qwen2.5-32B-Instruct	32B	128k	50
	Qwen2.5-72B-Instruct	72B	128k	50
	Qwen2.5-Coder-32B-Instruct	32B	128k	50
	Qwen3.0-235b-a22b	235B (active 22B)	41k	50
Meta-Llama [47]	Meta-Llama-3-8B-Instruct	8B	8k	50
	Meta-Llama-3-70B-Instruct	70B	8k	50
	Llama-3.1-8B-Instruct	8B	128K	50
	Llama-3.1-70B-Instruct	70B	128K	50
	Llama-3.2-1B-Instruct	1B	128K	50
	Llama-3.2-3B-Instruct	3B	128K	50
	Llama-3.3-70B-Instruct	70B	128K	50
	Llama-4.0-Maveric	400B (active: 17B)	1M	50
Microsoft-Phi [50]	Phi-3-mini-128k-instruct	3.8B	128k	50
	Phi-3-small-128k-instruct	7B	128k	50
	Phi-3-medium-128k-instruct	14B	128k	50
	Phi-3.5-mini-instruct	3.8B	128k	50
	Phi-3.5-MoE-instruct	42B (active: 6.6B)	128k	50
Infly-OpenCoder [53]	OpenCoder-8B-Instruct	8B	8k	50
Deepseek-ai [52]	deepseek-coder-33b-instruct	33B	16k	50
	deepseek-chat-v3-0324	685B (active: 37B)	64K–164K	50
	deepseek-r1	671B (active: 37B)	64K–164K	50

4.2 Experiment Results

With the configuration described we generated a dataset containing the detailed conversations and the evaluations for 20 to 50 iterations per task

and model. In the following section we show some analysis on tasks, models and the preference on input KG format.

4.2.1 Task Result Examples For each task, a box plot was generated, containing the individual values for the most important score for each model tested. Due to limited space, we show a selection for three tasks in fig. 5. The rest is available in the dataset repository.

Our evaluation of SPARQL-related capabilities is organized around the task types SparqlSyntaxFixing (SSF), Sparql2Answer (S2A) and Text2Sparql (T2S). For SSF and T2S we selected the `max_combined` score, which is 0.2 for syntactically correct but wrong queries. Figures 5a and 5c show several LLMs that seem to have no problem answering with a syntactically correct SPARQL query with scores ≥ 0.2. As fig. 5b shows, several LLMs answered correct with the expected result set for the given *SPARQL SELECT* queries. In fig. 5c, a box plot for the difficult task T2S for the Organizational-Numerical dataset is shown. Several semantically wrong answers were encountered, but Claude 3.5 Haiku & Sonnet, GPT 4o and Gemini 1.5 Pro provide good answers in this special case.

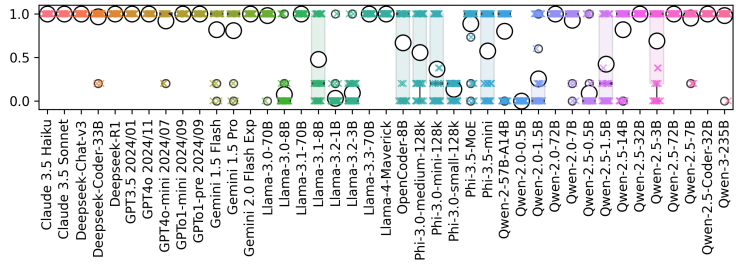
4.2.2 Capability Plots To get a quick overview on LLM models, the LLM-KG-Bench framework can create capability plots.They aggregate result scores for each model into categories according to a configuration file. For the analysis of SPARQL capabilities we use the following categories:

Syn-1: Working with *SPARQL SELECT* queries with first answer. This includes the `0_answerParse` scores (first answer per iteration) from task types SparqlSyntaxFixing (SSF) and Text2Sparql (T2S).

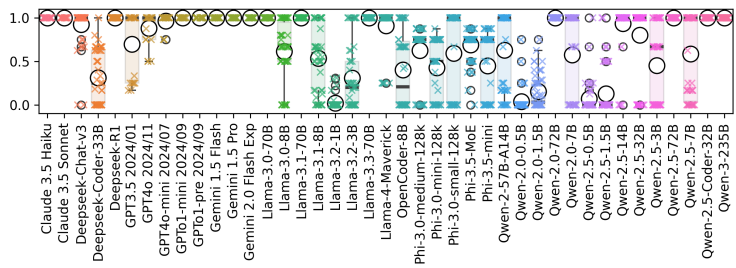
Syn-Max: Working with *SPARQL SELECT* queries with correction possibility in dialog. This includes the `max_answerParse` scores (best answer per iteration) from the same task types as Syn-1.

Sem-R: Working according to semantics of *SPARQL SELECT* queries. This includes the `combinedF1` score of the Sparql2Answer (S2A) task type.

