

RoboData: Toward Trustable Question Answering over Ontologies through Metacognitive Agentic Epistemology

Emanuele Musumeci¹, Vincenzo Suriani² and Daniele Nardi¹

¹Sapienza University of Rome, Department of Computer, Control and Management Engineering, Italy

²University of Basilicata, Department of Engineering, Italy

Abstract

Large Language Models (LLMs) have revolutionized Natural Language Processing (NLP), simplifying knowledge extraction from structured and unstructured sources. Despite pervasive usage in various applications, from Question Answering (QA) to goal-driven reasoning, they tend to produce *hallucinations*, factually incorrect responses that hindering accuracy and explainability of Knowledge-Based QA (KBQA) tasks. To address this limitation, we introduce RoboData, an Agentic AI approach to verifiable knowledge extraction and reasoning over structured ontologies like Wikidata. Through metacognitive self-reflection and goal-directed commonsense reasoning, a LLM-based epistemic agent dynamically self-orchestrates a query answering process. Unlike traditional information retrieval systems, the proposed architecture incrementally builds a local knowledge graph from remote knowledge sources to answer a natural language query with traceable facts, highlighting a "support set" for each claim, a set of nodes and edges in the local knowledge graph that backs the generated claims. The resulting accumulated knowledge provides an intermediate *explainability layer*, providing a reliable epistemic substrate for using trusted ontologies in goal-driven query answering applications, such as robotic planning and semantic map enrichment. Code is available at: <https://emanuelemusumeci.github.io/RoboData/>

Keywords

Agentic AI, Large Language Models, Knowledge Graph Query Answering, Wikidata

1. Introduction

LLMs have reshaped knowledge access and generation. Through commonsense reasoning, they act both as a source of knowledge and as universal regressors for NLP tasks, providing, for instance, implicit heuristics for extracting knowledge from structured and unstructured sources. Scaling laws of LLM capabilities show steady improvements on simple tasks [1] with an increase in the number of parameters and training datasets (both in size and complexity), yet their reliability on complex or highly factual tasks remains challenging [2], especially when specialized knowledge is involved. Recent efforts have sought to align LLMs with more specialized knowledge, through fine-tuning. Instruction tuning [3], while resource-intensive, has proven to be the best way to give new skills to foundational LMs. However, this is not an

Wikidata'25: Wikidata Workshop 2025 at ISWC 2025

✉ musumeci@diag.uniroma1.it (E. Musumeci); vincenzo.suriani@unibas.it (V. Suriani); nardi@diag.uniroma1.it (D. Nardi)

🌐 <https://sites.google.com/view/emanuelemusumeci/home> (E. Musumeci)



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

CEUR Workshop Proceedings (CEUR-WS.org)

option when the knowledge is constantly updated, as it would require continuous post-training LLM alignment. In-context Learning [4, 3], offers a reliable alternative to fine-tuning for both unstructured [5] and structured data [6]. Methods like ReAct [7] prove that LLMs can act "agentially": orienting their reasoning to plan and execute tasks strategically [8], performing actions with real effects on the execution environment. Agentic AI provides the means for unsupervised goal-driven knowledge collection and integration. Despite their potential, LLMs are limited by the size of their context windows (the maximum number of allowed input tokens): long contexts increase hallucination risks [2]. This problem can be mitigated by accurately engineering prompts to provide only relevant contextual information to the LLM agent [5] or by better organizing the task into a sequence or chain of thoughts [9, 10, 11]. Recent works about Knowledge Graph QA (KGQA) are based on the Data-Information-Knowledge-Wisdom (DIKW) paradigm [12], modeled as a pyramid. At the base, the *Data* layer contains unprocessed knowledge (facts), providing an informational grounding layer. The *Information* layer, more structured, organized (e.g. relations) and contextualized (e.g. semantics) acts as a substrate for complex reasoning and inference (e.g. Knowledge Graphs). In the *Knowledge* layer, information is synthetically integrated through explicit procedures or heuristics, or implicit criteria as in LLMs (e.g. Query Answering over KGs). Finally, the *Wisdom* layer ethically aligns knowledge, ensures trustability among other things. In KGQA, Data and Information layers are represented by the KG, while the two upper layers feature a LLM-based pipeline. In fact, commonsense reasoning in LLMs provides an implicit heuristic for all tasks requiring knowledge generalization. While the "Knowledge" abstraction is plausibly embodied by LLMs, it is more arguable that these systems may embody the "Wisdom" abstraction as well. Due to the undesirable behaviors exhibited by the latest generation of LLMs, including opaque agentic behaviors, or manipulative tendencies [7], ethical alignment is needed, but this may not be enough for solving trustability-critical tasks. To address answer verifiability in LLM-based KGQA, we present RoboData, a novel Agentic AI system for Query Answering, grounded on a local Knowledge Graph. RoboData actively performs knowledge extraction and reasoning over structured KGs, constructing a localized and verifiable epistemic substrate to support robust query resolution, by agentially exploring remote ontologies and fetching necessary information to build "*support sets*", subsets of the local Knowledge Graph directly referenced in the generated answer as ground truth for each generated claim, with the ultimate goal of allowing more trustable systems based on both commonsense and specialized knowledge. We present preliminary evidence of RoboData's potential as an epistemic trustability layer for Agentic AI through the following contributions:

- The architecture shown in Figure 1, where a LLM-based agent orchestrates a Query Answering effort. The process produces both a local KG, built from remote reliable data sources, and an answer, supported by "*support sets*", subsets of the local KG.
- An agent capable of reasoning through self-reflection on its past actions, dynamically correcting its strategy by reflecting on its own course of action.

