Text-to-SPARQL Goes Beyond English: Multilingual Question Answering Over Knowledge Graphs through Human-Inspired Reasoning

Aleksandr Perevalov[1,2,*], Andreas Both[1]

[1]WSE Research Group, Leipzig University of Applied Sciences, Karl-Liebknecht-Straße 132, 04277, Leipzig, Germany
[2]DICE Research Group, University of Paderborn, Warburger Str. 100, 33098, Paderborn, Germany

Abstract
Accessing knowledge via multilingual natural-language interfaces is one of the emerging challenges in the field of information retrieval and related ones. Structured knowledge stored in knowledge graphs can be queried via a specific query language (e.g., SPARQL). Therefore, one needs to transform natural-language input into a query to fulfill an information need. Prior approaches mostly focused on combining components (e.g., rule-based or neural-based) that solve downstream tasks and come up with an answer at the end. We introduce mKGQAgent, a human-inspired framework that breaks down the task of converting natural language questions into SPARQL queries into modular, interpretable subtasks. By leveraging a coordinated LLM agent workflow for planning, entity linking, and query refinement—guided by an experience pool for in-context learning—mKGQAgent efficiently handles multilingual KGQA. Evaluated on the DBpedia- and Corporate-based KGQA benchmarks within the Text2SPARQL challenge 2025, our approach took first place among the other participants. This work opens new avenues for developing human-like reasoning systems in multilingual semantic parsing.

Keywords
LLM Agents, Text2SPARQL, Knowledge Graph Question Answering, Semantic Parsing

1. Introduction

Previous approaches to multilingual knowledge graph question answering (KGQA), like Diefenbach et al. [1], Turganbay et al. [2], have employed both rule-based and neural methods to address downstream tasks (e.g., named entity recognition, relation detection, query template classification) necessary for constructing structured queries (e.g., SPARQL queries). More recent methods (e.g., Srivastava et al. [3]) leverage Large Language Models (LLMs) to generate such structured queries directly from non-English input. The application of newly introduced *LLM agents* (or *augmented* language models) to KGQA has demonstrated significantly improved performance compared to LLMs that rely solely on standard prompting techniques e.g., Jiang et al. [4], Huang et al. [5]). However, the multilingual aspect of these systems remains largely unexplored within the research community. To the best of our knowledge, *there are no studies investigating the LLM agent architectures for KGQA in multilingual settings.*

One of the key advantages of LLMs is that they enable developers and researchers to model human-like reasoning processes via agentic workflows (cf. Li et al. [6]). When solving complex problems, humans typically break them down into a series of simpler subtasks (cf. Diefenbach et al. [7], Correa et al. [8]), effectively creating a step-by-step *plan* to arrive at a solution. While generating a SPARQL query, this decomposition is essential: not only does one need to break down the task, but also *look up* query language syntax, identify relevant entity identifiers in the target knowledge graph (KG), and analyze *feedback* (e.g., from executing the SPARQL query candidate on the triplestore). To replicate this human-like process, we introduce mKGQAgent–an LLM-based agent framework designed as a KGQA system that follows a semantic-parsing approach. Specifically, given a user query (multiple languages

First International TEXT2SPARQL Challenge, Co-Located with Text2KG at ESWC25, June 01, 2025, Portorož, Slovenia.
[*]Corresponding author.
aleksandr.perevalov@htwk-leipzig.de (A. Perevalov); andreas.both@htwk-leipzig.de (A. Both)
https://perevalov.com (A. Perevalov); http://andreasboth.de (A. Both)
0000-0002-6803-3357 (A. Perevalov); 0000-0002-9177-5463 (A. Both)

are supported), it generates a SPARQL query to fulfill the information need. Accordingly, this paper aims to answer the following *research questions*:

$\mathcal{RQ}$*1* How do different LLM agent steps (e.g., plan, action, tool calling, feedback, etc.) impact the generation of SPARQL queries from natural language?

$\mathcal{RQ}$*2* How efficient are these LLM agent steps in terms of computation time and the number of additional calls required?

$\mathcal{RQ}$*3* How does the quality of SPARQL query generation vary when prompting LLM agents in non-English languages (especially low-resource ones)?

$\mathcal{RQ}$*4* How does translating non-English questions into English affect the quality of KGQA?

We conducted preliminary experiments on the widely used KGQA benchmark QALD-9-plus (introduced in Perevalov et al. [9]) with multilingual support. We evaluate 10 languages, including two classified as endangered. The experimental results on both proprietary and open-source LLMs demonstrate the effectiveness of mKGQAgent's architecture, achieving superior performance even in non-English settings. During the final evaluation on the DBpedia- and Corporate-based KGQA benchmarks within the Text2SPARQL challenge 2025, our approach took first place among the other participants. The source code and the evaluation results are available in our GitHub repository[1].

The paper is organized as follows. In the next section, an overview of the related work is presented. The mKGQAgent architecture is described in Section 3. Section 4 is dedicated to the experimental setup. The results are shown in Section 5 and discussed in Section 6. Section 7 concludes our paper.

2. Related Work

Recent KGQA research has included classical, rule-based, and neural approaches [10, 11]. Diefenbach et al. [1] (QAnswer) and Punjani et al. [12] used query templates and rule indexes without language models. Pellissier Tanon et al. [13] applied grammar rules for SPARQL query transformation. DeepPavlov 2023 employs a fine-tuned language model pipeline for query generation, cf. Turganbay et al. [2]. Omar et al. [14] proposed KGQAN, which integrates answer type prediction and triple pattern generation.

Multilingual KGQA solutions including Zhou et al. [15], which fine-tune multilingual transformers and leverage bilingual lexicon induction. Zhang et al. [16] address cross-lingual semantic parsing over multiple meaning representations in XSemPLR, including SPARQL. Tan et al. [17] improve cross-lingual reasoning, enhancing the Entity Alignment model performance in *English, Chinese, and French* in the CLRN approach.

Zong et al. [18] employ the multi-role LLM agent architecture Triad for SPARQL query construction. MST5 (Srivastava et al. [3]) fine-tunes mT5-XL for generating structured queries. Lehmann et al. [19] enhances LLMs with external tools to mimic human-like reasoning. Jiang et al. [4] integrates a KG-based executor (KG-Agent) and fine-tunes Llama2-7B for improved tool usage. QueryAgent (Huang et al. [5]) mitigates hallucinations with ERASER-based self-correction, excelling on GrailQA and GraphQ. Interactive-KBQA (Xiong et al. [20]) iteratively refines LLM outputs via direct KB interactions.

3. The mKGQAgent Architecture

The mKGQAgent workflow consists of several key steps (see Figure 1 for an overview). Our approach follows the terminology established in recent survey articles on LLM agents, cf. Mialon et al. [21], Wang et al. [22]. The framework operates in two main phases: the *offline phase* and the *evaluation (online) phase*. The offline phase is essential for preparing the *experience pool* (see Section 3.1.4). During the offline phase, we employ the *simple agent* ($\mathcal{SA}gent$) to gather intermediate processing steps for the experience pool (see Figure 2).

