

# Graf von Data: A Knowledge Graph Question Answering Agent for Organisational Usage

Rene Dorsch<sup>1,\*</sup>, Daniel Henselmann<sup>1,\*</sup> and Andreas Harth<sup>1,2</sup>

<sup>1</sup>Fraunhofer IIS, Nordostpark 84, 90411 Nuremberg, Germany

<sup>2</sup>Friedrich-Alexander-Universität Erlangen-Nürnberg, Lange Gasse 20, 90403 Nuremberg, Germany

## Abstract

Large Language Model-based (LLM) systems have advanced Knowledge Graph-based Question Answering (KGQA) but overlook organisational demands for transparency, control, and flexible deployment. We present Graf von Data (GvD), an LLM-augmented agent for KGQA, designed to meet key organisational demands. GvD translates natural language questions into SPARQL queries by iteratively exploring the knowledge graph (KG) using the ReAct framework and a set of three actions – search, describe, and query – requiring only a SPARQL endpoint with text search capabilities. Unlike other approaches, GvD is graph-agnostic and uses an open-weight LLM. We evaluate GvD in the Text2SPARQL challenge, demonstrating near state-of-the-art performance on the Corporate KG benchmark. In addition to quantitative results, we conduct a qualitative comparison against the challenge competitors, showing that GvD thrives under the organisational demands.

## Keywords

Large Language Model, RDF, Knowledge Graph, Question Answering, Knowledge Graph Question Answering

## 1. Introduction

Connecting data across various sources is a crucial task for many organisations, enabling process optimisation and informed decision-making. Industrial organisations [1, 2, 3], technology organisations [4], and government institutions [5] employ knowledge graphs (KGs) due to their flexible structure and schema to connect data, thereby simplifying data integration.

However, accessing and querying KGs remains challenging for non-technical users in organisations [6] due to the specialised expertise required for crafting SPARQL queries. For non-technical users, Knowledge Graph Question Answering (KGQA) approaches have been developed to offer an interface for natural language questions. Current approaches rely on Large Language Models (LLMs) to interact with the user and retrieve answers from the KG.

Nevertheless, existing LLM-based approaches (see Section 2) for KGQA are not developed with the demands of organisations in mind. The demands are [7, 8]:

- **Transparent Answers Generation:** Existing LLM-based approaches often lack transparency in their generated answers, which is a concern for organisational applications where trustworthy and verifiable information is necessary to support decision-making.
- **Organizational Control:** Current LLM-based approaches frequently rely on closed-weight models of OpenAI, Anthropic, or Google. However, this may lead to vendor lock-in and/or high operational costs, limiting an organisation’s autonomy and strategic control over technology adoption and expenditure. Furthermore, it may lead to privacy concerns as data is sent to a third party.
- **Flexible Deployment:** Organisations often need solutions that can be served on-demand or within the cloud and can be applied to various KGs without requiring excessive configuration or complex infrastructure.

*First International TEXT2SPARQL Challenge, Co-Located with Text2KG at ESWC25, June 01, 2025, Portorož, Slovenia.*

\*Corresponding author.

✉ rene.dorsch@iis.fraunhofer.de (R. Dorsch); daniel.henselmann@iis.fraunhofer.de (D. Henselmann); andreas.harth@iis.fraunhofer.de (A. Harth)

ORCID 0000-0001-6857-7314 (R. Dorsch); 0000-0001-6701-0287 (D. Henselmann); 0000-0002-0702-510X (A. Harth)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

To satisfy the additional demands of organisations, this paper introduces Graf von Data (GvD), an LLM-augmented agent. Users can ask questions in natural language, which GvD uses to explore and retrieve answers from the KG and then provide an answer. Users not only receive natural language answers but also corresponding executable SPARQL queries, supporting transparency. GvD’s agent-based approach enables iterative gathering and reasoning over KG information stored within a trace that is available to users, thereby also enhancing the transparency of query generation. Furthermore, GvD employs a KG-agnostic approach, requiring only a SPARQL endpoint and no prior assumption about the KG. Thus, it supports flexible deployments. By prioritising medium-sized, open-weight models, GvD supports deployment within trusted environments (on-premise or in the cloud), reduces operational costs, and avoids vendor lock-in.

GvD achieved the highest  $F_1$  score (44.4%) for the Corporate KG benchmark in the quantitative evaluation of the Text2SPARQL challenge<sup>1</sup>. A qualitative comparison with the competitors also shows that GvD is the only competitor addressing all organisational demands.

The remainder of this paper is structured as follows. Section 2 presents related work on LLM-augmented KGQA. Section 3 introduces GvD. Section 4 presents the experimental setup to evaluate GvD within the Text2SPARQL challenge. Section 5 introduces the competitors of the Text2SPARQL challenge. Section 6 discusses the results from the experiments and highlights differences between our approach and the other participants of the Text2SPARQL challenge. Section 7 concludes the paper with limitations and directions for future research.

## 2. Related Work

The emergence of LLMs significantly impacted KG question answering as LLM-augmented approaches outperformed the results of prior approaches (e.g., [5]), becoming state-of-the-art. Multiple strategies for LLM-augmented systems for KG question answering exist, which we distinguish in the following. We consider approaches [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22] that generate a SPARQL query for a natural language question and find the question’s answer by executing the query, resulting in comprehensible answers based on facts.

Fine-tuning trains an LLM on a specific KG, inherently adding its content to the LLM’s knowledge. Approaches using fine-tuning [14, 19] require expensive resources to adjust the weights, are optimized for individual KGs, and require fine-tuning after each update of the KG.

Approaches [14, 16, 17, 21, 22] implementing few-shot learning [23] provide example questions-query pairs to the LLM.

Retrieval-Augmented Generation (RAG) [24] allows an LLM to consider (up-to-date) information that is not inherent to the model by retrieving and considering additional resources after a question is asked. Actions, often referred to as tools, can be used to incorporate various resources. An action usually corresponds to an API call, providing information based on the question and receiving context information from the KG. Actions that implement RAG retrieve context information from an external knowledge source out of a given, finite set of information items. RAG actions in related work are entity/relation linking (retrieving IDs out of all IDs in the KG) [9, 10, 11, 12, 13, 14, 15, 16, 20, 21, 22], subgraph retrieval for an entity (out of the entire KG) [10, 11, 12, 13, 20, 21], subgraph retrieval specifically for a property (out of the entire KG) [10, 20], path discovery (out of all paths in the KG) [13], and SPARQL query execution (retrieving a combination of entities/relations from the KG) [10, 11, 12].

Other actions return context information that is not retrieved from an external knowledge source but instead generated based on the question’s information. Such actions in related work are entity/relation extraction from the question [9, 14, 15, 16], shape generation [18], SPARQL query generation [9], and SPARQL query validation [15, 16, 17, 22].