We argue that this metacognitive approach, based on commonsense reasoning, may allow managing more diverse data sources in the future, even ephemeral or unreliable ones. Agentic, verifiable QA systems like RoboData represent a crucial step toward reliable reasoning over structured knowledge but require additional epistemic traceability to be extensively employed. The code associated to this work is available at: <https://emanuelemusumeci.github.io/RoboData/>.

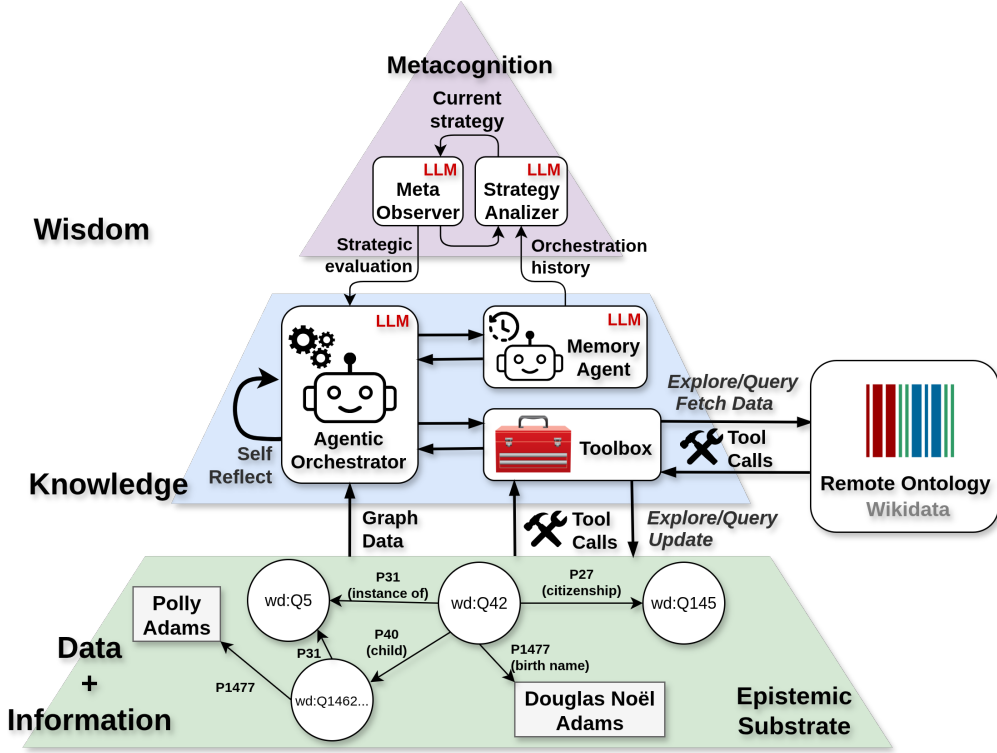


Figure 1: The KGQA agentic framework follows the DIKW architecture. The "Data + Information" layer is implemented as a knowledge graph. The "Knowledge" layer consists of a LLM-based orchestrator, using tools to inspect local and remote data. The "wisdom" layer provides feedback on the QA strategy.

2. Related Work

KG construction and enrichment. LLMs have the ability to replace the entire DIKW pyramid in KGQA tasks. For instance, LLMStore [13] treats a pre-trained LLM as a factual source by synthesizing Wikidata-like statements on-the-fly. Other methods extract facts from text via LLM-based pipelines. EDC+R uses a semantic extractor, a LLM and a trained *Schema Retriever* to extract triples from a document and map them to existing Schemas, producing structured and contextualized knowledge from an input document [14]. GraphAide [15] merges relations extracted from LLMs and documents, into an explicit KG, enabling hybrid query answering. In CooperKGC [16], specialized agents concurrently extract entities, relations and events, to build a KG, using ontologies to verify claims. In [17], relevant relations are selected from the Wikidata ontology to build a local KG, mirroring real-time updates remote or local information updates. The LLM answer is chosen if the KG can not generate one. The data selection method, though, is case-specific. RoboData adaptively queries Wikidata, integrating remote data, rather than interrogating a pre-built static KG, engaging in active graph construction. Additionally, RoboData self-reflects on the best course of action, correcting its data collection strategy.