[1]https://github.com/WSE-research/text2sparql-agent

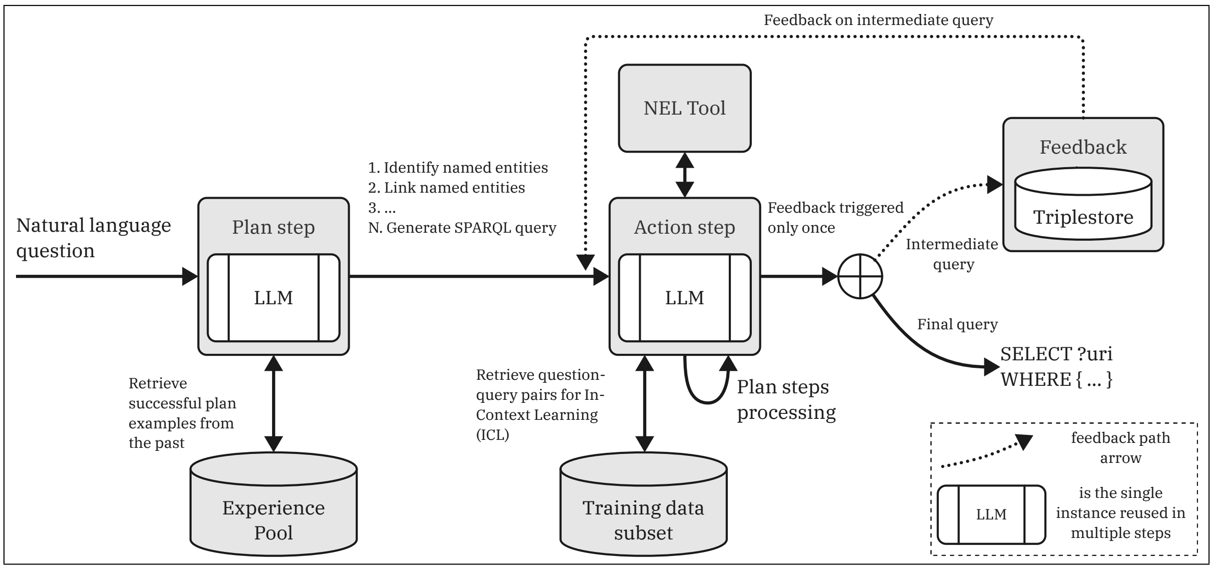
Feedback on intermediate query
NEL Tool
Feedback
Triplestore
1. Identify named entities
2. Link named entities
3. ...
N. Generate SPARQL query
Natural language question
Plan step
LLM
Action step
LLM
Feedback triggered only once
Intermediate query
Final query
SELECT ?uri WHERE { ... }
Retrieve successful plan examples from the past
Retrieve question-query pairs for In-Context Learning (ICL)
Plan steps processing
Experience Pool
Training data subset
feedback path arrow
is the single instance reused in multiple steps

Figure 1: mKGQAgent workflow demonstration (online phase). In the evaluation phase, the mKGQAgent is using the experience pool examples to improve planning, the in-context learning training examples to improve SPARQL query generation awareness and the feedback to correct possible errors. The offline phase, which is required for gathering experience pool. The evaluation or online phase – the routing of the mKGQAgent's components as well as their integral modules.

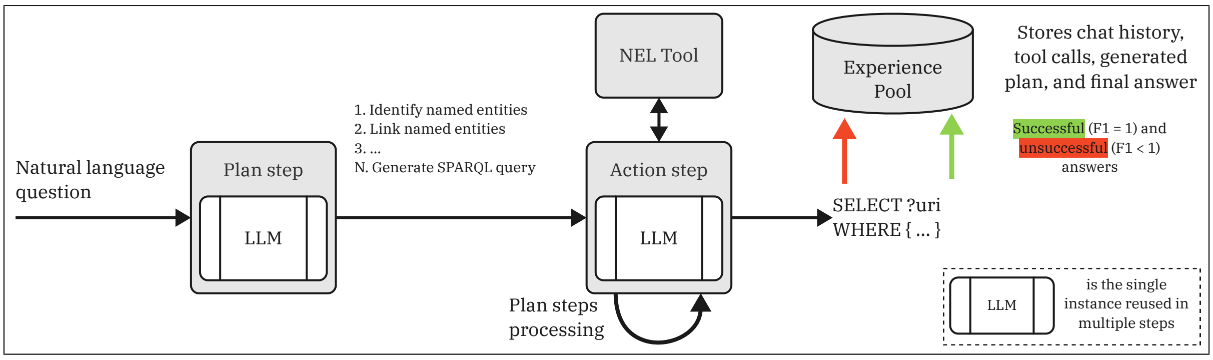
NEL Tool
Experience Pool
Stores chat history, tool calls, generated plan, and final answer
1. Identify named entities
2. Link named entities
3. ...
N. Generate SPARQL query
Successful (F1 = 1) and unsuccessful (F1 < 1) answers
Natural language question
Plan step
LLM
Action step
LLM
SELECT ?uri WHERE { ... }
Plan steps processing
is the single instance reused in multiple steps

Figure 2: *SAgent* workflow demonstration (offline phase). In the offline phase, the *SAgent* is used to gather the experience pool of positive and negative answers over the train subsets.

SAgent uses the *plan step* (cf. Section 3.1.2) to generate a structured step-by-step plan and the *action step*, that either calls the LLM or the *named entity linking (NEL) tool* (cf. Section 3.1.1) ultimately leading to the SPARQL query generation. In the *evaluation (online) phase*, the mKGQAgent is using the plan step and the action step with the experience pool and the NEL tool, and the feedback step that has access to the triplestore.

The important feature of our framework is that it does not require supervised fine-tuning, which significantly reduces the computation costs and preserves the generalizability of the original LLMs (cf. catastrophic forgetting); see Luo et al. [23].

3.1. Offline Phase of the mKGQAgent

3.1.1. Named Entity Linking (NEL) Tool

Likewise, humans look up a resource identifier in a KG, and the NEL step interacts with the environment (i.e., KG) and retrieves resource labels from there. Assuming the fact that an LLM was not given the URI-label mappings of a particular resource, the SPARQL query generation would not be possible. Importantly, while introducing the NEL tool, we do not propose a novel NEL algorithm. In contrast, we demonstrate how to utilize an existing NEL service in the LLM agent workflow (see Algorithm 1).

Algorithm 1 NEL Tool

```
Require: Entity candidates $E$, Relation candidates $R$, NEL service $\mathcal{NEL}$
Ensure: Dictionary with linked entities: linkedEntities
Ensure: Dictionary with linked relations: linkedRelations
 1: Initialize empty dictionaries linkedEntities and linkedRelations
 2: for each entity $e_i \in E$ do
 3:     $e_i^{\text{URI}} \leftarrow \mathcal{NEL}(e_i)$
 4:     if $e_i^{\text{URI}}$ is not empty then
 5:         linkedEntities$[e_i] \leftarrow e_i^{\text{URI}}$
 6:     end if
 7: end for
 8: for each relation $r_j \in R$ do
 9:     $r_j^{\text{URI}} \leftarrow \mathcal{NEL}(e_j)$
10:     if $r_j^{\text{URI}}$ is not empty then
11:         linkedEntities$[r_j] \leftarrow r_j^{\text{URI}}$
12:     end if
13: end for
14: return (linkedEntities, linkedRelations)
```

The entity and relation candidates are proposed by the backbone LLM within the tool calling process at the action step (see Sections 3.1.3 and 3.2.2). Entity and relation linking is crucial for the text-to-SPARQL process since the URIs representing resources in a KG may be done using random identifiers[2].