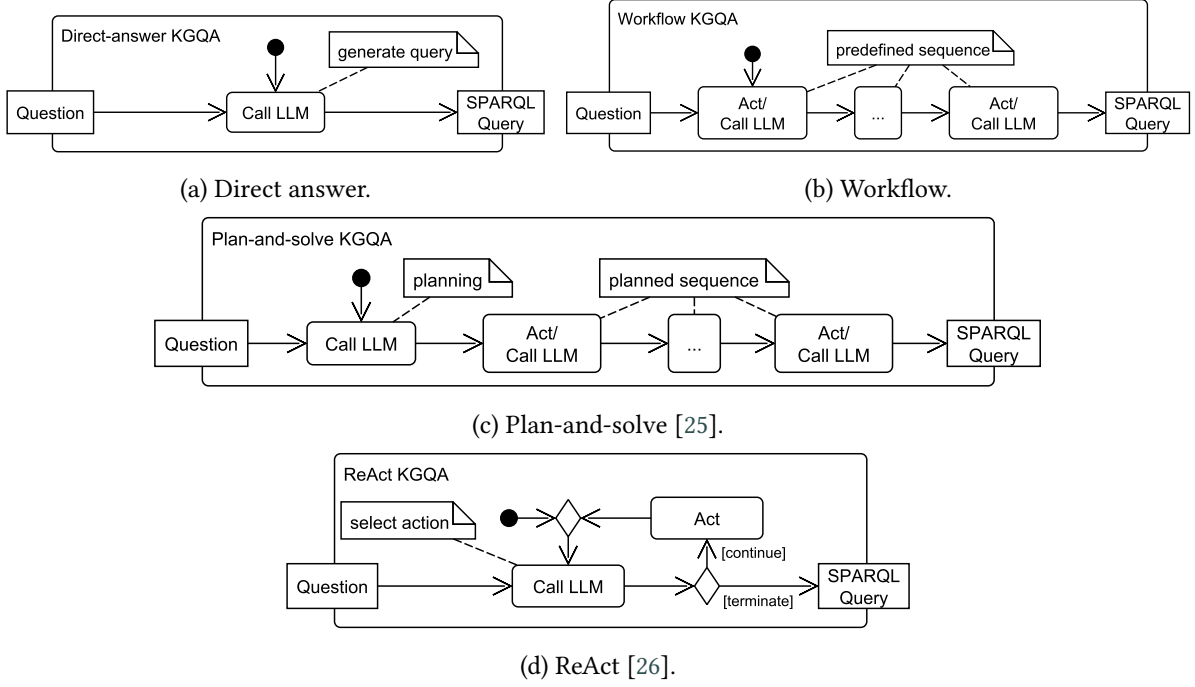
Besides actions, calling the LLM is a potential step in the question-answering process. In addition to the task of generating the SPARQL query, intermediate tasks to translate the question [18, 21], extract entities/relations [18], identify relevant classes and properties from the KG [15], filter retrieved

---

<sup>1</sup>See <https://text2sparql.aksw.org/>

subgraphs [10], and select the best SPARQL query among several candidates [9, 11] have been used.

Action and LLM calls may be combined in different ways. Figure 1 shows four strategies. Approaches [19] that find an *answer directly* execute a single LLM call (see Figure 1a) without considering up-to-date context information from the KG. *Workflow* approaches [9, 11, 14, 15, 16, 17, 18, 21] execute a predefined sequence of action and LLM calls (see Figure 1b). *Plan-and-solve* [25] approaches [22] initially call the LLM to draft a planned sequence of then executed action/LLM calls (see Figure 1c). *ReAct* [26] approaches [10, 12, 13, 20] iteratively alternate between calling the LLM to select the next action and executing the selected action, until the LLM decides to terminate, usually because the SPARQL query was found (see Figure 1d).



**Figure 1:** KG question answering (QA) strategies as UML 2.5 [27] activity diagram.

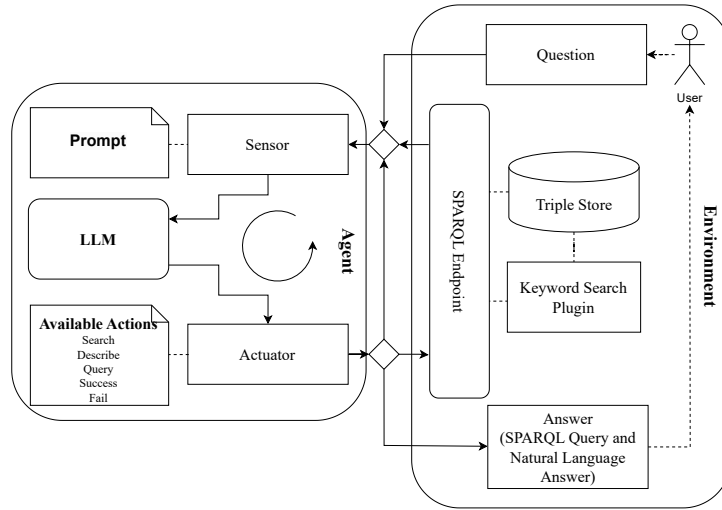
The current state-of-the-art for question answering on Wikidata is held by SPINACH [10] using a ReAct approach with actions for entity/relation linking, subgraph retrieval for an entity (enhanced with an LLM call to filter retrieved subgraphs), subgraph retrieval specifically for a property, and SPARQL query execution. SPINACH was developed explicitly for Wikidata, using three APIs provided by Wikidata and profiting from the LLM’s inherent knowledge on Wikidata entities from its training. In contrast, GvD requires only a SPARQL endpoint (with search extension) as API, supporting KG-agnostic operation.

### 3. Graf von Data Architecture

We provide a high-level overview of the GvD system and a description of its behaviour based on an example in Section 3.1. Afterwards, we provide a detailed description of the actions, prompts, and LLM selection for the agent in Section 3.2.

#### 3.1. High Level Overview

GvD operates within a structured environment composed of two elements (see Figure 2): the agent and the environment. These components interact iteratively to process natural language questions and produce answers based on knowledge graph data.



**Figure 2:** Graf von Data architecture.

- **Agent:** The agent orchestrates the question-answering process. It consists of three subcomponents: (i) an LLM, which processes input to select actions; (ii) a sensor, which parses and appends input data to the prompt; and (iii) an actuator, which executes the selected actions.
- **Environment:** The environment includes a User and a SPARQL endpoint. The SPARQL endpoint serves as the interface between the agent and the KG. The endpoint provides a standardised access point to the triple store and the keyword search, enabling GvD to work with different KGs without modification or additional infrastructure.

Figure 3 illustrates how GvD may handle a question of a procurement officer (User) that wants to identify how many suppliers offer alternative compatible products for a specific part: the K367 Strain Encoder. Thus, he asks GvD to retrieve the information within the KG for the following question: “How many suppliers can deliver alternative compatible products for the K367 Strain Encoder?” The GvD agent processes the question through the following iterative steps:

1. **Initial Search (Step 1):** The agent identifies the key term “K367 Strain Encoder” from the user question. Afterwards, the agent calls the search action for the term to identify related URIs from the KG.
2. **Entity Expansion (Step 2-3):** Once the relevant URIs are located, the agent retrieves their RDF description to identify compatible products and associated relationships (ns2:compatibleProduct).
3. **Intermediate Steps (Step ...):** The agent extends the search using additional keywords and/or explores connected entities to discover a solution path.
4. **Final Query Step (Step n-1):** After gathering all relevant data (compatible products and their suppliers), the agent composes and executes a SPARQL query to count the number of unique suppliers.
5. **Completion step (Step n):** The agent receives the final result (6 suppliers) and returns both the SPARQL query and the natural language answer to the user, terminating.

### 3.2. Agent Implementation

The agent implements the ReAct framework to iteratively interact with its environment. Within each interaction loop, the agent can execute multiple actions in parallel, allowing efficient and flexible graph exploration. The agent consists of three components: the sensor, the actuator, and the LLM.