Question Answering over KGs. LLMs may directly answer single-hop questions but struggle with multi-hop reasoning [18]. Better performance can be obtained by eliciting topological

reasoning: the QA task is decomposed in a sequence of thoughts, sequential like in Chain-of-Thoughts [9], explored like a tree as in Tree-of-Thoughts [10] or like a graph as in Graph-of-Thoughts [11]. Similar work injects information from KGs into LLMs, used as commonsense reasoners, providing an implicit goal-driven heuristic for knowledge integration and extraction for QA tasks. The main limitation is the LLM context window length: long contexts, in fact, may increase hallucinations [19]. Using the best KG encoding format for LLMs is crucial when they act both as a goal reasoner and as an answer generator. For instance, the GRAG architecture [20] retrieves relevant *textual subgraphs* from a pre-existing Knowledge Graph, using a divide-and-conquer approach, and feeds them in both a linearized textual format and in a structured topological view. KAG (Knowledge-Augmented Generation) [6] goes further, with a modular hybrid framework, comprising an entity selector, a semantic graph constructor and a multi-level contextual aligner, to extract semantically relevant subgraphs and align their representation with LLM-generated embeddings, for QA tasks where accuracy is critical. MindMap [21] produces "evidence subgraphs" from a KG: an LLM extracts entities from a query and then merges them into a "reasoning subgraph", enriched with commonsense knowledge, finally producing an answer from this enriched graph. These approaches show that injecting structured knowledge in a prompt can anchor LLM output to a knowledge base and reduce hallucinations. RoboData instead acquires knowledge procedurally constructing a knowledge base dynamically as part of a self-orchestrated reasoning process. In this way, the answer to the query is both a natural language answer and a *supporting* knowledge graph.

Agentic KG Reasoning. Similarly, in Think-on-Graph [22] a LLM is initially used to search for candidate KG triples that may be relevant to answering the query, then the LLM iteratively evaluates the most promising reasoning paths, by performing beam-search. Both these works allow traceable and explainable multi-hop query answering by performing iterative graph exploration, growing a local KG as a result. RoboData adds metacognitive reflection: it does not stop at finding an answer but checks answer plausibility and may loop back if the reasoning is unsupported by the underlying data. Other systems use a "toolkit" for KG queries to collect contextual information. SPINACH [23] mimicks a human writing a SPARQL query to answer a question, by exploring Wikidata through tool-calling to progressively refine the query. The collected knowledge, though, is not structured as a Knowledge Graph. Generate-on-Graph (GoG) [24] tackles QA over an incomplete KG, by generating missing triples via a Think-Search-Generate loop. Similarly, ODA [25] frames QA as an iterative cycle of Observe-Act-Reflect, while keeping the subgraph pruned to avoid exponential growth. Both GoG and ODA create a supporting KG but they rely on a well-structured workflow and their self-reflection power is limited to the control of subsequent iterations. KG-Agent [26] uses a small instruction-tuned LLaMA model with a toolbox of KG operations to generate a KG Reasoning program, featuring tool calls that extract knowledge from a KG, collecting the results in a Knowledge Memory, finally used to produce a response. Finally, AGENTiGraph [27] builds a multi-step pipeline, very similar to RoboData. A Task Planning step initially decomposes the original query into a series of discrete tasks, which are then delegated to specific agents (e.g. Semantic Parsing, KG Interaction, Reasoning Agent, etc.). Similarly to SPINACH, it plans the steps required to acquire enough context to build Cypher queries to query a local database. While AGENTiGraph operates on a pre-existing domain KG, RoboData collects data from remote ontologies like Wikidata. More in general, RoboData ranks among these Agentic systems for KGQA but emphasizes

robust orchestration and data provenance. It builds on the insight that LLMs can drive KG exploration agentically, but it uniquely centers on verifiability: every intermediate assumption in the query answering process is grounded in an external ontology.

If we consider KGQA a long-term task, *metacognition* might play an important role. In [28], answers produced for a Multi-Choice QA dataset with LLM models are evaluated after applying self-reflection, addressing potential answer optimization criteria, showing a performance improvement. In [29], agents accumulate observations in a long-term memory, populated by periodically running an introspective evaluation in relation to the assigned objectives. By influencing subsequent decisions, strategic adaptation capabilities emerge in long-term tasks. Here, the metacognitive functionality is clearly distinguished as a typical component of "System 1", the rational "slow" component of any cognitive system, in comparison with "System 2", the fast reactive one [30]. Only few KGQA systems exhibit some form of self-reflection or metacognitive behavior: ODA [25] incorporates an explicit "reflect" phase to refine its strategy after observations, GoG [24] engages in iterative assess-and-generate cycles when the KG is insufficient. SPINACH [23] refines the SPARQL query in a multi-turn process guided by past outcomes. RoboData incarnates this kind of strategic self-reflection, adding metacognitive self-evaluation, in order to optimize the effectiveness of its execution strategy.

3. Methodology

The system is designed to self-orchestrate, exploring the remote Wikidata ontology or the local KG, or to update it, progressively constructing the local graph, enabling explainable query resolution. Through self-reflection and metacognition, the agent is capable of correcting its own strategy, by reasoning on the past execution. As shown in Figure 1, the framework is organized in layers, following the DIKW paradigm.