3.1.2. Plan step

The plan step leverages the backbone LLM to generate a step-by-step list of tasks to come up with a SPARQL query given a question. The intuition behind the plan step is that it simplifies the task for the model such that it does not need to handle the whole complexity at once. For example, such tasks as entity recognition and linking, query refinement, etc. Thus, following the human-like behavior (cf. Huys et al. [25], Correa et al. [8]), the plan step intends to break down the complex task of writing a SPARQL query into a combination of simpler subtasks. Hence, the action step deals at one point in time with a simple subtask having the results of the previous steps in its conversation history. For details regarding the plan step (for details, see Algorithm 2).

Algorithm 2 Plan step without experience pool

```
Require: Natural language question $q_i$, system prompt $S_{\text{plan}}$, model $\mathcal{LLM}$
Ensure: Step-by-step plan $p_i$                                  ▷ List of textual tasks
 1: $p_i \leftarrow \mathcal{LLM}(S_{\text{plan}}, q_i)$          ▷ Query LLM with system prompt and question
 2: return $p_i$
```

3.1.3. Action Step without Experience Pool

Once the plan is generated, the action step executes each of the plan tasks sequentially, leveraging the NEL Tool for the entity linking (see Algorithm 3). This approach ensures that the agent follows the structured plan, interacting with necessary tools to refine and complete the SPARQL query generation process.

[2]e.g., in Wikidata [24], Q567 (https://www.wikidata.org/wiki/Q567) for “Angela Merkel”

Algorithm 3 Action Step without Experience Pool

```
Require: Step-by-step plan $p_i$, model $\mathcal{LLM}$, tool $NEL$, system prompt $S_{\text{action}}$
Ensure: Generated SPARQL query $\hat{\phi}_i$
1: Bind $NEL$ to $\mathcal{LLM}$
2: Initialize empty chat history $\mathcal{H}_i$
3: for each step $s_j \in p_i$ do
4:     $h_j \leftarrow \mathcal{LLM}(S_{\text{action}}, s_j)$                    ▷ LLM may call tool or just itself
5:     Append $h_j$ to $\mathcal{H}_i$
6: end for
7: $\hat{\phi}_i \leftarrow$ lastElementOf($\mathcal{H}_i$)
8: return $\hat{\phi}_i$
```

Algorithm 4 Add Example to the Experience Pool

```
Require: Training set example $d_i \in \mathcal{D}_{\text{train}}$, step-by-step plan $p_i$, chat history $\mathcal{H}_i$, Experience pool $\mathcal{E}$,
    Text embedding model $\mathcal{EMB}$
Ensure: Updated experience pool $\mathcal{E}'$
1: $q_i, \phi_i \leftarrow d_i$                ▷ Unpack training example (question and ground truth SPARQL)
2: $\hat{\phi}_i \leftarrow$ lastElementOf($\mathcal{H}$)                    ▷ Get the SPARQL generated by $\mathcal{SAgent}$
3: $\text{F1}_i \leftarrow F1_{\text{score}}(\phi_i, \hat{\phi}_i)$                                          ▷ Compute F1 score
4: $v_{q_i} \leftarrow \mathcal{EMB}(q_i)$                                   ▷ Convert question to a vector
5: $\mathcal{E}' \leftarrow \mathcal{E} + \{q_i, v_{q_i}, \phi_i, \hat{\phi}_i, p_i, \mathcal{H}, \text{F1}_i\}$
6: return $\mathcal{E}'$
```

3.1.4. Experience Pool Construction

During the offline phase, we utilize $\mathcal{SAgent}$ to collect the *experience pool.* This involves evaluating the correctness of the generated SPARQL queries (based on the ground truth data) and storing them together with the intermediate steps (i.e., plan, chat history) in a vector database (see Algorithm 4). Therefore, each natural language question from the train subset is converted into a vector representation that is associated with metadata, including the corresponding plan, intermediate steps of the action step, and the final results. The experience pool is a non-parametric memory of our agent that contains both successful (F1 score $= 1.0$) and unsuccessful (F1 score < 1.0) SPARQL query generation attempts based on a ground truth.

Therefore, the experience pool holds the information about the quality of the generated SPARQL queries (F1_i), the step-by-step plan (p_i) that was used to generate this particular query, and other metadata (e.g., ground truth SPARQL query).

3.2. Evaluation Phase of the mKGQAgent

3.2.1. Plan step with the Experience Pool

In the evaluation phase, the plan step leverages the experience pool to find relevant plan examples for better planning quality. The plan examples are included in the system prompt S_{plan} (see Algorithm 5).

Hence, the plan step benefits from the prior successful planning examples while using them in the system prompt for in-context learning.

3.2.2. Action step with the Experience Pool

Once the plan is generated, the action step executes each of the plan tasks sequentially, leveraging the NEL Tool for the entity linking (see Algorithm 6). The usage of the experience pool ensures that the LLM benefits from the in-context SPARQL query examples from the training subset. It is important

Algorithm 5 Plan step with Experience Pool

```
Require: Natural language question $q_i$, system prompt $S_{\text{plan}}$, model $\mathcal{LLM}$, experience pool $\mathcal{E}$, text embedding model $\mathcal{EMB}$
Ensure: Step-by-step plan $p_i$                                        ▷ List of textual tasks
 1: $v_{q_i} \leftarrow \mathcal{EMB}(q_i)$
 2: $P \leftarrow findTopNPlans(\mathcal{E}, v_{q_i})$          ▷ Finds top-N similar plans with F1 $= 1.0$
 3: $S^{\text{experience}}_{\text{plan}} \leftarrow S_{\text{plan}} + P$            ▷ The plans are included to the prompt
 4: $p_i \leftarrow \mathcal{LLM}(S^{experience}_{\text{plan}}, q_i)$    ▷ Query LLM with system prompt and question
 5: return $p_i$
```