The sensor is responsible for perceiving and parsing all incoming information, either from the user’s initial question or as feedback from the environment (search results, RDF descriptions, query outputs,

<b>User: How many suppliers can deliver alternative compatible products for the K367 Strain Encoder?</b>
<b>Graf von Data (Trace)</b>
Step 1
<b>Think:</b> The user is asking for the number of suppliers that can deliver alternative compatible products for a specific item, the K367 Strain Encoder. The first step is to
<b>Act:</b> search("K367 Strain Encoder")
<b>Observation:</b> The search for K367 Strain Encoder returned the following matches. <a href="http://ld.company.org/prod-instances/hw-K367-1320550">http://ld.company.org/prod-instances/hw-K367-1320550</a> 5.6152534 <a href="http://ld.company.org/prod-instances/bom-part-9-K367-1320550">http://ld.company.org/prod-instances/bom-part-9-K367-1320550</a> 3.1478124 ...
Step 2
<b>Think:</b> The search results include several URIs that seem related to the K367 Strain Encoder. The first URI, 'http://ld.company.org/prod-instances/hw-K367-1320550', appears to be the most relevant as it directly mentions the K367 Strain Encoder.
<b>describe</b> ("http://ld.company.org/prod-instances/hw-K367-1320550")
<b>Observation:</b> The description of <a href="http://ld.company.org/prod-instances/hw-K367-1320550">http://ld.company.org/prod-instances/hw-K367-1320550</a> is as follows: <pre> @prefix ns1: &lt;http://ld.company.org/prod-vocab/&gt; . @prefix ns2: &lt;http://ld.company.org/prod-instances/&gt; . @prefix rdf: &lt;http://www.w3.org/2000/01/22-rdf-syntax-ns#&gt; . @prefix rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; . @prefix xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt; . ns2:hw-K367-1320550 ns1:compatibleProduct ns2:hw-C119-5354812, ns2:hw-K845-4116844, ns2:hw-Q248-7597886, ns2:hw-R658-8902629, ns2:hw-S271-9518696, ns2:hw-T808-9704137 ; ns1:depth_mm 16 ; ns1:hasCategory ns2:prod-cat-Encoder, ns2:prod-cat-Strain ; ns1:hasProductManager ns2:empl-Dietlinde.Boehme%2540company.org ; ns1:hasSupplier ns2:suppl-7c4b3772-ee27-4915-8166-359d068db153 ; ns1:height_mm 62 ; ns1:id "K367-1320550" ; ns1:name "Strain Encoder" ; ns1:price ns2:price-hw-K367-1320550-EUR ; ns1:reliabilityIndex 0.929 ; ns1:weight_g 8 ; ns1:width_mm 27 ;0 rdf:type ns1:Hardware ; rdfs:label "K367-1320550 - Strain Encoder" . </pre>
Step 3
<b>Think:</b> The description of the K367 Strain Encoder includes a list of compatible products. To find out how many suppliers can deliver these alternative compatible products, I need to describe each of the compatible products to find their suppliers.
<b>Act:</b> describe(" <a href="http://ld.company.org/prod-instances/hw-C119-5354812">http://ld.company.org/prod-instances/hw-C119-5354812</a> ")   describe(" <a href="http://ld.company.org/prod-instances/hw-K845-4116844">http://ld.company.org/prod-instances/hw-K845-4116844</a> ")
<b>Observation:</b> ...
Step ...
<b>Think:</b> ...
<b>Act:</b> ...
<b>Observation:</b> ...
Step n-1
<b>Think:</b> Now that I have the descriptions of all the compatible products for the K367 Strain Encoder, I can see that each product has a 'hasSupplier' property that points to a specific supplier URI.
<b>Act:</b> query("PREFIX ns1: <http://ld.company.org/prod-vocab/> SELECT (COUNT(DISTINCT ?supplier) AS ?numSuppliers) WHERE { <http://ld.company.org/prod-instances/hw-K367-1320550> ns1:compatibleProduct ?compatibleProduct . ?compatibleProduct ns1:hasSupplier ?supplier .}")
<b>Observation:</b> The query "PREFIX ns1: <http://ld.company.org/prod-vocab/> SELECT (COUNT(DISTINCT ?supplier) AS ?supplierCount) WHERE { <http://ld.company.org/prod-instances/hw-K367-1320550> ns1:compatibleProduct ?compatibleProduct . ?compatibleProduct ns1:hasSupplier ?supplier .}" returned the following results: [{"supplierCount": "6" <a href="http://www.w3.org/2001/XMLSchema#integer">http://www.w3.org/2001/XMLSchema#integer</a> }]
Step n
<b>Think:</b> The query has confirmed that there are 6 unique suppliers that can deliver alternative compatible products for the K367 Strain Encoder. This matches the information I gathered from the descriptions of the compatible products. I have now successfully determined the number of suppliers that can deliver alternative compatible products for the K367 Strain Encoder.
<b>Act:</b> success("There are 6 suppliers that can deliver alternative compatible products for the K367 Strain Encoder.")
<b>Graf von Data (Answer)</b>
<b>User Response:</b> For the question "How many suppliers can deliver alternative compatible products for the K367 Strain Encoder?" we used the following SPARQL query: <pre> PREFIX ns1: &lt;http://ld.company.org/prod-vocab/&gt; SELECT (COUNT(DISTINCT ?supplier) AS ?supplierCount) WHERE {   &lt;http://ld.company.org/prod-instances/hw-K367-1320550&gt; ns1:compatibleProduct ?compatibleProduct .   ?compatibleProduct ns1:hasSupplier ?supplier . } </pre> and found that: There are 6 suppliers that can deliver alternative compatible products for the K367 Strain Encoder

**Figure 3:** Example of GvD answering a user question.

or error messages). The sensor transforms this information into structured text and appends it to the prompt, extending the history. The updated prompt serves as the internal state for the LLM's decision-making.

The prompt is the primary mechanism to direct the LLM towards robust and efficient query generation. GvD implements a zero-shot prompt that only includes high-level instructions. The initial

prompt consists of the following five sections:

1. **Role Definition:** This section instructs the model to act as an autonomous agent tasked with answering user questions by exploring the KG and generating appropriate SPARQL queries.
2. **Loop Definition:** The loop definition outlines the strategy based on the ReAct framework, allowing the agent to alternate between processing environmental feedback and taking actions, facilitating controlled and explainable behaviour. The loop is composed of three components: (i) Think to formulate the decision-making of the agent, (ii) Act, in which the agent selects and executes the following action, and (iii) Observation to perceive data from the environment.
3. **Actions:** Each action is described in terms of its input requirements, its intended purpose, and the expected output format. The actions are keyword search, entity description, SPARQL query execution, and termination (success or failure). By explicitly defining these actions, the prompt provides the agent with a clear operational vocabulary for interacting with the KG.
4. **Stage Description:** The stage description suggests an "explore and expand" approach, in which the agent begins with a coarse-grained search and incrementally builds upon the results of previous actions.
5. **General Guidelines:** The guidelines provide recommendations that support correct and efficient system behaviour. For example, the agent is advised to avoid relying on known namespaces (such as DBpedia), always to use fully expanded URIs when composing SPARQL queries, and to be aware of the maximum number of interaction loops. The heuristics help the agent to remain compatible with different KGs and prevent common failure modes.