Epistemic Substrate. Acting as the *Data+Information* layer in the *DIKW* taxonomy, the accumulated local graph, initially empty, is progressively populated during the Query Answering. The local KG represents data using a common schema, to potentially accomodate multiple remote knowledge sources. The local KG can be accessed through exploration tools, callable by the agent, returning data in a readable text format, similarly to GRAG [20].

Agentic Layer. Acts as the main operational component, representing the *Knowledge* layer in the *DIKW* paradigm. This layer comprises several modules. The main one is the **Agentic Orchestrator**, modeled as a Finite-State Machine (FSM), shown in Figure 2, managing how the Agent interacts with the LLM. Each state has a specific prompt and set of tools that the agent can use. *Exploration states* can inspect the remote ontology and the local data while the *Update state* fetches remote data and integrates it locally. Prompts are divided in a *System Prompt*, containing task instructions for the specific state, an *Assistant Prompt*, containing various types of working memory and a snapshot of the current KG in a readable format, and a *User Prompt*, containing the original query. An **Agentic Memory** contains a trace of the previous tool calls and state transitions, providing an overview of the past execution. To prevent prompts from becoming too long, the memory actively summarizes old entries using a LLM; in case the previous state fetched data from a remote knowledge source, this information is added to the prompt; a *metacognitive observation*, a high-level evaluation about the past execution trace,

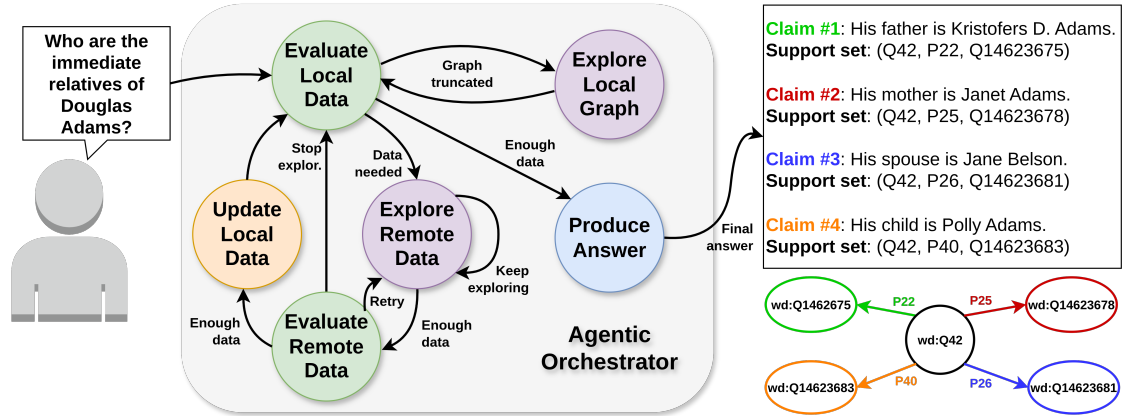


Figure 2: Initially, the query is evaluated in *Local Data Evaluation* and the KG is empty: the orchestrator transitions to *Remote Data Evaluation*. Here, the proper tool calls are executed. Remote data is evaluated in *Remote Data Evaluation*. Then the self-orchestration determines the remaining steps.

produced by the *Metacognitive Layer* before every *evaluation state* in the orchestrator. Finally, the LLM is informed about the possible successor states and is instructed to specify the next state. Depending on the received answer, tool calls are run, the memory is updated, and the transition is performed. At specific states, the LLM can decide whether the agent is ready to produce an answer. In this case, the LLM is explicitly instructed to return a response, where each sentence must be associated to a "*supporting set*" of nodes and edges of the local KG, acting as a grounding set for the answer. In the orchestrator, the agent dynamically determines the order in which states are visited and tools are called and performs self-reflection, by observing previous successes or failures or evaluating data completeness. Workflows are replaced with a dynamic transition system, as the state sequence is therefore determined by the commonsense reasoning capabilities of the LLM. The orchestrator is allowed to run a predefined number of turns, after which it is forced to produce an answer. The orchestration process, represented in Figure 2, is articulated in the following stages:

Local Data Evaluation: Evaluates whether the local knowledge graph contains enough information to answer the query. If the graph is empty, the system proceeds directly to the *Remote Exploration* state. If there is enough data to answer the original query, the orchestrator visits the *Produce Answer* state. Finally, if the local KG contents exceed the maximum allowed token count, the *Local Graph Exploration* is visited.

System Prompt: Evaluate if the local graph data is enough to answer the query. [Exploration strategy suggestions]. If you feel that key relationships are missing from the local graph, you can suggest using remote tools to gather more data and go back to remote exploration. [List of possible transitions]. [Invitation to keep KG topologically consistent]. [Invitation to self-reflection].

Assistant Prompt: Self-reflection, memory context, local graph data, local exploration results, and optional metacognitive strategy.