Algorithm 6 Action Step with the Experience Pool

```
Require: Step-by-step plan $p_i$, model $\mathcal{LLM}$, tool $NEL$, system prompt $S_{\text{action}}$ (see appendix), experience pool $\mathcal{E}$, text embedding model $\mathcal{EMB}$
Ensure: Generated SPARQL query $\hat{\phi}_i$
 1: Bind $NEL$ to $\mathcal{LLM}$
 2: Initialize empty chat history $\mathcal{H}_i$
 3: $v_{q_i} \leftarrow \mathcal{EMB}(q_i)$
 4: $P \leftarrow findTopNQueries(\mathcal{E}, v_{q_i})$         ▷ Finds top-N similar SPARQL queries
 5: $S^{experience}_{\text{action}} \leftarrow S_{\text{action}} + P$        ▷ The queries are included to the prompt
 6: for each step $s_j \in p_i$ do
 7:     $h_j \leftarrow \mathcal{LLM}(S^{experience}_{\text{action}}, s_j)$             ▷ LLM may call tool or just itself
 8:     Append $h_j$ to $\mathcal{H}_i$
 9: end for
10: $\hat{\phi}_i \leftarrow$ lastElementOf($\mathcal{H}_i$)
11: return $\hat{\phi}_i$
```

Algorithm 7 Feedback Step

```
Require: Intermediate query $\phi_i$, prompt template $S_{\text{feedback}}$ (see appendix), triplestore $\mathcal{KG}$
Ensure: Feedback prompt $S'_{\text{feedback}}$
 1: $\mathcal{A}_i \leftarrow \mathcal{KG}(\phi_i)$                ▷ Query the triplestore and get the response
 2: $S'_{\text{feedback}} \leftarrow S_{\text{feedback}} + \mathcal{A}_i$            ▷ Populate prompt with the response
 3: return $S'_{\text{feedback}}$
```

to note that the plan p_i can be populated with the result of the feedback step (in case the feedback is triggered).

3.2.3. Feedback Step

The feedback executes the generated SPARQL query ϕ on a triplestore, collects the response, and integrates it into a pre-defined prompt template for the action step. Once the first version of a SPARQL query is generated (i.e., the result of the last planning step executed at the action step), it is redirected to the feedback step. The feedback is formulated only once per input question, i.e., there are no multiple feedback options intended to avoid infinite loops. The detailed feedback step workflow is defined in Algorithm 7. After that, the feedback S'_{feedback} is redirected to the action step. The action step executes the feedback to refine the SPARQL query and delivers the final query as the result.

4. Experimental Setup

We conduct our experiments on the commonly used KGQA benchmark: QALD-9-plus (Perevalov et al. [9]). QALD-9-plus contains 558 questions in multiple languages and queries over DBpedia [26] and Wikidata cf. [27]. We consider all available languages from QALD-9-plus, in addition, we also take the Spanish questions, which were contributed to this dataset later (Soruco et al. [28]). The structure of QALD-9-plus includes question texts and the corresponding ground truth SPARQL queries that return the expected answer to a question. For the evaluation of KGQA quality, we use the Macro F1 score [29].

4.1. Large Language Models and Text Embedding Models

In this work, we use both open-source and proprietary LLMs. The proprietary ones are provided by OpenAI[3], namely, GPT-3.5 (`gpt-3.5-turbo-0125`), and GPT-4o (`gpt-4o-2024-05-13`). The models are accessed via the official Python SDK[4] with `temperature=0`, and other hyperparameters are set to default.

The open-source LLMs are: Qwen2.5 72B Instruct[5] and Meta Llama 3.1 70B Instruct[6]. Both models were used with the AWQ (Lin et al. [30]) quantization (4-bit) to fit into the memory. The models were deployed via the vLLM framework (Kwon et al. [31]). The maximal context size of the models was set to 16384 tokens to avoid out-of-memory exceptions. The other hyperparameters were set to default. For the open-source LLMs, we used a server with two Nvidia L40S GPUs (each 48GB VRAM).

For creating text embeddings for the experience pool, we used a specific model trained for producing high-quality text embeddings for multilingual input – `multilingual e5 large`[7] (introduced by Wang et al. [32]). According to the MTEB leaderboard[8] introduced by Muennighoff et al. [33], the model is listed among the top-3 in different languages (we considered embedding models with a size smaller than 1 billion parameters).

4.2. Implementation of mKGQAgent

The mKGQAgent *architecture* is implemented within the LangChain framework[9] in Python. This framework facilitates the integration of various components required for the agent's functionality.

The *entity linking* within the NEL tool is implemented via Wikidata's official public entity lookup endpoint[10]. This endpoint is capable of handling input in multiple languages. The NEL tool also uses an *external relation linker*, Falcon 2.0 (Sakor et al. [34]), for enhanced linking capabilities.

The *routing* between the plan, action, and feedback is implemented within the LangGraph framework[11], which is part of the LangChain ecosystem.

The *prompts* used within the mKGQAgent are written in different languages, s.t., they match the input question language. The prompts in English, German, and Russian were written by native speakers, the other prompts were acquired via machine translation and further structure validation. We list the prompts in Figure 3.

The SPARQL queries generated by the mKGQAgent are executed on the official Wikidata SPARQL endpoint[12].

[3]https://platform.openai.com/docs/models
[4]https://github.com/openai/openai-python
[5]https://huggingface.co/Qwen/Qwen2-72B-Instruct-AWQ
[6]https://huggingface.co/hugging-quants/Meta-Llama-3.1-70B-Instruct-AWQ-INT4
[7]https://huggingface.co/intfloat/multilingual-e5-large
[8]https://huggingface.co/spaces/mteb/leaderboard
[9]https://python.langchain.com
[10]https://www.wikidata.org/w/api.php?action=wbsearchentities
[11]https://langchain-ai.github.io/langgraph/
[12]https://query.wikidata.org/bigdata/namespace/wdq/sparql

```
You are an intelligent Knowledge Graph-based Question Answering system.

You can use the tools to help yourself only if you \
DON't have this information in chat history:
- 'wikidata_el' for named entity linking  \
(e.g. "Person name" -> "URI" or "is child of" -> "URI")
to determine URIs in the Wikidata KG

{QUESTION_QUERY_EXAMPLE} # comes from the experience pool
```

(a) The system prompt ($S^{\text{experience}}_{action}$) for the action step with the usage of the experience pool $\mathcal{E}$

```
For the given objective, come up with a simple step by step plan to write a SPARQL query.
This plan should involve individual tasks (e.g., named entity linking, relation linking,
expected answer type classification), that if executed correctly \
will yield the correct SPARQL.
Do not add any superfluous steps.
The result of the final step should be the final  SPARQL query.
Don't propose to execute the query.
At the end step you MUST output exactly **ONE** SPARQL query string \
**without extra text or markdown**.

{USER_QUESTION}
{PLAN_EXPERIENCE_EXAMPLE} # comes from the experience pool
```

(b) The planning prompt ($S^{\text{experience}}_{\text{plan}}$) for the plan step with the usage of the experience pool $\mathcal{E}$

```
This is feedback to your generated SPARQL query produced by executing it on a triplestore.
Please rework your query if neccessary.

Initial question: {USER_QUESTION}
Your query: {GENERATED_SPARQL}# intermediate SPARQL query

--- Start triplestore response ---
{FEEDBACK}# comes from the query execution on a triplestore
--- End triplestore response ---

Make sure that the query is formatted correctly.
No extra text. No markdown. Just plain SPARQL.
Determine whether to output a URI (SELECT ?uri), number (COUNT), date, \
boolean (ASK), string (SELECT ?label).
DON'T USE "SERVICE wikibase:label"
```

(c) The feedback prompt (S_{feedback}) for the feedback step

Figure 3: The English versions of the prompts used within the mKGQAgent. Placeholders are color-coded blue. Comments are color-coded red. Important: We use language-specific prompts for every considered language. In case of the evaluation within Text2SPARQL challenge, we used English prompts only despite the input language.