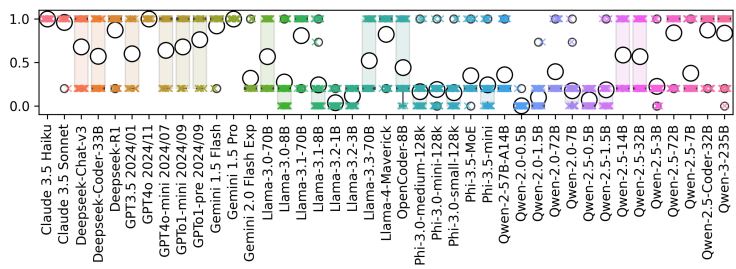
Sem-W-1: Creating semantically correct *SPARQL SELECT* queries with first answer. This includes the `0_combined` scores (first answer per iteration) from Text2Sparql (T2S) task types.



(a) SparqlSyntaxFixing (SSF): max_combined score



(b) Sparql2Answer (S2A): combinedF1 score



(c) Text2Sparql (T2S) for Organizational-Numerical dataset: max_combined score

Figure 5. Selection of box plots for task score values of the LLMs evaluated. The arithmetic mean value is indicated with a circle, individual score values for each task iteration are indicated with crosses.

Sem-W-Max: Creating semantically correct *SPARQL SELECT* queries with correction possibility in dialog. This includes the `max_combined` scores (best answer per iteration) from Text2Sparql (T2S) task types.

Figure 6 shows a selection of typical capability plots for LLMs. Plots for all LLMs evaluated can be found in the supplemental material.

About a quarter of the models have plots similar to fig. 6a showing an almost perfect result with only slight problems with the writing of semantically correct *SPARQL SELECT* queries. For these models the framework would benefit from the inclusion of more complex benchmark datasets.

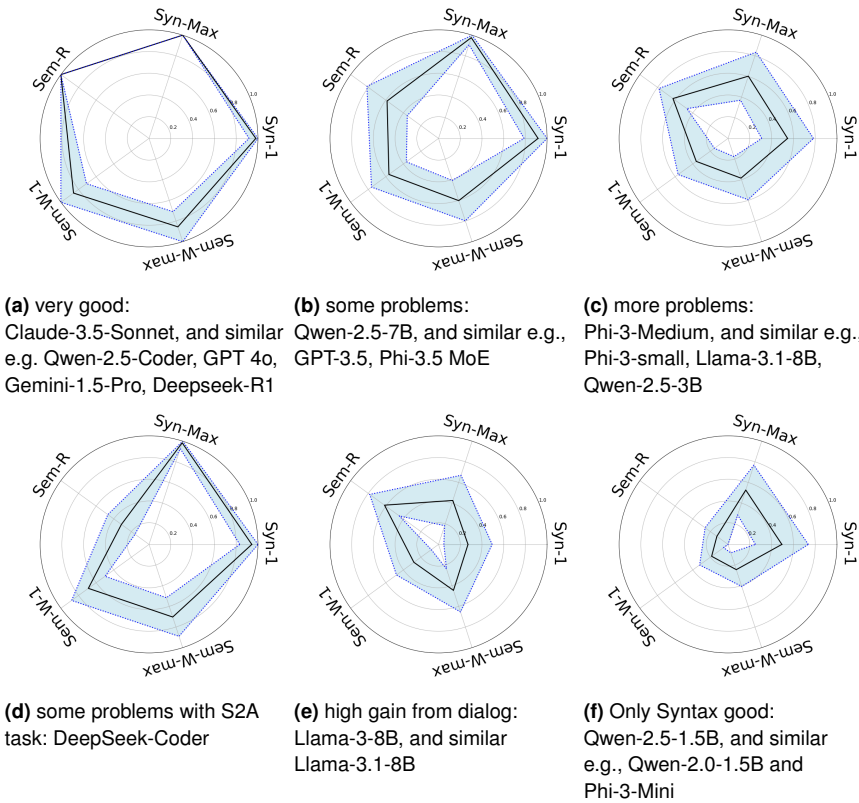
The rest of the models had more problems with answering the tasks resulting in plots like figs. 6b and 6c. Moreover, some very small models from Llama and Qwen family had even worse plots.

Two models, DeepSeek-Coder and OpenCoder revealed strong problems with the Sparql2Answer Task type resulting in low values for the Semantic Read capabilities. This is especially interesting, as their results in the rest of the categories are very good.

Figure 6e shows Llama-3-8B benefits from the correction infos in the dialog. The first answer was often improved during the dialog. Other LLMs evaluated here seem to have fewer benefit from the dialog.

Understanding the syntax of *SPARQL SELECT* queries seems to be much easier for LLMs than the other capabilities. Most models show syntax understanding capability scores that are higher than the rest of the capability scores. Especially small models have a capability plot similar to fig. 6f, showing a good syntax understanding but big problems with the other capabilities.