Local Graph Exploration: The agent uses tools to explore the local KG, gathering additional information relevant to the query but not initially visible in the truncated graph. The agent can

choose to transition again to the *Local Data Evaluation* state or to keep exploring.

System Prompt: Explore the local knowledge graph to answer the query. Use available tools to find relevant information. Avoid redundancy. [List of possible transitions]. [Invitation to self-reflection]

Assistant Prompt: Self-reflection, memory context, local graph data, and local exploration results.

Tools: get_node (retrieve node details by ID), get_edge (retrieve edge details by ID), cypher_query (execute Cypher queries on graph database)

Remote Data Exploration: The agent can invoke tools to collect data from the remote ontology. It can then decide to transition to the *Local Data Evaluation* state or to keep exploring.

System Prompt: The local graph is insufficient. Use remote tools to gather relevant data that fills gaps. Focus on entities/relationships related to the query. [List of possible transitions]

Assistant Prompt: Self-reflection, memory context, and local graph data.

Tools: get_entity_info (get Wikidata entity information), get_property_info (get Wikidata property information), search_entities (search Wikidata entities by text), sparql_query (execute SPARQL queries on Wikidata), explore_entity_neighbors (explore entity relationships), build_local_graph (build local graph around entity)

Remote Data Evaluation: The relevance of remote retrieved information to the original query is evaluated. If the data is considered relevant, the system transitions *Local Graph Update* state to integrate it into the local KG. Otherwise, the system can decide whether to keep exploring or go back to the *Local Data Evaluation* state.

System Prompt: Evaluate if remote data is relevant for building a knowledge graph for the query. [List of possible transitions]. [Invitation to self-reflection].

Assistant Prompt: Previous thoughts, local graph data, remote graph data, memory context, and optional metacognitive strategy.

Tools: get_node (retrieve node details by ID), get_edge (retrieve edge details by ID), cypher_query (execute Cypher queries on graph database)

Local Graph Update: The selected remote entities and relations related to the query are fetched into the local graph, then the agent transitions back to the *Local Data Evaluation* state.

System Prompt: Update the local graph with relevant remote data. Only add entities/relationships directly related to the query. Avoid isolated nodes and redundancy. [Update strategy suggestions]. [Invitation to keep KG topologically consistent].

Assistant Prompt: Self-reflection, local graph data, remote graph data, and memory context.

Tools: fetch_node (add Wikidata entity to local graph), fetch_relationship_[to|from]_node (add all statements for a property from/to a specific node to local graph)

Answer Production: Final state, where the agent generates an answer. Each sentence is associated to a set of entities and relations in the local data, acting as proof for that claim.

System Prompt: You are an expert agent that produces comprehensive answers using data from a local knowledge graph. Your task is to create a final answer where each sentence is backed by specific evidence from the local graph. [Detailed format and consistency constraints]. [Format example].

Assistant Prompt: Self-reflection, local graph data, and memory context.

Metacognition. FSM-based orchestration balances freedom of agentic action with guidance over reasoning paths, through. Hallucinations induced by excessive context lengths, due to the need to provide the KG to the LLM in data-related prompts, increase with complex queries requiring long answers and big underlying knowledge graphs. As a mitigation, a corrective strategy is generated by the metacognition module and embedded in subsequent Orchestrator prompts. The metacognition happens in evaluation states at two distinct levels: the first one is performed inline in the orchestrator. A **self-reflection** behavior in the LLM is elicited, by instructing the Agent to reflect on the past execution trace to determine corrective advices for the next selected state in the orchestrator, with the desired outcome of correcting the short-term strategy in reaction to failures or stagnation in the query answering process. Then, each time the agent reaches an *evaluation state*, in the "Wisdom" layer, explicit **metacognition** is performed to provide a higher-order strategic evaluation of the overall agentic strategy, by analyzing the sequence of past states, tool calls and graph updates. If a previous metacognitive observation is available, the inferred strategy is compared against it. The module finally produces a *metacognitive observation*, a corrective plan to optimize suboptimal tendencies in the agent strategy. This observation is then injected into the prompts of the orchestrator evaluation states. As turns run out, a variable "**turn urgency**" message (depending on how many orchestrator turn are left), urges the LLM to consolidate the KG, avoiding isolated nodes and unconnected components, leading to a better consistency in output KGs. Metacognition components:

1. Strategic Assessment. Infer the current strategy from statistics and memory.

System Prompt: Analyze the sequence of actions and identify the underlying strategic approach. Focus on: high-level strategy, tool usage preferences, data retrieval patterns, knowledge graph construction strategy, and performance indicators. **Turn urgency.**

Assistant Prompt: Action sequence, execution statistics (state and graph statistics, tool usage), available tools, memory context, local graph data, local/remote exploration results.

2. Meta-Observation. The strategy is compared to previous ones, and corrective feedback is generated, enabling the agent to navigate complex queries and reorient sub-optimal behavior.