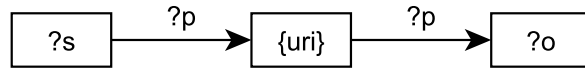
The agent can interact with its environment through the actuator. The actuator parses the textual action call and executes the action on the environment. The actuator has five actions available. Two actions specify termination conditions (success and failure) to enable the agent to stop the query generation. The success condition allows the agent to signal a found query based on the information from the graph. The fail condition indicates that the agent found either no or an unsatisfactory query. The other three actions enable the agent to interact with any KG:

1. **Search action:** Is an entity linking action that matches entity URIs from natural language keywords by the similarity score of the keyword and entity textual descriptions. The action uses keyword search plugins that are widely available for RDF triple stores<sup>2</sup>, to enable search based on text similarity within the KG through a SPARQL SELECT query. The similarity score can be specified on any literal. The search extension creates an index, such as a Lucene Index, using the literals to facilitate faster retrieval.
2. **Describe action:** Is a subgraph retrieval method that returns a subgraph for a provided entity URI according to defined entity boundaries. The boundaries vary based on the entity's type. The specific boundaries for class entities [28] are shown in Figure 5 and expressed by a SPARQL query with the union of the graph patterns in Listings 1, 3, 4, and 5 from Appendix A. The specific boundaries for object property entities [28] are shown in Figure 6 and expressed by a SPARQL query with the union of the graph patterns in Listings 1, 7, 8, and 6 from Appendix A. The entity boundaries for all other entities are shown in Figure 4 and expressed by a SPARQL query with the union of the graph patterns in Listings 1 and 2 from Appendix A. For all outgoing (see Listing 1) and incoming (see Listing 2) triples, we prune property occurrences that exist over ten times (i.e., remove additional triples with that property) to limit the total number of triples in the subset.
3. **Query action:** Executes a SPARQL query specified by the LLM. The results are returned as a JSON result set.

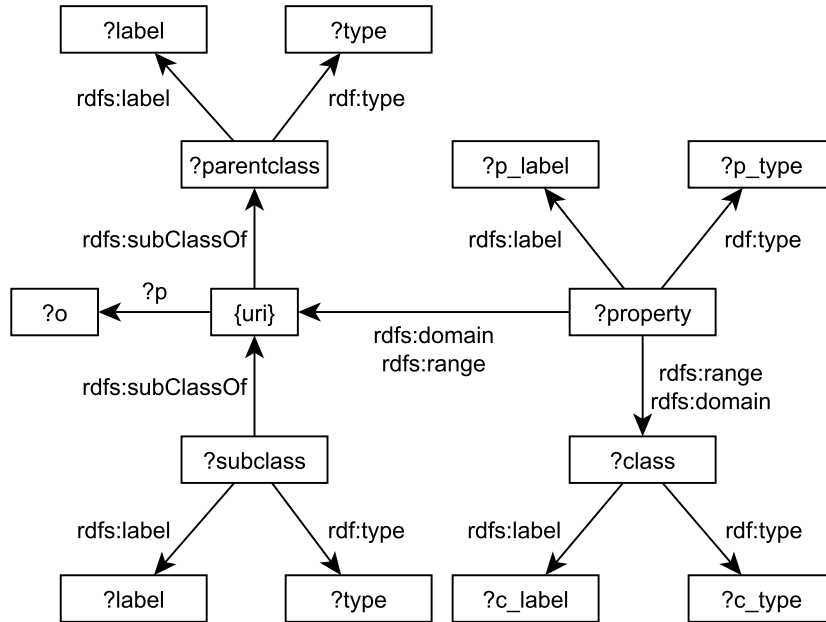
---

<sup>2</sup>See e.g., for Apache Jena: <http://loopasam.github.io/jena-doc/documentation/query/text-query.html>, GraphDB: <https://graphdb.ontotext.com/documentation/11.0/full-text-search.html>, Blazegraph: <https://github.com/blazegraph/database/wiki/FullTextSearch>

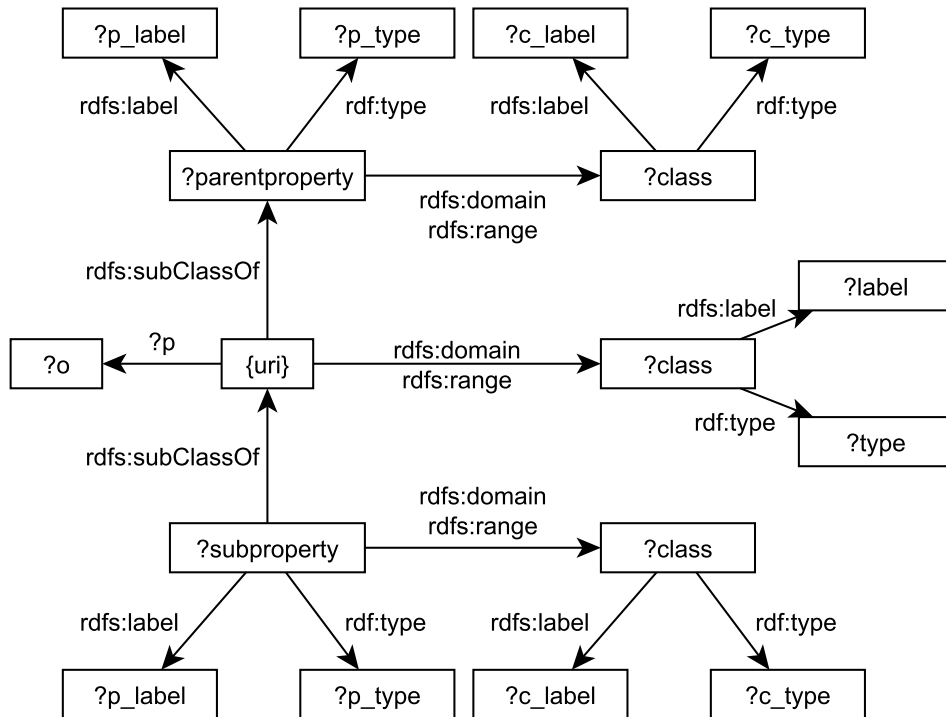




**Figure 4:** Entity boundaries retrieved with the describe action [28]. Equivalent to a SPARQL query with the union of the graph patterns in Listings 1 and 2 from Appendix A.



**Figure 5:** Specific class boundaries retrieved with the describe action [28]. Equivalent to a SPARQL query with the union of the graph patterns in Listings 1, 3, 4, and 5 from Appendix A.



**Figure 6:** Specific object property boundaries retrieved with the describe action [28]. Equivalent to a SPARQL query with the union of the graph patterns in Listings 1, 7, 8, and 6 from Appendix A.

The actuator executes actions (including termination conditions) selected by the LLM. A previous study [29] found that selected actions should be evaluated before execution to prevent failures, unnecessary repetitions, and hallucinations. Thus, the actuator includes a function to validate each action before execution and provides, in the case of a failure, feedback to the sensor. The control function ensures that only syntactically valid and non-repeated actions are executed against the KG.