Aside from the capability compasses, we also aggregated all scores in table 6. Although the values behind these scores have a high variance, the scores can give a first impression of the SPARQL capabilities. Since the results are very close together near the top, it is hard to make out a clear winner. For these runs of experiments, Claude 3.5 Haiku has scored the best results, closely followed by other large commercial models. In order to distill a list of highest scoring models, we chose Claude 3.5 Haiku as our anchor point and conducted pairwise t-tests with each of the models below it to find the point at which a significant difference in performance could be observed. In other words, the first test was conducted between Claude 3.5 Haiku and Claude 3.5 Sonnet, the next test on Claude 3.5 Haiku and Qwen-2.5-Coder-32B, and so forth. We chose $\alpha = 0.05$ for these tests as this is a common default. For this series of experiments, the list of highest scoring models ends with Gemini 1.5 Pro. It is interesting to see that GPTo1-pre 2024/09 and Qwen-2.5-72B did not make the list, while Gemini 1.5 Pro did. This can be attributed to the fact that we chose



(a) very good: Claude-3.5-Sonnet, and similar e.g. Qwen-2.5-Coder, GPT 4o, Gemini-1.5-Pro, Deepseek-R1

(b) some problems: Qwen-2.5-7B, and similar e.g., GPT-3.5, Phi-3.5 MoE

(c) more problems: Phi-3-Medium, and similar e.g., Phi-3-small, Llama-3.1-8B, Qwen-2.5-3B

(d) some problems with S2A task: DeepSeek-Coder

(e) high gain from dialog: Llama-3-8B, and similar Llama-3.1-8B

(f) Only Syntax good: Qwen-2.5-1.5B, and similar e.g., Qwen-2.0-1.5B and Phi-3-Mini

Figure 6. Selection of typical capability plots for *SPARQL-SELECT*-query-related capabilities of the evaluated LLMs. The scores of the models are aggregated along the following capabilities: *Syn-1* = SPARQL Syntax of first answer; *Syn-max* = SPARQL Syntax of best answer per iteration; *Sem-R* = SPARQL Semantic Read; *Sem-W-1* = SPARQL Semantic Write of first answer; *Sem-W-max* = SPARQL Semantic Write of best answer per iteration. The black line shows the mean value, the blue area the standard deviation. Scores range from 0 to best value 1.

a smaller sample size for propietary models, and the higher variance in scores for Gemini 1.5 Pro compared to the other two makes it harder to differentiate it from Claude 3.5 Haiku. In a similar vein, Gemini 1.5 Flash only barely fails this test with a p-value of 0.039, so depending on the α, it could be included or not. However, starting at GPTo1-mini 2024/09 and below, the p value drops down to 0.0001 and lower, meaning that below Gemini 1.5 Flash none of the models would make it into the leader group. The language models that show no significant difference in performance

Table 6. Combined scores of the SPARQL-related capabilities of the LLMs tested. The categories combined into the single score are described in section 4.2.2. Models prefixed with a star (*) show no significant difference in performance compared to Claude 3.5 Haiku.

ModelName	Score	ModelName	Score
***Claude 3.5 Haiku**	0.94 (±0.20)	...	
***Claude 3.5 Sonnet**	0.94 (±0.21)	GPT3.5 2024/01	0.80 (±0.34)
***Qwen-2.5-Coder-32B**	0.93 (±0.22)	Phi-3.5-MoE	0.74 (±0.35)
***GPT4o 2024/11**	0.91 (±0.22)	Qwen-2.5-7B	0.73 (±0.39)
GPTo1-pre 2024/09	0.91 (±0.25)	Deepseek-Coder-33B	0.73 (±0.38)
Qwen-2.5-72B	0.91 (±0.25)	Qwen-2.0-57B-A14B	0.70 (±0.41)
***Gemini 1.5 Pro**	0.90 (±0.27)	Qwen-2.0-7B	0.58 (±0.46)
Deepseek-R1	0.90 (±0.27)	Qwen-2.5-3B	0.57 (±0.44)
Gemini 1.5 Flash	0.90 (±0.28)	OpenCoder-8B	0.55 (±0.45)
Qwen-3-235B	0.89 (±0.28)	Phi-3.0-medium-128k	0.50 (±0.44)
GPTo1-mini 2024/09	0.89 (±0.27)	Phi-3.5-mini	0.49 (±0.44)
Llama-3.1-70B	0.89 (±0.27)	Llama-3.1-8B	0.42 (±0.44)
Llama-4-Maverick	0.87 (±0.29)	Llama-3.0-8B	0.41 (±0.44)
Deepseek-Chat-v3	0.87 (±0.29)	Phi-3.0-mini-128k	0.39 (±0.41)
GPT4o-mini 2024/07	0.87 (±0.29)	Phi-3.0-small-128k	0.39 (±0.43)
Llama-3.0-70B	0.86 (±0.30)	Qwen-2.5-1.5B	0.32 (±0.41)
Llama-3.3-70B	0.85 (±0.31)	Llama-3.2-3B	0.23 (±0.36)
Qwen-2.0-72B	0.85 (±0.30)	Qwen-2.0-1.5B	0.20 (±0.34)
Gemini 2.0 Flash Exp	0.85 (±0.31)	Qwen-2.5-0.5B	0.11 (±0.27)
Qwen-2.5-14B	0.81 (±0.34)	Llama-3.2-1B	0.03 (±0.14)
Qwen-2.5-32B	0.81 (±0.35)	Qwen-2.0-0.5B	0.01 (±0.10)

compared to Claude 3.5 Haiku with our chosen α of 0.05 are highlighted in bold in table 6

4.2.3 Model Preferences on Input Format After running each task both on TTL and JSON-LD serializations of the same knowledge graph, we wanted to find out whether there is a preference for a certain format (i.e., a higher score when using one format over the other). To that end, we conducted several t-tests; both on the task level and the model level. The latter helps us understand if any given model has an overall preference for a certain input format, while the former gives us a better understanding on a per-task basis.