4.3. Baselines

To compare the performance of the mKGQAgent we select both "pre-LLM era" KGQA systems and the ones that use different prompting techniques with LLMs. Also, the baselines were selected in a way that they can generate SPARQL queries over Wikidata. In particular, the following approaches are selected for comparison with ours: QAnswer, Platypus, DeepPavlov 2023, KGQAN, Triad, MST5, and HQA (cf. Section 2).

The selection of the baselines was also influenced by the results reported in the KGQA leaderboard by Perevalov et al. [10]. We reuse the reported results in our paper for comparison with our mKGQAgent approach.

4.4. Machine Translation of the Input

Following our research agenda [35, 36], we evaluate how well machine translation to English serves as an alternative to processing non-English questions natively with the OPUS MT models; cf. Tiedemann

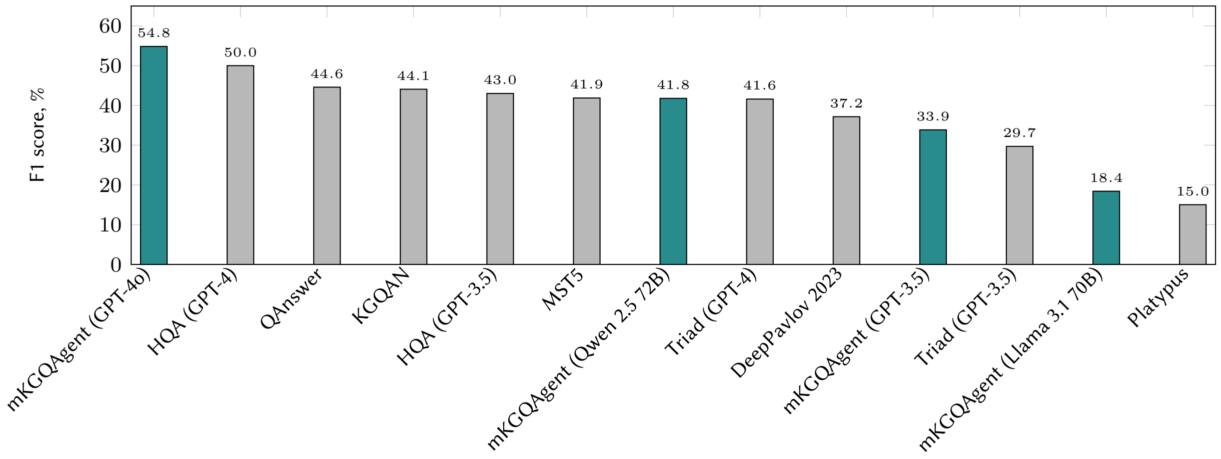

Figure 4: Comparison between our mKGQAgent approach (teal) and the baselines (grey) on English questions of QALD-9-plus.

Table 1
Evaluation results of mKGQAgent (our approach) compared to the baselines that support multiple languages. They were conducted on the test subset of the QALD-9-plus. The best results per language are highlighted in bold.

	English	German	Spanish	French	Russian	Belarusian	Ukrainian	Lithuanian	Bashkir	Armenian
QAnswer	44.59	31.71	16.8	**23.00**	21.43			N/A		
Platypus	15.03	N/A		4.17			N/A			
DeepPavlov 2023	37.16		N/A		31.17			N/A		
MST5	41.87	41.26	N/A	41.67	**37.61**	29.07	**34.67**	**31.15**	18.42	N/A
mKGQAgent (GPT-3.5)	33.85	25.85	22.08	22.87	11.75	16.05	12.36	16.27	6.63	8.88
mKGQAgent (GPT-4o)	**54.83**	**43.08**	**38.28**	22.76	31.67	**31.56**	28.54	25.54	**40.48**	9.09
mKGQAgent (Qwen 2.5 72B)	41.77	21.86	17.96	17.60	12.92	10.81	10.86	9.56	10.40	0.00
mKGQAgent (Llama 3.1 70B)	18.42	19.64	5.23	17.70	5.22	5.73	6.70	9.52	4.65	15.07

and Thottingal [37].

Our machine translation experiments are complementary to the main contribution and, therefore, are limited to the German, Russian, and Spanish languages. We selected these languages as they all represent different language branches—the Germanic, Slavic, and Romance, respectively.

5. Evaluation and Analysis

5.1. English-only Comparison with the Baselines

The results presented in Figure 4 illustrate a comparative analysis between our mKGQAgent approach (highlighted in teal) and various baseline methods (depicted in grey) on the English questions from the QALD-9-plus benchmark. Our mKGQAgent (GPT-4o) achieves the highest F1 score of 54.83%, surpassing all baselines, including HQA (GPT-4), which attains 50.00%. This demonstrates the effectiveness of our approach in leveraging structured planning and retrieval mechanisms to enhance semantic parsing performance.

Among the baselines, QAnswer (44.59%) and KGQAN (44.07%) show competitive results but still fall short of our top-performing model. Interestingly, HQA (GPT-3.5) achieves 43.00%, indicating that the transition to GPT-4 has significantly improved query generation capabilities. The performance of mKGQAgent (Qwen 2.5 72B) (41.87%) and Triad (GPT-4) (41.77%) suggests that large models, even with structured workflows, benefit from additional fine-tuning and experience pooling. Notably, our mKGQAgent (GPT-3.5) variant scores just 37.16%, still outperforming several baselines but trailing behind its GPT-4o counterpart.

Table 2
Evaluation results of the mKGQAgent (showing F1 score and percentages): A comparison between the quality of original language questions and the same questions translated into English using machine translation (MT).

	German		Russian		Spanish	
	Native	MT	Native	MT	Native	MT
mKGQAgent (GPT-3.5)	25.85	28.23	11.75	27.93	22.08	27.67
mKGQAgent (GPT-4o)	43.08	35.66	31.67	35.66	38.28	44.18
mKGQAgent (Qwen 2.5 72B)	21.86	30.95	12.92	32.97	17.96	31.35
mKGQAgent (Llama 3.1 70B)	19.64	17.29	5.22	17.29	5.23	9.63

Table 3
A comparison between the MT performance against the native questions (original language in the header row). We demonstrate the relative improvement when using MT (▲) or deterioration (▼) in terms of F1 score.

	German	Russian	Spanish
mKGQAgent (GPT-3.5)	+9.21% ▲	+137.69% ▲	+25.31% ▲
mKGQAgent (GPT-4o)	-17.22% ▼	+12.59% ▲	+15.42% ▲
mKGQAgent (Qwen 2.5 72B)	+41.58% ▲	+155.21% ▲	+74.52% ▲
mKGQAgent (Llama 3.1 70B)	-11.97% ▼	+231.46% ▲	+84.03% ▲

5.2. Multilingual Comparison with the Baselines

In Table 1, we present the evaluation results of our approach in comparison to the selected baselines that support multiple languages (see Section 4.3).