For the LLM selection, we prioritised different LLMs according to the organisational demands. Thus, we evaluated different open-weight models considering three key metrics: performance, robustness, and model size. We employed SupplyBench<sup>3</sup> for the evaluation of LLMs. SupplyBench is a benchmark for enterprise-focused KGQA systems, featuring a KG with question/query pairs of varying complexity. Our experimental findings revealed distinct performance patterns across model sizes. Small LLMs with fewer than 10 billion parameters showed unstable agent loop behaviour. Large LLMs with more than 100 billion parameters showed higher performance but required prohibitively higher computational resources. Medium-sized LLMs with approximately 70 billion parameters have achieved the optimal balance between performance and efficiency. Specifically, current state-of-the-art medium-sized models, including Llama 3.3 70B Instruct and Qwen 2.5 72B Instruct, successfully managed the robustness requirements for effective agent loops and parallel action planning. LLM-KG-Bench-3 [30] confirmed our findings, as Llama 3.3 70B and Qwen 2.5 72B demonstrated strong performance across various KG-specific tasks.

## 4. Experimental Setup

We participated in the Text2SPARQL challenge for the evaluation of GvD. For the Text2SPARQL challenge, GvD and the associated Text2SPARQL APIs were self-hosted among two Apache Jena Fuseki<sup>4</sup> servers extended by Jena Full Text Search<sup>5</sup>. The two APIs provided the corporate and DBpedia SPARQL endpoints to GvD. Jena Full Text Search used Lucene indexes based on `rdfs:label` values in the respective KGs and the default analyser and tokeniser.

The configuration of GvD limited the number of interaction loops to eight and the response times for the search action to 10 seconds, describe action to 20 seconds, and query action to 30 seconds.

We submitted two versions of GvD to the Text2SPARQL challenge, which differed only in the LLM used. One used Llama 3.3 70B Instruct<sup>6</sup> and the other Qwen 2.5 72B Instruct<sup>7</sup>. We used both models hosted at Chat AI<sup>8</sup> [31]. Regarding model parameters, the temperature (influencing the output creativity) was set to 0 (lowest creativity), and the `top_p` (influencing the output variety) was set to 1 (highest variety).

All experiments were conducted with one run by the Text2SPARQL organizers. The organizers evaluated GvD with the metrics precision, recall,  $F_1$ , and for SPARQL queries with sorted results, NDCG (Normalised Discounted Cumulative Gain). The organizers did not differentiate between different types of queries (e.g. ASK, SELECT). We were not able to reproduce the results from the organizers with our experimental setup<sup>9</sup>. This may be caused by changed or unknown configuration parameters of the organizers or Chat AI.

## 5. Competitor Profiles

Nine groups participated in the Text2SPARQL challenge with twelve competing implementations. Three competitors withdrew from the competition after the results were revealed. Therefore, we excluded them from the evaluation. Table 1 lists the remaining participants with their KGQA strategy.

<sup>3</sup>See <https://github.com/wintechis/supplybench>

<sup>4</sup>See <https://jena.apache.org/documentation/fuseki2/index.html>

<sup>5</sup>See <https://jena.apache.org/documentation/query/text-query.html>

<sup>6</sup>See <https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct>

<sup>7</sup>See <https://huggingface.co/Qwen/Qwen2.5-72B-Instruct>

<sup>8</sup>See <https://docs.hpc.gwdg.de/services/chat-ai/index.html>

<sup>9</sup><https://github.com/AKSW/text2sparql.aksw.org/issues/101>



**Table 1**

Text2SPARQL competitors that presented their approaches. Our approach is highlighted in bold.

Organisation	KGQA Approach
Institute of Applied Informatics and Formal Description Methods (AIFB) [18]	Workflow
Artificial Intelligence Research Institute (AIRI) [21]	Workflow
DBpedia Group (DBpedia) [19]	Direct
<b>Fraunhofer IIS (IIS)</b>	<b>ReAct</b>
ETi @ Institute for Applied Informatics (InfAI) [20]	ReAct
WSE Research Group @ Leipzig University of Applied Sciences (WSE) [22]	Plan-and-Solve

The Institute of Applied Informations and Formal Description Methods (AIFB) of the Karlsruher Institute of Technology developed a four-step workflow approach [18] involving language translation, entity extraction, ShEx shape generation via shexer [32], and finally SPARQL generation.

The Artificial Intelligence Research Institute (AIRI) proposed Text-to-RDF Executor (TReX) [21], a workflow-based approach that begins with language translation but then diverges into KG-specific processing pipelines. The DBpedia KG pipeline includes additional steps for query rewriting, entity linking through additional dependencies, and query refinement through question-query examples provided through the LCQuad<sup>10</sup> and QALD-9<sup>11</sup> datasets to improve question generation performance.

The DBpedia Group’s approaches [19] fine-tuned three open-weight models, CodeGen 2.5 350M [33] (DBpedia-CG), StarCoder 1B [34] (DBpedia-SC), and CodeLlama 7b [35] (DBpedia-CL) using the NSpM data set<sup>12</sup> containing 8 million natural language question-SPARQL pairs.

The Institute for Applied Informatics (INFAI) developed a ReAct agent approach [20]. While sharing GvD’s loop definition, the system relies on a vector database as an additional infrastructure. The approach has access to six actions. It has three actions - entity search, property search and class search - for entity and relation linking, two actions - property example and knowledge graph entry - for subgraph retrieval and a query action.

The Web & Software Engineering (WSE) Research Group of the Leipzig University of Applied Sciences developed the mKGQAgent [22]. The mKGQAgent is a plan-and-solve approach comprising both offline and online phases. The offline phase creates an experience pool from training data to generate successful sequences of actions, while the online phase uses this pool to guide question analysis and step-by-step action execution with GPT-4o. The experience pool is stored within a vector database, using the multilingual e5 text embeddings [36] to store successful plans.

## 6. Results and Discussion

This section evaluates GvD’s performance in three parts: a quantitative evaluation based on the Text2SPARQL challenge (Sec. 6.1), a qualitative comparison with Text2SPARQL challenge competitors regarding usability criteria in an organisational context, and a qualitative investigation of system errors of GvD (Sec. 6.3).

### 6.1. Performance Evaluation

We evaluate GvD’s effectiveness through quantitative analysis of the Text2SPARQL challenge performance results, examining results across the Corporate KG and DBpedia benchmarks to assess system performance in SPARQL query generation from natural language questions.

Table 2 shows that the ReAct agent approaches achieve the best performance on the Corporate KG benchmark. GvD with the Qwen 2.5 72b Instruct model (IIS-Q) achieved the highest recall (0.452) and F<sub>1</sub>-score (0.444). The approach from INFAI achieved the highest performance in terms of NDCG score (1.0) and precision (0.542).

<sup>10</sup><https://github.com/AskNowQA/LC-QuAD>

<sup>11</sup><https://github.com/ag-sc/QALD/tree/master>

<sup>12</sup><https://github.com/LiberAI/NSpM/tree/master>

**Table 2**

Text2SPARQL Corporate KG benchmark results. Our approach is highlighted in bold. **IIS-L** stands for the Llama, **IIS-Q** for the Qwen variant. The best values are highlighted in bold, the second best underlined. DBpedia did not participate in the Corporate KG benchmark.

Competitors	Precision	Recall	$F_1$	NDCG	Combined ( $F_1$ and NDCG)
AIFB	0.000	0.000	0.000	0.000	0.000
<b>IIS-L (=GvD)</b>	0.365	0.428	0.374	0.261	0.371
<b>IIS-Q (=GvD)</b>	<u>0.522</u>	<b>0.452</b>	<b>0.444</b>	0.422	<u>0.443</u>
INFAI	<b>0.542</b>	<u>0.436</u>	<u>0.413</u>	<b>1.000</b>	<b>0.447</b>
MIPT	0.218	<u>0.228</u>	<u>0.219</u>	0.315	0.219
WSE	0.315	0.343	0.321	<u>0.429</u>	0.322

GvD with the Llama 3.3 70b Instruct model (IIS-L) followed behind the other two competitors. While individual metric leadership varies across approaches, the consistent dominance of ReAct agent approach over the other approaches suggests that flexible exploration capabilities provide advantages for KGQA applications without prior knowledge.