Looking at table 7, we can see that seven models have an overall preference for TTL, while eight models performed overall better on JSON-LD serializations. In contrast to that, 25 models do not have any kind of preference, leaving us with the conclusion that there is no universally preferred format.

Table 7. Result of two-sided t-tests checking for a preference for Turtle (TTL) vs. JSON-LD serialization. Preferences are expected if the confidence interval is at least 95%, bold font indicates 99% or better.

	RdfConnectionExplainAllTasks	RdfFriendCount-Static	RdfFriendCount-1	RdfFriendCount-2	RdfSyntaxFixList	Sparql2AnswerListOrga	Text2AnswerListOrga
Claude 3.5 Haiku	**TTL**	**JSON**	**TTL**	**TTL**	-	-	-
Claude 3.5 Sonnet	-	-	-	-	-	-	-
Deepseek-Coder-33B	JSON	-	**JSON**	**JSON**	**JSON**	-	-
deepseek-chat-v3-0324	-	**JSON**	**TTL**	**TTL**	-	-	-
deepseek-r1	-	-	-	-	-	-	-
GPT3.5 2024/01	-	-	JSON	**JSON**	-	-	**TTL**
GPT4o 2024/11	-	-	-	-	JSON	-	-
GPT4o-mini 2024/07	**TTL**	-	-	TTL	-	-	**TTL**
GPTo1-mini 2024/09	-	-	-	-	-	-	-
GPTo1-pre 2024/09	-	-	-	-	-	-	-
Gemini 1.5 Flash	**TTL**	-	**TTL**	-	-	-	-
Gemini 1.5 Pro	-	-	-	-	-	-	-
Gemini 2.0 Flash Exp	-	-	-	-	-	-	-
Llama-3.3-70B	**JSON**	-	**JSON**	**JSON**	**JSON**	-	-
Meta-Llama-3-70B	**JSON**	-	**JSON**	**JSON**	**JSON**	-	-
Meta-Llama-3-8B	-	**TTL**	-	-	-	-	-
Meta-Llama-3.1-70B	-	-	-	-	**JSON**	**TTL**	-
Meta-Llama-3.1-8B	**JSON**	-	**JSON**	**JSON**	JSON	-	-
Meta-Llama-3.2-1B	**JSON**	-	-	-	**JSON**	-	-
Meta-Llama-3.2-3B	-	**JSON**	**TTL**	**TTL**	**JSON**	-	-
llama-4-maverick	**JSON**	-	**JSON**	**JSON**	-	-	-
OpenCoder-8B	-	-	-	-	-	TTL	-
Phi-3-medium-128k	**TTL**	**TTL**	-	JSON	-	-	TTL
Phi-3-mini-128k	-	-	-	**JSON**	**JSON**	-	-
Phi-3-small-128k	-	**JSON**	-	-	TTL	-	-
Phi-3.5-MoE	**JSON**	-	**JSON**	**JSON**	**JSON**	**TTL**	-
Phi-3.5-mini	-	**TTL**	JSON	-	**JSON**	**TTL**	-
Qwen2-0.5B	-	-	-	JSON	-	-	-
Qwen2-1.5B	**JSON**	**TTL**	**JSON**	**JSON**	**JSON**	-	-
Qwen2-7B	-	**JSON**	-	-	-	-	TTL
Qwen2-57B-A14B	TTL	-	-	-	**JSON**	TTL	**TTL**
Qwen2-72B	-	-	-	-	**JSON**	-	-
Qwen2.5-0.5B	-	-	-	-	-	-	-
Qwen2.5-1.5B	-	-	-	-	**JSON**	-	-
Qwen2.5-14B	-	**TTL**	-	-	-	-	TTL
Qwen2.5-32B	TTL	-	**TTL**	**TTL**	**JSON**	**JSON**	-
Qwen2.5-3B	-	**JSON**	-	-	**JSON**	-	-
Qwen2.5-7B	**TTL**	**JSON**	-	-	-	**TTL**	**TTL**
Qwen2.5-72B	-	**JSON**	JSON	**JSON**	-	-	**TTL**
Qwen2.5-Coder-32B	**TTL**	-	**TTL**	**TTL**	-	-	-
Qwen-3-235B	-	-	-	-	-	-	-
All Models		-	**JSON**	**JSON**	**JSON**	TTL	**TTL**

Looking at the table 7 as a whole, we can see that preferences for one format over the other appear roughly equally distributed. However, there are some clusters like the Llama family of models preferring JSON, or Qwen leaning more towards TTL.

This might have to do with the amount of training data that covers TTL versus that which covers JSON. Unfortunately, as not all training data is publicly available, we cannot investigate this further.