The experimental results demonstrate mKGQAgent's robust performance across multiple languages on the QALD-9-plus benchmark. When implemented with GPT-4o, the system achieves state-of-the-art results with the F1 scores of 54.83%, 43.08%, 38.28%, 31.56%, and 40.48%, respectively, for English, German, Spanish, Belarusian, and Bashkir. The languages using Cyrillic-based scripts (Russian, Belarusian, Ukrainian, and Bashkir) generally yield poorer results in comparison to the languages using Latin-based scripts.

While comparing mKGQAgent to the other baselines, we see that QAnswer outperforms mKGQAgent (GPT-3.5) on French; however, the difference is not substantial (23.00% vs 22.87%). The MST5 system significantly outperforms mKGQAgent (GPT-4o) on Russian (37.61% vs 31.67%), Ukrainian (34.67% vs 28.54%), and Lithuanian (25.54% vs 31.15%).

5.3. Machine Translation for non-English Questions

The evaluation compares the performance of the models on native-language questions and those translated into English using machine translation (see Table 2). Across all models and languages, the performance of mKGQAgent is generally higher for translated questions than for native-language questions. This suggests that translating non-English questions into English before processing yields better results.

The mKGQAgent based on GPT-4o achieves the highest performance across all settings, demonstrating the superior comprehension and reasoning capabilities of this model. Qwen 2.5 72B exhibits strong performance in translation-based settings but falls behind GPT-4o. The variance in performance between languages suggests that translation quality and linguistic characteristics play a role in how effectively mKGQAgent can process and answer questions. In general, this comparison demonstrates that the translation of non-English questions into English consistently improves the quality and underlines the unequal quality distribution among the languages.

Table 3 presents a comparative evaluation of the performance of MT against questions in their native language within the KGQA task. The results demonstrate that, for most models and languages MT yields improvements over native-language question answering. This effect is particularly pronounced in Russian and Spanish, where MT provides significant gains. GPT-4o, despite its strong overall performance, exhibits slight performance degradation when using MT for German (-17.22%), suggesting that this model may already be well optimized for handling German-language queries natively. Overall, these findings highlight the advantages of translations in multilingual KGQA, even when objectively strong LLMs are used.

5.4. LLM Calls and Costs

An important aspect of using LLM agent frameworks is the number of model calls within one task solution, i.e., in our case, we report the number of calls per generated SPARQL query for an input question. In addition, we report the estimated number of tokens per question and the underlying costs of the LLMs' usage.

5.4.1. Costs calculation for the OpenAI models

According to our calculations, mKGQAgent *requires 13.03 LLM calls on average* to generate a SPARQL query for an input question (cf. Table 4). Consequently, every LLM call consumes 144 input and 199 output tokens on average. This includes chat history that gradually grows during agent execution. The pricing strategy of OpenAI is based on token consumption. Therefore, we calculate the token-based price (TBP) as in Equation 1.

$$\text{TBP} = \big[(n_i \times p_i) + (n_o \times p_o)\big] \times n_c \times n_q \tag{1}$$

Where: n_i represents the number of input tokens, p_i represents the price per input token, n_o represents the number of output tokens, p_o represents the price per output token, n_c represents the number of LLM calls per question, n_q represents the number of questions. This results in USD 0.48 per 100 questions for the GPT-3.5 and USD 3.06 per 100 questions for the GPT-4o, respectively (prices as of March 01, 2025). For the costs of the different $\mathcal{SA}gent$ configurations, see Table 4.

5.4.2. Costs calculation for the open source models

For the open-source LLMs, we use the same values regarding the average number of LLM calls to generate a SPARQL query (13.03) and the average number of tokens per call–144 input tokens and 199 output tokens. The pricing of open source models relies on the GPU hours of cloud providers and the model efficiency measured in tokens per second (tok/sec). Therefore, we calculate the GPU hours-based price (GBP) as in Equation 2.

$$\text{GBP} = \frac{n_q \times n_c \times n_o}{r_{\text{tok/sec}}} \times p_{\text{GPU/sec}} \tag{2}$$

Where: $r_{\text{tok/sec}}$ represents model efficiency rate (tok/sec), $p_{\text{GPU/sec}}$ price per GPU-second. We estimated the market prices of our GPU experimental setup (2x Nvidia L40S GPU) according to one of the well-known cloud providers[13]. The model performance (tok/sec) was retrieved from the official documentation of the respective models[14] taking into account the usage of the vLLM framework for deployment and the size of the context window – 16384 tokens. Hence, for processing 100 questions, mKGQAgent requires 1.96 GPU hours when using the Qwen model and 0.97 GPU hours when using the Llama model. Therefore, the prices per 100 questions are USD 4.05 and 2.01, respectively. For the costs of the different $\mathcal{SA}gent$ configurations, see Table 4.

5.5. Impact of Individual Components on the Quality (Ablation)

To understand the contribution of each architectural component to the overall system performance, we conducted an ablation study using the English questions from the QALD-9-plus dataset. As our baseline system, we consider the $\mathcal{SA}gent$ with plan step and NEL tool components.

[13] https://www.runpod.io/pricing
[14] Qwen: https://qwen.readthedocs.io/en/latest/benchmark/speed_benchmark.html,
Llama: https://artificialanalysis.ai/models/llama-3-1-instruct-70b

Table 4
Impact (regarding the baseline evaluation – *SAgent* (Plan step + NEL tool) of individual components on the QA quality (F1 score) measured on English questions of QALD-9-plus dataset. The strategy of pricing calculation is described in Section 5.4. ▲ – increases (the higher, the better), ▲ – increases (the higher the worse).

Backbone LLM	F1 score, %	Impact	LLM calls	Price per 100 questions	Price impact
SAgent (Plan step + NEL tool)					
GPT-3.5	23.15	N/A	8.87 (avg.)	USD 0.33	N/A
GPT-4o	34.37			USD 2.08	
Qwen 2.5 72B Instruct	24.87			USD 2.75	
Llama 3.1 70B Instruct	8.12			USD 1.37	
SAgent + Experience Pool for Plan Step					
GPT-3.5	31.17	+34.64% ▲	9.71 (avg.)	USD 0.36	+9.31% ▲ (avg.)
GPT-4o	46.48	+35.23% ▲		USD 2.28	
Qwen 2.5 72B Instruct	29.45	+18.42% ▲		USD 3.02	
Llama 3.1 70B Instruct	17.06	+110.09% ▲		USD 1.49	
SAgent + Experience Pool for Action Step					
GPT-3.5	26.97	+16.50% ▲	9.02 (avg.)	USD 0.33	+1.73% ▲ (avg.)
GPT-4o	52.68	+53.27% ▲		USD 2.12	
Qwen 2.5 72B Instruct	32.87	+32.17% ▲		USD 2.80	
Llama 3.1 70B Instruct	15.58	+91.87% ▲		USD 1.39	
SAgent + Feedback step					
GPT-3.5	26.91	+16.24% ▲	10.93 (avg.)	USD 0.40	+22.65% ▲ (avg.)
GPT-4o	40.47	+17.74% ▲		USD 2.57	
Qwen 2.5 72B Instruct	25.72	+3.42% ▲		USD 3.39	
Llama 3.1 70B Instruct	9.48	+4.43% ▲		USD 1.68	
mKGQAgent (Plan step + NEL tool + Experience Pool for Plan and Action + Feedback step)					
GPT-3.5	33.85	+46.22% ▲	13.03 (avg.)	USD 0.48	+46.62% ▲ (avg.)
GPT-4o	54.83	+59.53% ▲		USD 3.06	
Qwen 2.5 72B Instruct	41.77	+67.95% ▲		USD 4.05	
Llama 3.1 70B Instruct	20.42	+151.47% ▲		USD 2.01	

The integration of the experience pool for the plan step demonstrated substantial improvements across all models. Notably, Llama exhibited the most substantial relative improvement of 110%.