System Prompt: Evaluate current strategy against task outcome and previous observations. Consider: strategy coherence, performance assessment, strategic continuity, opportunity identification, and tool optimization. **Turn urgency.**

Assistant Prompt: Detected strategy, previous metacognition, task outcome assessment, available tools, memory context, local graph data, local/remote exploration results.

Table 1

Batch A: Results for the basic KGQA functionality experiments.

OK Query A1: Who is the author of 'The Hitchhiker's Guide to the Galaxy'?	
Answer: The 1979 novel 'The Hitchhiker's Guide to the Galaxy' is authored by Douglas Adams. (1) The series 'The Hitchhiker's Guide to the Galaxy,' which spans from 1979 to 1992, is also authored by Douglas Adams. (2) Additionally, Eoin Colfer is an author of the series 'The Hitchhiker's Guide to the Galaxy.' (3)	Support Sets: (1) <The Hitchhiker's Guide to the Galaxy (Q3107329), author (P50), Douglas Adams (Q42)>, (2) <The Hitchhiker's Guide to the Galaxy (Q25169), author (P50), Douglas Adams (Q42)>, (3) <The Hitchhiker's Guide to the Galaxy (Q25169), author (P50), Eoin Colfer (Q211893)>
Iterations: 6 Tools: 4 Tokens: 15043 Time (s): 12.54 Nodes: 4 Edges: 3	
OK Query A2: Who are the relatives of Douglas Adams?	
Answer: Douglas Adams' father is Kristofers Duglass Adamss. (1) His mother is Janet Adams. (2) Douglas Adams was married to Jane Belson. (3) He has a daughter named Polly Adams. (4)	Support Sets: (1) <Douglas Adams (Q42), father (P22), Kristofers Duglass Adamss (Q14623675)>, (2) <Q42, mother (P25), Janet Adams (Q14623678)>, (3) <Q42, spouse (P26), Jane Belson (Q14623681)>, (4) <Q42, child (P40), Polly Adams (Q14623683)>
Iterations: 8 Tools: 7 Tokens: 17652 Time (s): 14.88 Nodes: 5 Edges: 4	
INCOMPLETE Query A3: Who are the relatives of Douglas Adams and what books did he write?	
Answer: Polly Adams is a relative of Douglas Adams, specifically his child. (1) Douglas Adams wrote "The Hitchhiker's Guide to the Galaxy." (2) Douglas Adams is also the author of the "Dirk Gently series." (3) Additionally, Douglas Adams co-authored "The Private Life of Genghis Khan." (4)	Support Sets: (1) <Douglas Adams (Q42), child (P40), Polly Adams (Q14623683)>, (2) <Q42, notable work (P800), The Hitchhiker's Guide to the Galaxy (Q25169)>, (3) <Q42, P800, Dirk Gently series (Q20736364)>, (4) <Q42, P800, The Private Life of Genghis Khan (Q7758404)>
Iterations: 6 Tools: 5 Tokens: 14166 Time (s): 14.06 Nodes: 5 Edges: 4	
OK Query A4: Among the founders of tencent company, who has been member of national people's congress?	
Answer: Ma Huateng is a founder of Tencent. (1) Ma Huateng has held the position of National People's Congress deputy. (2)	Support Sets: (1) <Tencent (Q860580), founded by (P112), Ma Huateng (Q1739008)>, (2) <Q1739008, position held (P39), National People's Congress deputy (Q10891456)>
Iterations: 28 Tools: 32 Tokens: 101825 Time (s): 84.02 Nodes: 11 Edges: 9	
INCOMPLETE Query A5: In which federal state is the Veltins brewery headquarter?	
Answer: The Veltins Brewery has its headquarters in Meschede. (1) Meschede is a town located in the Hochsauerland district. (2) The Hochsauerland district is in North Rhine-Westphalia, Germany. (3)	Support Sets: (1) <Veltins Brewery (Q608350), headquarters location (P159), Meschede (Q5632)>, (2) <Meschede (Q5632), (3) Q5632
Iterations: 6 Tools: 4 Tokens: 10303 Time (s): 11.36 Nodes: 2 Edges: 1	
INCOMPLETE Query A6: Which philosophers influenced by Kant were also mathematicians?	
Answer: David Hilbert was a philosopher influenced by Immanuel Kant and was also a mathematician. (1) Hermann Weyl was influenced by Immanuel Kant and is described as a German mathematician. (2) Luitzen Egbertus Jan Brouwer was influenced by Immanuel Kant and is identified as a Dutch mathematician and logician. (3) (...)	Support Sets: (1) <David Hilbert (Q41585), influenced by (P737), Immanuel Kant (Q9312)>, Q41585, (2) <Hermann Weyl (Q71029), P737, Q9312>, Q71029, (3) <Luitzen Egbertus Jan Brouwer (Q155887), P737, Q9312>, Q155887, (4) <Shem-Tov Gefen (Q6992366), P737, Q9312>, Q6992366
Iterations: 22 Tools: 29 Tokens: 124613 Time (s): 70.51 Nodes: 32 Edges: 33	
WRONG Query A7: Which Italian and Japanese cities are twinned?	
Answer: The local knowledge graph does not show any city that is twinned with both Italian and Japanese cities. (1)	Support Sets: (1) P190 (P190)
Iterations: 28 Tools: 148 Tokens: 643275 Time (s): 148.49 Nodes: 480 Edges: 623	