When looking at the global preferences per task, we find that the Text2Answer and Sparql2Answer tasks appear to be easier to solve when operating on a TTL-formatted KG. In contrast to that, counting friends seems to work better for JSON KGs. This could come from the fact that, in order to find the person with the most incoming connections, the model needs to be able to parse the knowledge graph and have some notion of what an array is and how to count. In contrast, extracting facts (or answers in that case) may profit from having the desired answers close to the question subjects , which is a strength of TTL.

Looking at the table, it is important to note that a model having no preference could mean one of these two things:

1. The model performs equally well on both formats and has no problem solving the task, no matter what. This is the case for Gemini 1.5 Pro, GPTo1 and some others.
2. The model has big problems solving the task. This is, for instance, the case for Qwen2 0.5B.

The former is a desired state, as this relieves the user of the burden of having to choose a serialization that maximizes the quality of the results, while models having preferences in certain areas indicate that they are still growing towards that state. To find out whether a model falls into the former or the latter category, one should cross-reference the table with the corresponding capability compass.

5 Discussion and Outlook

Coming back to the research questions, the LLM-KG-Bench framework helps answering them.

RQ1: We showed here and in previous works[5,36] the possibilities for automated evaluation of KGE-related capabilities. The LLM-KG-Bench framework is open for further development to support additional KGE task areas.

RQ2: In the analysis of the experiment result and in previous work[5,6,35,36] we could identify ways to distinguish the KGE-related

capabilities of different LLMs. For *SPARQL SELECT* queries we propose here the scores calculated in table 6 as guidance.

RQ3: At least for the tasks evaluating the understanding of *SPARQL SELECT* queries syntax seems to be possible most of the time for most evaluated LLMs as shown in fig. 5a and in Meyer et al.[5]. Further work is needed to check this claim for more complex queries and errors.

RQ4: Figure 5b shows at least some LLMs seem to have no problems with working according to the semantics of *SPARQL SELECT* queries.

RQ5: At least for the evaluated tasks, the creation of semantically correct *SPARQL SELECT* queries seems to be possible most of the time for some LLMs, when looking at fig. 5c and the good capability plot type in fig. 6a. However, this also needs to be checked for other more complex queries. Moreover, it is important to consider that the evaluation is currently only done based on the answer set, which does not necessarily mean that the semantic of the *SPARQL SELECT* query is correct.

Still there are some things to keep in mind. Some of the tasks, like Sparql2Answer or RdfFriendCount, can be answered with graph databases much more efficiently and reliable. But the measuring of this RDF-related capabilities is of a more fundamental scientific interest. On the other side, the focus on this basis tasks induce a risk of overestimating the KGE-related capabilities of LLMs, as real world scenarios are often of a more complex nature.

The answer quality of LLMs can vary even on small changes in prompt or task. We tested only with a limited amount of tasks and task entries.

The automated evaluation could miss important aspects of the answer. For example, the *SPARQL SELECT* queries generated were tested only extensionally on the result set generated, which does not always imply that the query was well formulated to match the expected semantics.

Although we tried to hide the benchmark data, we can not guarantee no LLM is trained with them, as we want to keep the benchmark and its results open and reproducible.

Further work is needed to include more datasets into the framework, extend the list of tasks to cover more aspects related to Knowledge Graphs and extend the analytical capabilities of the framework. The framework and the evaluation data generated is publicly available to enable contribution.

The framework would benefit from more datasets and especially more evaluation task classes. We see here on the one hand additional topics like OWL, RML, SHACL, and on the other hand more complex tasks and analyses like nested SPARQL queries or even the interaction with real graph databases.

As the number of LLMs published and the size of datasets generated with the framework grow, new ways to check the results are needed. Maybe one could utilize another level of LLMs to check on the automated evaluation results for details missed and highlight these findings for manual inspection. This way, the evaluation could remain deterministic and reproducible, but we gain another level of checks and maybe even insight.

Acknowledgements

We thank DFKI for running the experiments involving open LLMs on their GPU cluster. This work was partially supported by grants from the German Federal Ministry of Education and Research (BMBF) to the projects ScaleTrust (16DTM312D) and KupferDigital2 (13XP5230L) as well as from the German Federal Ministry for Economic Affairs and Climate Action (BMWK) to the KISS project (01MK22001A) and CoyPu project (01MK21007A) and from the German Federal Ministry of Transport and Digital Infrastructure (BMDV) to the project ADA (19F2190B).

Supplemental material

The framework source code and generated result datasets are publicly available:

- LLM-KG-Bench Framework Code:
  https://github.com/AKSW/LLM-KG-Bench/tree/v3.0.2
  (DOI: 10.5281/zenodo.18024503)
- Generated Dataset:
  https://github.com/AKSW/LLM-KG-Bench-v3-0-results/tree/2025-SWJ (DOI: 10.5281/zenodo.18017450)

Notes

1. Leaderboard Chatbot Arena: https://huggingface.co/spaces/lmarena-ai/chatbot-arena-leaderboard
2. Description MMLU on HELM: https://crfm.stanford.edu/2024/05/01/helm-mmlu.html
3. Leaderboard Open LLM: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard
4. Leaderboard HELM: https://crfm.stanford.edu/helm/lite/latest/#/leaderboard