The addition of the experience pool to the action step proved particularly effective, especially when combined with GPT-4o, pushing the F1 score to 52.68% (a 53.27% improvement over baseline). This component's impact was consistently positive across all models.

The feedback step introduced more modest but still significant improvements. GPT-4o's performance with this component reached 40.47%, representing a 17.74% increase over the baseline.

The full integration of all components in mKGQAgent yielded the most impressive results, with GPT-4o achieving a peak F1 score of 54.83%, marking a 59.53% improvement over the baseline. This comprehensive integration demonstrated synergistic effects across all models, with even the initially lower-performing Llama showing a remarkable improvement of 151.47% (see Table 4). These results strongly indicate that the combination of all components creates a more robust and effective KGQA system.

6. Discussion, Research Questions and Limitations

RQ1 The analysis reveals that the proposed agent architecture enables more accurate SPARQL query generation. In particular, mKGQAgent achieved state-of-the-art results (54.83% F1 score) on English and demonstrated superior quality on German, Spanish, Belarusian, and Bashkir.

RQ2 The evaluation indicates that the full mKGQAgent setup with all components achieved substantially better quality but requires additional computational resources. For example, the *SAgent* requires 8.87 LLM calls on average to achieve the end goal, while the final mKGQAgent requires 13.03 LLM calls on average.

RQ3 Our work indicates that multilingual SPARQL generation presents significant challenges even to the state-of-the-art LLMs. In particular, even among European languages, the quality of SPARQL query generation may degrade by more than a factor of three.

$\mathcal{RQ}4$ Our results indicate that machine translation generally leads to higher KGQA performance compared to processing questions in their native languages. However, the effectiveness of translation-based approaches varies by language and model.

We acknowledge several limitations of our work. Since our evaluation relies on Wikidata-based datasets, it may not fully capture the ability of LLMs to generalize or compose SPARQL queries for previously unseen data. The issue of data memorization was not the primary focus of this study; however, we address it in a separate publication [38].

7. Conclusion

The paper introduces a novel LLM agent framework called mKGQAgent for the multilingual Text-to-SPARQL task. The experiments carried out have shown that each step of the mKGQAgent workflow contributes positively to the quality of the results. The mKGQAgent substantially outperforms previous systems in the English, German, Spanish, Belarusian, and Bashkir questions of the QALD-9-plus data set.

We highlighted significant challenges when LLMs deal with non-English languages, especially low-resource ones. The latter challenge can be partially covered by the use of MT techniques, which was demonstrated by our experiments. However, the use of different translation techniques requires further systematic study to identify settings where each of them performs best. Despite this, the mKGQAgent framework demonstrates a promising approach to KGQA by adopting the LLM agent paradigm. While it shows its ability to work with multiple languages having reasonably good quality, we also demonstrated the trade-off between the quality and computational costs that increase with the agent paradigm adoption.

Acknowledgments

This work has been partially supported by grants for the ITZBund[15]-funded research project "QA4CB—Entwicklung von Question-Answering-Komponenten zur Erweiterung des Chatbot-Frameworks" at the Leipzig University of Applied Sciences in Leipzig (Germany).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT by OpenAI in order to: Grammar and spelling check. After using this service, the authors reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] D. Diefenbach, P. H. Migliatti, O. Qawasmeh, V. Lully, K. Singh, P. Maret, QAnswer: A question answering prototype bridging the gap between a considerable part of the lod cloud and end-users, in: The World Wide Web Conference, WWW '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 3507–3510. doi:10.1145/3308558.3314124.

[2] R. Turganbay, V. Surkov, D. Evseev, M. Drobyshevskiy, Generative question answering systems over knowledge graphs and text, volume 22, 2023, pp. 1112–1126. doi:10.28995/2075-7182-2023-22-1112-1126.

[3] N. Srivastava, M. Ma, D. Vollmers, H. Zahera, D. Moussallem, A.-C. N. Ngomo, MST5–multilingual question answering over knowledge graphs, arXiv preprint arXiv:2407.06041 (2024).

[15] https://www.itzbund.de/

[4] J. Jiang, K. Zhou, W. X. Zhao, Y. Song, C. Zhu, H. Zhu, J.-R. Wen, KG-Agent: An efficient autonomous agent framework for complex reasoning over knowledge graph, arXiv preprint arXiv:2402.11163 (2024).

[5] X. Huang, S. Cheng, S. Huang, J. Shen, Y. Xu, C. Zhang, Y. Qu, QueryAgent: A reliable and efficient reasoning framework with environmental feedback based self-correction, arXiv preprint arXiv:2403.11886 (2024).

[6] Y. Li, Y. Zhang, L. Sun, MetaAgents: Simulating interactions of human behaviors for LLM-based task-oriented coordination via collaborative generative agents, CoRR abs/2310.06500 (2023). doi:10.48550/ARXIV.2310.06500. arXiv:2310.06500.

[7] D. Diefenbach, K. Singh, A. Both, D. Cherix, C. Lange, S. Auer, The Qanary ecosystem: Getting new insights by composing question answering pipelines, in: J. Cabot, R. De Virgilio, R. Torlone (Eds.), Web Engineering, Springer International Publishing, Cham, 2017, pp. 171–189.

[8] C. G. Correa, M. K. Ho, F. Callaway, T. L. Griffiths, Resource-rational task decomposition to minimize planning costs, in: Proceedings of the 42th Annual Meeting of the Cognitive Science Society - Developing a Mind: Learning in Humans, Animals, and Machines, CogSci 2020, virtual, July 29 - August 1, 2020, cognitivesciencesociety.org, 2020. URL: https://cogsci.mindmodeling.org/2020/papers/0746/index.html.

[9] A. Perevalov, D. Diefenbach, R. Usbeck, A. Both, QALD-9-plus: A multilingual dataset for question answering over DBpedia and Wikidata translated by native speakers, in: 2022 IEEE 16th International Conference on Semantic Computing (ICSC), IEEE, 2022, pp. 229–234.