**Table 3**

Text2SPARQL DBpedia benchmark results. Our approach is highlighted in bold. **IIS-L** stands for the Llama, **IIS-Q** for the Qwen variant. The best values are highlighted in bold, the second best underlined.

Competitors	Precision	Recall	$F_1$	NDCG	Combined ( $F_1$ and NDCG)
AIFB	0.448	<u>0.487</u>	<u>0.451</u>	<b>0.820</b>	<u>0.456</u>
DBpedia-CG	0.055	<u>0.055</u>	0.055	0.000	0.055
DBpedia-CL	<u>0.492</u>	0.053	0.051	0.000	0.050
DBpedia-SC	0.097	0.100	0.098	0.000	0.098
<b>IIS-L (=GvD)</b>	0.349	0.389	0.354	0.444	0.354
<b>IIS-Q (=GvD)</b>	0.375	0.348	0.339	0.424	0.340
INFAI	0.371	0.462	0.376	0.256	0.376
MIPT	0.412	0.418	0.407	0.194	0.406
WSE	<b>0.544</b>	<b>0.555</b>	<b>0.530</b>	<u>0.570</u>	<b>0.531</b>

Table 3 reveals decreased performance across all metrics on the DBpedia benchmark compared to the Corporate KG benchmark. Both ReAct agent-based approaches (GvD and INFAI) showed worse results.

Considering the benchmark results by language (see Table 4), the decrease could potentially be attributed to the lack of language translation, which the other approaches besides DBpedia’s use. Table 4a presents the results for Spanish questions in the DBpedia benchmark, while Table 4b displays the results for English questions. ReAct agent approaches without language translation preprocessing (both GvD configurations and INFAI) experienced larger performance degradation when processing Spanish questions (30–60% decline). In contrast, the other approaches, using natural language translation or multilingual embeddings, showed only minimal changes in terms of point and percentage differences. Furthermore, the evaluation also highlighted performance differences between GvD’s Qwen 2.5 and Llama 3.3 configurations, illustrating model-specific multilingual capabilities, with Qwen’s more diverse multilingual training [37] contributing to reduced performance degradation. These findings suggest that language translation preprocessing or multilingual model selection are relevant for organisations operating in multilingual environments.

## 6.2. Comparison to Organisational Demands

To evaluate GvD’s suitability for organisational usage, we compare the Text2SPARQL challenge competitors across three demands derived from organisational needs: transparent answer generation, organisational control, and flexible deployment. Table 5 summarises the evaluation of GvD and the competitor systems.

**Table 4**

Text2SPARQL DBpedia benchmark results by language. Our approach is highlighted in bold. **IIS-L** stands for the Llama, **IIS-Q** for the Qwen variant. The best values are highlighted in bold, the second best underlined.

(a) Spanish questions				(b) English questions			
Competitors	Precision	Recall	F <sub>1</sub>	Competitors	Precision	Recall	F <sub>1</sub>
AIFB	<u>0.442</u>	<u>0.482</u>	<u>0.446</u>	AIFB	<u>0.453</u>	0.491	0.457
DBPedia-CG	0.050	0.050	0.050	DBPedia-CG	0.060	0.060	0.060
DBPedia-CL	0.034	0.038	0.036	DBPedia-CL	0.064	0.068	0.064
DBPedia-SC	0.087	0.090	0.088	DBPedia-SC	0.107	0.110	0.108
<b>IIS-L (=GvD)</b>	0.285	0.315	0.290	<b>IIS-L (=GvD)</b>	0.412	0.463	0.417
<b>IIS-Q (=GvD)</b>	0.311	0.339	0.302	<b>IIS-Q (=GvD)</b>	0.384	0.411	0.377
INFAI	0.214	0.262	0.207	INFAI	<b>0.527</b>	<b>0.661</b>	<b>0.546</b>
MIPT	0.373	0.376	0.367	MIPT	0.452	0.461	0.447
WSE	<b>0.562</b>	<b>0.556</b>	<b>0.537</b>	WSE	<b>0.527</b>	<u>0.554</u>	<u>0.522</u>

**Table 5**

Comparison to Text2SPARQL competitors based on organisations’ demands for trustworthiness and approach reusability. Our approach is highlighted in bold.

Organisation	Transparent Answer Generation	Organisational Control	Flexible Deployment
AIFB	medium	low	high
AIRI	low	low	low
DBpedia Group	low	high	low
<b>IIS (=GvD)</b>	high	high	high
INFAI	medium	low	low
WSE	high	low	medium

The AIFB team employed a modular four-step workflow that supports generalisation across KGs. However, their dependence on the closed-weight GPT-4o model from OpenAI limits organisational control and may introduce privacy concerns. Additionally, their use of ShEx shapes to provide schema information about the KG during query construction may obscure how answers are generated, reducing transparency and making error analysis difficult for end users.

The AIRI group created specialised pipelines for each KG. This approach limits the flexibility to apply the system easily in new domains and increases configuration overhead. Additionally, their reliance on the closed-weight GPT-4 model further restricts organisational control and prevents on-premise usage.

The DBpedia Group uses fine-tuned, open-weight models, which support organisational control by enabling local hosting and avoiding third-party dependencies. However, the model operates in a static inference mode solely based on training data rather than considering up-to-date information, which limits the transparency of the answer generation. The reliance on small language models lowers deployment demands, but enforces continuous retraining if the KG changes.

The mKGQAgent from the WSE group relies on a precomputed experience pool to support plan generation. This offline component helps with generating more transparent answers, as plans are composed of verifiable steps. However, the need for training data that resembles the target domain reduces the system’s ability to generalise, thereby increasing deployment overhead. The usage of GPT-4o again limits organisational control due to the reliance on a closed-weight LLM.

The INFAI system adopts a ReAct agent approach similar to GvD, but relies on a vector database for resolving entity identifiers. This dependency increases the deployment effort, particularly for organisations aiming to reduce infrastructure complexity. The usage of OpenAI’s GPT-4.1 mini model reduces organisational control, introducing external dependencies and may raise data privacy concerns. Trustworthiness was not explicitly reported, but the reasoning traces may be available, showing how the agent came up with a result.

In contrast, GvD is explicitly designed to address all three organisational demands. It ensures trans-

parent answers by generating structured reasoning traces that show how a query was derived. It supports full organisational control by using open-weight LLMs that can be deployed locally, avoiding external dependencies and supporting data privacy. Furthermore, GvD has a high deployment flexibility, as it is not optimised for a specific KG or requires infrastructure beyond a SPARQL endpoint with a keyword search plugin.

### 6.3. Error Analysis

We conducted an error analysis to identify current system limitations and opportunities for improvement. We created a dataset of 40 problematic cases by examining the execution logs from questions generating at least three errors. The error distribution revealed language and benchmark dependencies. Only 3 (7.5%) cases were caused by the Corporate KG benchmark compared to 37 (92.5%) from the DBpedia benchmark. Furthermore, 23 of 37 (62%) of the errors from the DBpedia benchmark were caused by Spanish questions.