5. Leaderboard Big Code Models: https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard
6. Leaderboard EvalPlus: https://evalplus.github.io/leaderboard.html
7. Updated literature overview: https://github.com/zjukg/KG-LLM-Papers
8. Repository OpenAI python: https://github.com/openai/openai-python
9. Description OpenAI REST API: https://platform.openai.com/docs/api-reference/chat
10. Repository Google Connector for LLMs: https://github.com/google-gemini/generative-ai-python
11. Description Google API: https://ai.google.dev/api
12. Anthropic REST API description: https://docs.anthropic.com/en/api
13. Repository Anthropic Python: https://github.com/anthropics/anthropic-sdk-python
14. Webpage vLLM: https://docs.vllm.ai
15. Tasks implemented in LLM-KG-Bench v3.0.2: https://github.com/AKSW/LLM-KG-Bench/tree/v3.0.2/LlmKgBench/tasks
16. Repository LMs4Text2SPARQL Dataset: https://github.com/AKSW/LMs4Text2SPARQL/tree/main/datasets
17. Project page CoyPu: https://coypu.org/
18. Repository Beastiary dataset and KG: https://github.com/danrd/sparqlgen

References

1. Srivastava A, Rastogi A, Rao A et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research* 2023; .
2. Liang P, Bommasani R, Lee T et al. Holistic evaluation of language models. *Transactions on Machine Learning Research* 2023; .
3. Chiang WL, Zheng L, Sheng Y et al. Chatbot arena: An open platform for evaluating LLMs by human preference. In Salakhutdinov R, Kolter Z, Heller K et al. (eds.) *Proceedings of the 41st International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, volume 235. PMLR, pp. 8359–8388.
4. Meyer LP, Frey J, Heim D et al. *LLM-KG-Bench 3.0: A Compass for SemanticTechnology Capabilities in the Ocean of LLMs*. Springer Nature Switzerland. ISBN 9783031945786, 2025. pp. 280–296. DOI:10.1007/978-3-031-94578-6_16.
5. Meyer LP, Frey J, Brei F et al. Assessing SPARQL capabilities of large language models. In Vakaj E, Iranmanesh S, Stamartina R et al. (eds.) *Proceedings of the 3rd International Workshop on Natural Language Processing for Knowledge Graph Creation co-located with 20th International Conference on Semantic Systems (SEMANTiCS 2024)*, *CEUR Workshop Proceedings*, volume 3874. pp. 35–53. URL https://ceur-ws.org/Vol-3874/paper3.pdf.
6. Heim D, Meyer LP, Schröder M et al. How do scaling laws apply to knowledge graph engineering tasks? the impact of model size on large language model performance. In *ESWC*

2025 Workshops and Tutorials Joint Proceedings, *CEUR Workshop Proceedings*, volume 3977. URL https://ceur-ws.org/Vol-3977/elmke-3.pdf.

7. Bai G, Liu J, Bu X et al. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, p. 7421–7454. DOI:10.18653/v1/2024.acl-long.401.
8. Pan S, Luo L, Wang Y et al. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 2024; DOI: 10.1109/TKDE.2024.3352100.
9. Meyer LP, Stadler C, Frey J et al. Llm-assisted knowledge graph engineering: Experiments with chatgpt. In Zinke-Wehlmann C and Friedrich J (eds.) *First Working Conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow (AITomorrow) 2023*. Informatik aktuell, pp. 101–112. DOI:10.1007/978-3-658-43705-3_8.
10. Pan JZ, Razniewski S, Kalo JC et al. Large language models and knowledge graphs: Opportunities and challenges. *Transactions on Graph Data and Knowledge* 2023; 1(1): 2:1–2:38. DOI:10.4230/TGDK.1.1.2.
11. Mihindukulasooriya N, Tiwari S, Enguix CF et al. Text2KGBench: A benchmark for ontology-driven knowledge graph generation from text. In Payne TR, Presutti V, Qi G et al. (eds.) *The Semantic Web – ISWC 2023*. Cham: Springer Nature Switzerland. ISBN 978-3-031-47243-5, pp. 247–265. DOI:10.1007/978-3-031-47243-5_14.
12. Zhu Y, Wang X, Chen J et al. Llms for knowledge graph construction and reasoning: recent capabilities and future opportunities. *World Wide Web* 2024; 27(5). DOI:10.1007/s11280-024-01297-w.
13. Petroni F, Rocktäschel T, Riedel S et al. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics. DOI:10.18653/v1/d19-1250.
14. Singhania S, Nguyen TP and Razniewski S. Lm-kbc: Knowledge base construction from pre-trained language models. In Singhania S, Nguyen TP and Razniewski S (eds.) *Proceedings of the Semantic Web Challenge on Knowledge Base Construction from Pre-trained Language Models 2022co-located with the 21st International Semantic Web Conference (ISWC2022)*, *CEUR Workshop Proceedings*, volume 3274. pp. 1–10. URL https://ceur-ws.org/Vol-3274/paper1.pdf.
15. Kalo JC, Razniewski S, Zhang B et al. Lm-kbc 2025: 4th challenge on knowledge base construction from pre-trained language models. In Razniewski S, Kalo JC, Islakoğlu D et al. (eds.) *Joint Proceedings of the 3rd Workshop on Knowledge Base Construction from Pre-Trained Language Models and the 4th Challenge on Language Models for Knowledge Base Construction (KBC-LM+LM-KBC 2025)co-located with the 24th International Semantic Web Conference (ISWC 2025)*, *CEUR Workshop Proceedings*, volume 4041. URL https://ceur-ws.org/Vol-4041/paper7.pdf.
16. Usbeck R, Röder M, Hoffmann M et al. Benchmarking question answering systems. *Semantic Web* 2019; 10(2): 293–304. DOI:10.3233/sw-180312.
17. Kovriguina L, Teucher R, Radyush D et al. SPARQLGEN: One-shot prompt-based approach for SPARQL query generation. In *International Conference on Semantic Systems*, *CEUR*