[10] A. Perevalov, X. Yan, L. Kovriguina, L. Jiang, A. Both, R. Usbeck, Knowledge graph question answering leaderboard: A community resource to prevent a replication crisis, in: Proceedings of the Language Resources and Evaluation Conference, European Language Resources Association, Marseille, France, 2022, pp. 2998–3007. URL: https://aclanthology.org/2022.lrec-1.321.

[11] A. Perevalov, A. Both, A.-C. Ngonga Ngomo, Multilingual question answering systems for knowledge graphs—a survey, Semantic Web 15 (2024) 2089–2124.

[12] D. Punjani, K. Singh, A. Both, M. Koubarakis, I. Angelidis, K. Bereta, T. Beris, D. Bilidas, T. Ioannidis, N. Karalis, C. Lange, D. Pantazi, C. Papaloukas, G. Stamoulis, Template-based question answering over linked geospatial data, in: Proceedings of the 12th Workshop on Geographic Information Retrieval, GIR'18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3281354.3281362. doi:10.1145/3281354.3281362.

[13] T. Pellissier Tanon, M. D. de Assunção, E. Caron, F. M. Suchanek, Demoing Platypus – a multilingual question answering platform for Wikidata, in: A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, M. Alam (Eds.), The Semantic Web: ESWC 2018 Satellite Events, Springer International Publishing, Cham, 2018, pp. 111–116.

[14] R. Omar, I. Dhall, P. Kalnis, E. Mansour, A universal question-answering platform for knowledge graphs, Proceedings of the ACM on Management of Data 1 (2023) 1–25.

[15] Y. Zhou, X. Geng, T. Shen, W. Zhang, D. Jiang, Improving zero-shot cross-lingual transfer for multilingual question answering over knowledge graph, in: K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, Y. Zhou (Eds.), Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 5822–5834. URL: https://aclanthology.org/2021.naacl-main.465/. doi:10.18653/v1/2021.naacl-main.465.

[16] Y. Zhang, J. Wang, Z. Wang, R. Zhang, XSemPLR: Cross-lingual semantic parsing in multiple natural languages and meaning representations, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Toronto, Canada, 2023, pp. 15918–15947. URL: https://aclanthology.org/2023.acl-long.887.

[17] Y. Tan, X. Zhang, Y. Chen, Z. Ali, Y. Hua, G. Qi, CLRN: A reasoning network for multi-relation question answering over cross-lingual knowledge graphs, Expert Systems with Applications 231 (2023) 120721. URL: https://www.sciencedirect.com/science/article/pii/S095741742301223X.

doi:https://doi.org/10.1016/j.eswa.2023.120721.

[18] C. Zong, Y. Yan, W. Lu, J. Shao, Y. Huang, H. Chang, Y. Zhuang, Triad: A framework leveraging a multi-role LLM-based agent to solve knowledge base question answering, in: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024, pp. 1698–1710.

[19] J. Lehmann, D. Bhandiwad, P. Gattogi, S. Vahdati, Beyond boundaries: A human-like approach for question answering over structured and unstructured information sources, Transactions of the Association for Computational Linguistics 12 (2024) 786–802.

[20] G. Xiong, J. Bao, W. Zhao, Interactive-KBQA: Multi-turn interactions for knowledge base question answering with large language models, CoRR abs/2402.15131 (2024). doi:10.48550/ARXIV.2402.15131.

[21] G. Mialon, R. Dessi, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Roziere, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz, et al., Augmented language models: a survey, Transactions on Machine Learning Research (2023).

[22] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, et al., A survey on large language model based autonomous agents, Frontiers of Computer Science 18 (2024) 186345. doi:10.1007/s11704-024-40231-1.

[23] Y. Luo, Z. Yang, F. Meng, Y. Li, J. Zhou, Y. Zhang, An empirical study of catastrophic forgetting in large language models during continual fine-tuning, arXiv preprint arXiv:2308.08747 (2023).

[24] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85. doi:10.1145/2629489.

[25] Q. J. Huys, N. Lally, P. Faulkner, N. Eshel, E. Seifritz, S. J. Gershman, P. Dayan, J. P. Roiser, Interplay of approximate planning strategies, Proceedings of the National Academy of Sciences 112 (2015) 3098–3103. doi:10.1073/pnas.1414219112.

[26] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, DBpedia: A nucleus for a web of open data, in: The semantic web, Springer, 2007, pp. 722–735.

[27] D. Vrandečić, M. Krötzsch, Wikidata: A free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85. doi:10.1145/2629489.

[28] J. Soruco, D. Collarana, A. Both, R. Usbeck, QALD-9-ES: A Spanish dataset for question answering systems, in: Knowledge Graphs: Semantics, Machine Learning, and Languages, IOS Press, 2023, pp. 38–52.

[29] R. Usbeck, M. Röder, M. Hoffmann, F. Conrads, J. Huthmann, A.-C. Ngonga-Ngomo, C. Demmler, C. Unger, Benchmarking question answering systems, Semantic Web 10 (2019) 293–304.

[30] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, S. Han, AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration, in: P. Gibbons, G. Pekhimenko, C. D. Sa (Eds.), Proceedings of Machine Learning and Systems, volume 6, 2024, pp. 87–100. URL: https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf.

[31] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, I. Stoica, Efficient memory management for large language model serving with pagedattention, in: Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

[32] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, F. Wei, Multilingual E5 text embeddings: A technical report, arXiv preprint arXiv:2402.05672 (2024).

[33] N. Muennighoff, N. Tazi, L. Magne, N. Reimers, MTEB: Massive text embedding benchmark, arXiv preprint arXiv:2210.07316 (2022).

[34] A. Sakor, K. Singh, A. Patel, M.-E. Vidal, Falcon 2.0: An entity and relation linking tool over Wikidata, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, CIKM ’20, Association for Computing Machinery, New York, NY, USA, 2020, p. 3141–3148. doi:10.1145/3340531.3412777.

[35] A. Perevalov, A. Both, D. Diefenbach, A.-C. Ngonga Ngomo, Can machine translation be a reasonable alternative for multilingual question answering systems over knowledge graphs?, in: Proceedings of the ACM Web Conference 2022, WWW ’22, Association for Computing Machinery, New York, NY, USA, 2022, p. 977–986. URL: https://doi.org/10.1145/3485447.3511940. doi:10.1145/

3485447.3511940.

[36] N. Srivastava, A. Perevalov, D. Kuchelev, D. Moussallem, A.-C. Ngonga Ngomo, A. Both, Lingua franca – entity-aware machine translation approach for question answering over knowledge graphs, in: Proceedings of the 12th Knowledge Capture Conference 2023, K-CAP '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 122–130. URL: https://doi.org/10.1145/3587259.3627567. doi:10.1145/3587259.3627567.

[37] J. Tiedemann, S. Thottingal, OPUS-MT — Building open translation services for the World, in: Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT), Lisbon, Portugal, 2020.

[38] A. Gashkov, A. Perevalov, M. Eltsova, A. Both, Sparql query generation with llms: Measuring the impact of training data memorization and knowledge injection, 2025. URL: https://arxiv.org/abs/2507.13859. arXiv:2507.13859.