4. Experiments

This section evaluates the proposed system across varying query complexity and ambiguity. By default, metacognition is disabled while self-reflection is enabled, as it is an established practice in the current state of the art. Experiments are conducted using GPT-4o as a Reasoning Model. The orchestrator is allowed to run for a limited number of turns, after which it is forced to produce an answer with the collected data. The only remote ontology used in these

experiments is Wikidata, accessed through the official API. To embed Knowledge Graphs in prompts, we follow an approach similar to [20], by first listing nodes and relationship types in the graph and then listing all the triples (relationship instances), instead of using a knowledge graph dump. To keep the context as short as possible, the agentic memory is summarized after 30 entries are added, halving its length. In the first batch of queries, **Batch A** (A1-A7), we test RoboData on a curated set of single and multi-hop queries, at an increasing level of complexity and scope, to assess its basic KGQA capabilities. The results are reported in Table 1, containing for each experiment, the original query, the answer with its supporting set, and several statistics, including number of iterations and tool calls, exchanged tokens and execution time, number of nodes and edges added. In this batch, we allow the orchestrator to run for at most 30 turns. Queries A1-A4 assess the Question Answering capability of this system at an increasing level of complexity. In particular, A1-A3 require a single hop to answer correctly (basically a local exploration around the entity Q42 in Wikidata as all the required elements for the answer are located one relation away from the main subject of the question). A2 and A3 though require multiple relations to answer and A2 is contained in A3. The result shows consistency in the intersection of the answers to A2 and A3. The answer to A3 is only partial: the part of the answer in common with A2 is only partially found; therefore, the answer is considered wrong. Query A4 instead requires multiple hops to answer correctly, showing longer-term task execution capabilities. Queries A5-A7 are more complex both in the number of hops required and in scope. A5 requires a chain of 4 hops to answer but only 3 are found: this is due to the fact that all entities along the chain contain part of the answer already in the textual description of the Wikidata entity (in our case, the name of the german federal district containing the Veltins brewery), tricking the LLM into assuming that the answer is fully supported by the node. In A6, the answer is incomplete. Both queries A5 and A6 highlight the main limitation of this system, when operating without a metacognitive strategy observer: the orchestrator alone tends to lose track of tasks involving multiple hops or big sets of entities and properties. This is confirmed by query A7, showing how the limited context window negatively affects tasks where more working memory is required to keep track of longer-term tasks over big sets of entities (in this case, the big sets of cities to compare to find twin cities). In this query, in fact, even if the full 30 iterations are reached and 148 tool calls are performed, collecting 480 nodes and 623 edges, the limitations due to the working memory and context window prevent the agent from returning an answer. With **Batch B** (B1-B3), reported in Table 2, we explore the behavior of RoboData in tasks requiring more structural knowledge about the ontology: we test three queries, not consisting of purely factual questions but more operational ones, designed to trigger different exploration patterns. Query B1 tests the capability of performing the simple exploratory operation of finding the subclasses of the immediate neighbors of the Wikidata entity Q42, successfully exploring the neighborhood. Queries B2 and B3 instead test the system’s ability to navigate Wikidata’s upper ontology and explain its taxonomic structure, by asking to fetch the Wikidata upper ontology. In particular, B2 asks to find its root (without knowing that the root is "entity" (Q35120)) and to fetch its immediate subclasses (referring to the "subclass of" property (P279)) and their subclasses, while B3 adds the "instance of" property (P31) to the request. Given the strictly operational nature of these request, both queries B2 and B3 are executed correctly, which is probably due to the operational reasoning of the instruction tuned GPT-4o model. Still, the model struggles over the more complex task in B3: this is

Table 2

Batch B: Results for the batch of operational experiments.