Workshop Proceedings, volume 3526. CEUR-WS.org. URL https://ceur-ws.org/Vol-3526/paper-08.pdf.

18. Zahera HM, Ali M, Sherif MA et al. Generating sparql from natural language using chain-of-thoughts prompting. In *SEMANTiCS 2024: Knowledge Graphs in the Age of Language Models and Neuro-Symbolic AI*. Amsterdam, Netherlands. DOI:10.3233/ssw240028.
19. Marx E, do Carmo PV, Gôlo M et al. Preface of the first international text2sparql challenge (text2sparql'25). In Tramp S, Gôlo M, Marx E et al. (eds.) *Proceedings of the First International TEXT2SPARQL Challenge, Co-Located with Text2KG at ESWC25, June 01, 2025, Portorož, Slovenia*, *CEUR Workshop Proceedings*, volume 4094. URL https://ceur-ws.org/Vol-4094/preface.pdf.
20. Hofer M, Frey J and Rahm E. Towards self-configuring knowledge graph construction pipelines using llms - a case study with rml. In *Fifth International Workshop on Knowledge Graph Construction @ ESWC2024*, *CEUR Workshop Proceedings*, volume 3718. CEUR-WS.org. URL https://ceur-ws.org/Vol-3718/paper6.pdf.
21. Shorten C, Pierse C, Smith TB et al. Structuredrag: Json response formatting with large language models, 2024. URL https://arxiv.org/abs/2408.11061. 2408.11061.
22. Rangel JC, de Farias TM, Sima AC et al. Sparql generation: an analysis on fine-tuning openllama for question answering over a life science knowledge graph, 2024. To appear in Proceedings of SWAT4HCLS 2024: Semantic Web Tools and Applications for Healthcare and Life Sciences, 2402.04627.
23. Bustamante D and Takeda H. Sparql generation with entity pre-trained gpt for kg question answering, 2024. 2402.00969.
24. Avila CVS, Vidal VMP, Franco W et al. Experiments with text-to-sparql based on chatgpt. In *18th IEEE International Conference on Semantic Computing, ICSC 2024, Laguna Hills, CA, USA, February 5-7, 2024*. IEEE, pp. 277–284. DOI:10.1109/ICSC59802.2024.00050.
25. Li Z, Fan S, Gu Y et al. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering, 2024. DOI:10.1609/aaai.v38i17.29823. 2308.12060.
26. Diallo PAKK, Reyd S and Zouaq A. A comprehensive evaluation of neural sparql query generation from natural language questions, 2024. 2304.07772.
27. Hirigoyen R, Zouaq A and Reyd S. A copy mechanism for handling knowledge base elements in SPARQL neural machine translation. In He Y, Ji H, Li S et al. (eds.) *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*. Online only: Association for Computational Linguistics, pp. 226–236. URL https://aclanthology.org/2022.findings-aacl.22.
28. Pliukhin D, Radyush D, Kovriguina L et al. Improving subgraph extraction algorihtms for one-shot sparql query generation with large language models. In *1st Scholarly QALD Challenge 2023 and 4th SeMantic Answer Type, Relation and Entity Prediction Tasks Challenge 2023*, *CEUR Workshop Proceedings*, volume 3592. CEUR-WS.org. URL https://ceur-ws.org/Vol-3592/paper6.pdf.
29. Lehmann J, Ferré S and Vahdati S. Language models as controlled natural language semantic parsers for knowledge graph question answering. In Gal K, Nowé A, Nalepa GJ et al. (eds.) *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent*

Systems (PAIS 2023), *Frontiers in Artificial Intelligence and Applications*, volume 372. IOS Press, pp. 1348–1356. DOI:10.3233/FAIA230411.