In the analysis of the errors, we made three observations:

- **Data Parsing Failures:** Unsupported formatting in SPARQL queries (through the query action) and RDF responses (through the describe action) are in 95% (38/40) cases. These failures stem from missing namespace definitions for standard vocabularies (e.g., missing `rdfs: prefix`) and inadequate URI escaping (e.g., `dbr:New_York_(City)` vs. `dbr:New_York_(City\)`).
- **Function Repetition Errors:** The LLM repeatedly tried to execute identical functions with identical arguments in 77.5% of cases (31/40), suggesting currently insufficient feedback mechanisms to prevent unproductive exploration loops.
- **Stable Agentic Loop:** We observed no case with a loss of the agentic loop control or failure to execute a proper action call, indicating that ReAct agent approaches may be adequate for KGQA.

## 7. Conclusion

This paper tackles the accessibility and query challenges of KGs within organisations for non-technical users. We introduced Graf von Data (GvD), an LLM-augmented agent that bridges this accessibility gap by translating natural language questions into executable SPARQL queries while fulfilling organisational usage demands.

GvD’s innovation lies in its knowledge graph-agnostic approach, which operates exclusively through a SPARQL endpoints with keyword search plugin, enabling seamless integration across diverse KG environments without further customisation. This approach, combined with our use of open-weight models, distinguishes GvD among the Text2SPARQL challenge competitors. Despite this focus on organisational demands, GvD achieved the highest  $F_1$ -score on the Corporate KG benchmark and competitive results in the DBpedia benchmark.

An evaluation of error cases presents opportunities for future enhancement. Future research directions may include expanding the actions of GvD through additional SPARQL interface mappings, integrating specialised entity recognition capabilities, and exploring hybrid approaches combining keyword search with embedding-based indexing for improved entity linking.

## Acknowledgments

This work was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) through the Antrieb 4.0 project (Grant No. 13IK015B).

The authors gratefully acknowledge the scientific support and HPC resources provided by the Erlangen National High Performance Computing Center (NHR@FAU). The hardware is funded by the German Research Foundation (DFG).

## Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly to: Grammar and spelling check. After using these tool(s)/service(s), the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

## References

- [1] E. G. Kalaycı, I. Grangel González, F. Lösch, G. Xiao, A. ul-Mehdi, E. Kharlamov, D. Calvanese, Semantic Integration of Bosch Manufacturing Data Using Virtual Knowledge Graphs, in: *The Semantic Web – ISWC 2020*, volume 12507 of *LNCS*, Springer International Publishing, 2020, pp. 464–481. doi:10.1007/978-3-030-62466-8\_29.
- [2] T. Hubauer, S. Lamparter, P. Haase, D. Herzig, Use Cases of the Industrial Knowledge Graph at Siemens, in: *ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks*, Monterey, USA, 2018.
- [3] T. Liebig, A. Maisenbacher, M. Opitz, J. R. Seyler, G. Sudra, J. Wissmann, Building a Knowledge Graph for Products and Solutions in the Automation Industry, in: *Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics co-located with 16th Extended Semantic Web Conference (ESWC 2019)*, 2019. URL: <https://ceur-ws.org/Vol-2489/paper2.pdf>.
- [4] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, J. Taylor, Industry-scale Knowledge Graphs: Lessons and Challenges, *ACM Queue* 17 (2019) 48–75. doi:10.1145/3329781.3332266.
- [5] D. Diefenbach, M. D. Wilde, S. Alipio, Wikibase as an Infrastructure for Knowledge Graphs: The EU Knowledge Graph, in: *The Semantic Web – ISWC 2021*, volume 12922 of *LNCS*, Springer International Publishing, 2021, pp. 631–647. doi:10.1007/978-3-030-88361-4\_37.
- [6] E. Kaufmann, A. Bernstein, How Useful Are Natural Language Interfaces to the Semantic Web for Casual End-Users?, in: *The Semantic Web*, volume 4825, Springer, 2007, pp. 281–294. doi:10.1007/978-3-540-76298-0\_21.
- [7] T. Chen, M. Gascó-Hernandez, M. Esteve, The Adoption and Implementation of Artificial Intelligence Chatbots in Public Organizations: Evidence from U.S. State Governments, *The American Review of Public Administration* 54 (2024) 255–270. doi:10.1177/02750740231200522.
- [8] T. Bruckhaus, RAG Does Not Work for Enterprises, 2024. doi:10.48550/arXiv.2406.04369. arXiv:2406.04369.
- [9] D. M. Schmidt, M. F. Elahi, P. Cimiano, Lexicalization Is All You Need: Examining the Impact of Lexical Knowledge in a Compositional QALD System, in: *Knowledge Engineering and Knowledge Management*, Springer Nature, 2025, pp. 102–122. doi:10.1007/978-3-031-77792-9\_7.
- [10] S. Liu, S. Semnani, H. Friedman, J. Xu, I. D. Zhao, M. Lam, SPINACH: SPARQL-Based Information Navigation for Challenging Real-World Questions, in: *Findings of the Association for Computational Linguistics: EMNLP 2024*, Association for Computational Linguistics, 2024, pp. 15977–16001. doi:10.18653/v1/2024.findings-emnlp.938.
- [11] C. V. S. Avila, M. A. Casanova, V. M. P. Vidal, A Framework for Question Answering on Knowledge Graphs Using Large Language Models, in: *The Semantic Web: ESWC 2024 Satellite Events*, volume 15344 of *LNCS*, Springer, 2025, pp. 168–172. doi:10.1007/978-3-031-78952-6\_20.
- [12] G. Xiong, J. Bao, W. Zhao, Interactive-KBQA: Multi-Turn Interactions for Knowledge Base Question Answering with Large Language Models, 2024. doi:10.48550/arXiv.2402.15131. arXiv:2402.15131.
- [13] L. Sun, Z. Tao, Y. Li, H. Arakawa, ODA: Observation-Driven Agent for integrating LLMs and Knowledge Graphs, 2024. doi:10.48550/arXiv.2404.07677. arXiv:2404.07677.
- [14] S. Xu, S. Liu, T. Culhane, E. Pertseva, M.-H. Wu, S. Semnani, M. Lam, Fine-tuned LLMs Know More, Hallucinate Less with Few-Shot Sequence-to-Sequence Semantic Parsing over Wikidata, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Process-*

- ing, Association for Computational Linguistics, 2023, pp. 5778–5791. doi:10.18653/v1/2023.emnlp-main.353.
- [15] G. Piao, M. Mountantonakis, P. Papadakis, P. Sonawane, A. OMahony, Toward Exploring Knowledge Graphs with LLMs, in: Joint Proceedings of Posters, Demos, Workshops, and Tutorials of the 20th International Conference on Semantic Systems (SEMANTiCS 2024), volume Vol-3759, CEUR-WS, Amsterdam, The Netherlands, 2024.
  - [16] H. M. Zahera, M. Ali, M. A. Sherif, D. Moussallem, A.-C. Ngonga Ngomo, Generating SPARQL from Natural Language Using Chain-of-Thoughts Prompting, in: Knowledge Graphs in the Age of Language Models and Neuro-Symbolic AI, volume 60 of *Studies on the Semantic Web*, IOS Press, Amsterdam, 2024, pp. 353–368. doi:10.3233/SSW240028.
  - [17] L. Kovriguina, R. Teucher, D. Radyush, D. Mourmstev, SPARQLGEN: One-Shot Prompt-based Approach for SPARQL Query Generation, in: Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems (SEMANTiCS 2023), volume Vol-3526, CEUR, Leipzig, Germany, 2023.
  - [18] J. Wardenga, T. Käfer, Challenge submission: AIFB, <https://text2sparql.aksw.org/assets/talks/2-AIFB-tobias-kaefer-kit.pdf>, 2025. Accessed 2025-06-13.
  - [19] M. Shahinmoghdam, T. Soru, S. Josh, S. Tiwari, DBpedia Group at the 1st Text2SPARQL Challenge, <https://text2sparql.aksw.org/assets/talks/3-DBPEDIA-tommaso-soru.pdf>, 2025. Accessed 2025-06-13.
  - [20] D. Gerber, L. Bühmann, L.-P. Meyer, C. Stadler, Felix Brei, J. Frey, K. Bulert, ETi / InfAI: Text2Sparql Agent, <https://text2sparql.aksw.org/assets/talks/5-INFAl-lars-peter-meyer.pdf>, 2025. Accessed 2025-06-13.
  - [21] O. Somov, D. Berezin, R. Avdeev, TEXT2SPARQL: TReX (Text-to-RDF Executor), <https://text2sparql.aksw.org/assets/talks/6-MIPT-oleg-somov-daniil-berezin-roman-avdeev-airi.pdf>, 2025. Accessed 2025-06-13.
  - [22] A. Perevalov, A. Both, mKGQAgent: Multilingual Human-inspired Question Answering over Language Graphs, <https://text2sparql.aksw.org/assets/talks/7-WSE-andreas-both-htwk.mp4>, 2025. Accessed 2025-06-13.
  - [23] L. Fei-Fei, R. Fergus, P. Perona, One-shot learning of object categories, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28 (2006) 594–611. doi:10.1109/TPAMI.2006.79.
  - [24] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, in: NIPS’20: Proceedings of the 34th International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, United States, 2020, pp. 9459–9474.
  - [25] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, E.-P. Lim, Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, 2023.
  - [26] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing Reasoning and Acting in Language Models, in: The Eleventh International Conference on Learning Representations ICLR 2023, Kigali, Rwanda, 2023. doi:10.48550/arXiv.2210.03629.
  - [27] OMG Unified Modeling Language (OMG UML), Version 2.5.1, 2017.
  - [28] D. Henselmann, R. Dorsch, A. Harth, Impact of Knowledge Graph Representations on Question Answering with Language Models, in: Advanced Information Systems Engineering Workshops, volume 556 of *LNBIP*, Springer, Vienna, Austria, 2025, pp. 81–92. doi:10.1007/978-3-031-94931-9\_7.
  - [29] S. Schmid, M. Freund, A. Harth, Adaptive Planning on the Web: Using LLMs and Affordances for Web Agents, in: Knowledge Graphs and Semantic Web, volume 15459 of *LNCS*, Springer Nature Switzerland, 2025, pp. 93–108. doi:10.1007/978-3-031-81221-7\_7.
  - [30] 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, volume 15719 of *LNCS*, 2025, pp. 280–296. doi:10.1007/978-3-031-94578-6\_16.
  - [31] A. Doosthosseini, J. Decker, H. Nolte, J. M. Kunkel, Chat AI: A Seamless Slurm-Native Solution



- for HPC-Based Services, 2024. doi:10.48550/arXiv.2407.00110.
- [32] D. Fernandez-Álvarez, J. E. Labra-Gayo, D. Gayo-Avello, Automatic extraction of shapes using sheXer, Knowledge-Based Systems 238 (2022). doi:10.1016/j.knsys.2021.107975.
  - [33] E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, Y. Zhou, CodeGen2: Lessons for Training LLMs on Programming and Natural Languages, 2023. doi:10.48550/arXiv.2305.02309.
  - [34] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, H. de Vries, StarCoder: May the source be with you!, 2023. doi:10.48550/arXiv.2305.06161.
  - [35] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, G. Synnaeve, Code Llama: Open Foundation Models for Code, 2024. doi:10.48550/arXiv.2308.12950.
  - [36] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, F. Wei, Multilingual E5 Text Embeddings: A Technical Report, 2024. doi:10.48550/arXiv.2402.05672.
  - [37] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, Z. Qiu, Qwen2.5 Technical Report, 2025. doi:10.48550/arXiv.2412.15115.

## A. Graph Patterns for Entity Boundaries

Listing 1: Graph pattern: Outgoing triples of the focal entity.

```
<{uri}> ?p ?o .
```

Listing 2: Graph pattern: Incoming triples of the focal entity.

```
?s ?p <{uri}> .
```

Listing 3: Graph pattern: Subclasses of the focal class.

```
?subclass rdfs:subClassOf <{uri}> .
OPTIONAL { ?subclass rdf:type ?type . }
OPTIONAL { ?subclass rdfs:label ?label . }
```

Listing 4: Graph pattern: Parent classes of the focal class.

```
<{uri}> rdfs:subClassOf ?parentclass .
OPTIONAL { ?parentclass rdf:type ?type . }
OPTIONAL { ?parentclass rdfs:label ?label . }
```

Listing 5: Graph pattern: Properties having the focal class as domain or range plus classes in domain or range of such properties.

```
?property ?domainrange <{uri}> .
FILTER ( ?domainrange IN ( rdfs:domain , rdfs:range ))
OPTIONAL { ?property rdf:type ?p_type . }
```

```

OPTIONAL { ?property rdfs:label ?p_label . }
OPTIONAL {
    ?property ?rangedomain ?class .
    FILTER ( ?rangedomain IN ( rdfs:domain , rdfs:range ))
    OPTIONAL { ?class rdfs:label ?c_label . }
    OPTIONAL { ?class rdf:type ?c_type . }
}

```

Listing 6: Graph pattern: Classes in domain or range of the focal object property.

```

<{uri}> ?domainrange ?class .
FILTER ( ?domainrange IN ( rdfs:domain , rdfs:range ))
OPTIONAL { ?class rdfs:label ?label . }
OPTIONAL { ?class rdf:type ?label . }

```

Listing 7: Graph pattern: Subproperties of the focal object property plus classes in domain or range of subproperties.

```

?subproperty rdfs:subPropertyOf <{uri}> .
OPTIONAL { ?subproperty rdf:type ?p_type . }
OPTIONAL { ?subproperty rdfs:label ?p_label . }
OPTIONAL {
    ?subproperty ?rangedomain ?class .
    FILTER ( ?rangedomain IN ( rdfs:domain , rdfs:range ))
    OPTIONAL { ?class rdf:type ?c_type . }
    OPTIONAL { ?class rdfs:label ?c_label . }
}

```

Listing 8: Graph pattern: Parent properties of the focal object property plus classes in domain or range of parent properties.

```

<{uri}> rdfs:subPropertyOf ?parentproperty .
OPTIONAL { ?parentproperty rdf:type ?p_type . }
OPTIONAL { ?parentproperty rdfs:label ?p_label . }
OPTIONAL {
    ?parentproperty ?rangedomain ?class .
    FILTER ( ?rangedomain IN ( rdfs:domain , rdfs:range ))
    OPTIONAL { ?class rdf:type ?c_type . }
    OPTIONAL { ?class rdfs:label ?c_label . }
}

```