30. Diefenbach D, Tanon TP, Singh KD et al. Question answering benchmarks for wikidata. In Nikitina N, Song D, Fokoue A et al. (eds.) *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017*, *CEUR Workshop Proceedings*, volume 1963. CEUR-WS.org. URL https://ceur-ws.org/Vol-1963/paper555.pdf.
31. Bordes A, Usunier N, Chopra S et al. Large-scale simple question answering with memory networks, 2015. 1506.02075.
32. Dubey M, Banerjee D, Abdelkawi A et al. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *Proceedings of the 18th International Semantic Web Conference (ISWC)*. Springer. DOI:10.1007/978-3-030-30796-7_5.
33. Trivedi P, Maheshwari G, Dubey M et al. Lc-quad: A corpus for complex question answering over knowledge graphs. In d'Amato C, Fernández M, Tamma VAM et al. (eds.) *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*, *Lecture Notes in Computer Science*, volume 10588. Springer, pp. 210–218. DOI:10.1007/978-3-319-68204-4_22.
34. Banerjee D, Nair PA, Kaur JN et al. Modern baselines for sparql semantic parsing. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22, ACM. DOI:10.1145/3477495.3531841.
35. Frey J, Meyer LP, Brei F et al. Assessing the evolution of LLM capabilities for knowledge graph engineering in 2023. In *The Semantic Web: ESWC 2024 Satellite Events*. Springer Nature Switzerland. ISBN 9783031789526, pp. 51–60. DOI:10.1007/978-3-031-78952-6_5.
36. Frey J, Meyer L, Arndt N et al. Benchmarking the abilities of large language models for RDF knowledge graph creation and comprehension: How well do llms speak turtle? In Alam M and Cochez M (eds.) *Proceedings of the Workshop on Deep Learning for Knowledge Graphs (DL4KG 2023) co-located with the 21th International Semantic Web Conference (ISWC 2023), Athens, November 6-10, 2023*, *CEUR Workshop Proceedings*, volume 3559. CEUR-WS.org. URL https://ceur-ws.org/Vol-3559/paper-3.pdf.
37. Perevalov A and Both A. Text-to-sparql goes beyond english: Multilingual question answering over knowledge graphs through human-inspired reasoning. In Tramp S, Gôlo M, Marx E et al. (eds.) *Proceedings of the First International TEXT2SPARQL Challenge, Co-Located with Text2KG at ESWC25, June 01, 2025, Portorož, Slovenia*, *CEUR Workshop Proceedings*, volume 4094. pp. 77–93. DOI:10.48550/arXiv.2507.16971. URL https://ceur-ws.org/Vol-4094/paper6.pdf.
38. Brei F, Bühmann L, Frey J et al. ARUQULA - an LLM based Text2SPARQL approach using ReAct and knowledge graph exploration utilities. In Tramp S, Gôlo M, Marx E et al. (eds.) *Proceedings of the First International TEXT2SPARQL Challenge, Co-Located with Text2KG at ESWC25, June 01, 2025, Portorož, Slovenia*, *CEUR Workshop Proceedings*, volume 4094. pp. 49–63. DOI:10.48550/arXiv.2510.02200. URL https://ceur-ws.org/Vol-4094/paper4.pdf.
39. Liu S, Semnani S, Triedman H et al. Spinach: Sparql-based information navigation for challenging real-world questions. In *Findings of the Association for Computational*

Linguistics: EMNLP 2024. Association for Computational Linguistics, pp. 15977–16001. DOI:10.18653/v1/2024.findings-emnlp.938.

40. Meyer LP, Frey J, Junghanns K et al. Developing a scalable benchmark for assessing large language models in knowledge graph engineering. In Keshan N, Neumaier S, Gentile AL et al. (eds.) *Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems (SEMANTICS 2023)*, *CEUR Workshop Proceedings*, volume 3526. CEUR-WS.org. URL https://ceur-ws.org/Vol-3526/paper-04.pdf.
41. Kwon W, Li Z, Zhuang S et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
42. Brei F, Frey J and Meyer LP. Leveraging small language models for text2sparqltasks to improve the resilience of ai assistance. In Holze J, Tramp S, Martin M et al. (eds.) *Proceedings of the Third International Workshop on Linked Data-driven Resilience Research 2024 (D2R2'24), colocated with ESWC 2024*, *CEUR Workshop Proceedings*, volume 3707. CEUR-WS.org. URL https://ceur-ws.org/Vol-3707/D2R224_paper_5.pdf.
43. Bloom BS. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook 1: Cognitive Domain*. New York: Longman, 1956.
44. Anderson LW and Krathwohl DR. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman, Inc., 2001.
45. Halford GS, Wilson WH and Phillips S. Processing capacity defined by relational complexity: Implications for comparative, developmental, and cognitive psychology. *Behavioral and Brain Sciences* 1998; 21(6): 803–831.
46. Fourrier C, Habib N, Lozovskaya A et al. Open llm leaderboard v2. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard, 2024.
47. Grattafiori A et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783. 2407.21783.
48. Hui B et al. Qwen2.5-coder technical report, 2024. URL https://arxiv.org/abs/2409.12186. 2409.12186.
49. Yang A et al. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671. 2407.10671.
50. Abdin M et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219. 2404.14219.
51. Liu J, Xia CS, Wang Y et al. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems* 2024; 36.
52. Guo D, Zhu Q, Yang D et al. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *CoRR* 2024; abs/2401.14196. DOI:10.48550/arXiv.2401.14196.
53. Huang S, Cheng T, Liu JK et al. Opencoder: The open cookbook for top-tier code large language models. *CoRR* 2024; abs/2411.04905. DOI:10.48550/arXiv.2411.04905.