OK Query B1: Find all subclasses of the neighbors of Q42	
Answer: The occupation "playwright" (Q214917) is a subclass of "writer" (Q36180). (1) The occupation "screenwriter" (Q28389) is a subclass of "writer" (Q36180). (2) The occupation "novelist" (Q6625963) is a subclass of "writer" (Q36180). (...)	Support Sets: (1) playwright (Q214917) -> subclass of (P279) -> writer (Q36180), (2) screenwriter (Q28389) -> P279 -> Q36180, (3) novelist (Q6625963) -> P279 -> Q36180 (...)
Iterations: 5 Tools: 21 Tokens: 13123 Time (s): 26.82 Nodes: 20 Edges: 16	
INCOMPLETE Query B2: Explore the ontology: Start from the root and list all direct subclasses. Then for each subclass, list its direct subclasses	
Answer: The root entity in the ontology is "Q35120 (entity)." (1) Q35120 (entity) is an instance of Q23958852 (variable-order class). (2) Q35120 (entity) is also an instance of Q124711104 (BFO class). (3) Q23958852 (variable-order class) is an instance of itself. (...)	Support Sets: (1) entity (Q35120), (2) <entity (Q35120), P31, Q23958852>, (3) <Q35120, P31, Q124711104>, (4) <Q23958852, P31, Q23958852>, (5) <Q23958852, subclass of (P279), Q16889133>, (6) <Q124711104, P31, Q19478619>, (7) <Q124711104, subclass of (P279), Q16889133>
Iterations: 10 Tools: 11 Tokens: 31380 Time (s): 48.5 Nodes: 5 Edges: 6	
WRONG Query B3: Explore the ontology: Start from the root and list all direct subclasses and instances. Then for each subclass, list its direct subclasses.	
Answer: The root entity in the ontology is identified as Q35120, which is described as anything that can be considered, discussed, or observed. (1) Q35120 is an instance of both Q124711104 (BFO class) and Q23958852 (variable-order class). (2) (...) (3) Several entities are instances of Q35120, including Q496651 (National Baptist Convention, USA), Q1647158 (V-Modell), and Q3306469 (Prairie View State College).(...)	Support Sets: (1) entity (Q35120), (2) <entity (Q35120), instance of (P31), BFO class (Q124711104)>, <Q35120, instance of (P31), variable-order class (Q23958852)>, (3) <Q35120, subclass of (P279), no_value (Q35120_P279_931456939447813722)>(...)
Iterations: 30 Tools: 55 Tokens: 165987 Time (s): 132.57 Nodes: 62 Edges: 30	

mainly due to noisy and incorrect usage of the "instance of" and "subclass of" relationships with "entity": as an example, the answer of B3 refers to the "National Baptist Convention, USA" entity, represented as an "instance of" "entity", which is formally true, as all Wikidata entities are an instance of "entity" or one of its subclasses, but technically wrong, as it introduces noise in the upper ontology. Finally, **Batch C**, in Table 3, repeats several failed or incomplete queries from batch A and B, with the metacognitive layer enabled, allowing execution for 50 turns instead of 30. In general, metacognition brought an improved strategic guidance over longer-term tasks, improving performance with queries requiring to explore a bigger set of entities. This time, Query A5 is correctly answered, completing the required 4 hops to support the answer that in Batch A was only semantically available in the entity descriptions but not formally supported by the KG. Also, Query A6 is answered completely (to our knowledge), listing the missing philosopher and mathematician influenced by Kant (Q155887), in even fewer iterations than in Batch A. The operational task B2 shows an improvement: reaching the whole allowed 50 turns, it manages to find most of the subclasses of "entity" (but not all of them). Instead, answer quality even drops for query B3, where the agent collected only 4 nodes and 3 edges, against the 62/30 collected without metacognition. Finally, answer quality improves for query A7, showing a better management of the limited context window and of the long-term task. Results over queries A5-A7 support the conclusion that agentic metacognition improves performance over broader and longer KGQA and KG construction tasks, normally hindered by the limited working memory, by summarizing and evaluating previous strategies with respect to the original task, and adjusting the agentic strategy accordingly.

Table 3

Batch C: Metacognition experiments over selected failed or incomplete queries of Batch A and B.

OK Query A5: In which federal state is the Veltins brewery headquarter?	
Answer: [CONSISTENT WITH QUERY A5] (...) (2) Hochsauerlandkreis is part of the Arnsberg Government Region. (3) The Arnsberg Government Region is located in the federal state of North Rhine-Westphalia. (4)	Support Sets: [CONSISTENT WITH QUERY A5] (...) (2) <Q5632, located in the administrative territorial entity (P131), Hochsauerlandkreis (Q5946)>, (3) <Q5946, (P131), Arnsberg Government Region (Q7924)>, (4) <Q7924, (P131), North Rhine-Westphalia (Q1198)>
Iterations: 46 Tools: 34 Tokens: 180733 Time (s): 122.28 Nodes: 7 Edges: 5	
OK Query A6: Which philosophers influenced by Kant were also mathematicians?	
Answer: [CONSISTENT WITH QUERY A6] (...) (3) Luitzen Egbertus Jan Brouwer was influenced by Immanuel Kant and was also a mathematician. (...)	Support Sets: [CONSISTENT WITH QUERY A6] (...) (4) <Luitzen Egbertus Jan Brouwer (Q155887), P737, Q9312>, <Q155887, P106, Q170790>, (...)
Iterations: 14 Tools: 19 Tokens: 68844 Time (s): 38.5 Nodes: 31 Edges: 37	
INCOMPLETE Query B2: Explore the ontology: Start from the root and list all direct subclasses. Then for each subclass, list its direct subclasses	
Answer: The root entity in the ontology is "entity" (Q35120). (1) The direct subclasses of "entity" (Q35120) include "substance" (Q378078), "object" (Q488383), "continuant" (Q103940464), "location" (Q115095765), "unidentified entity" (Q120725535), "conceptual entity" (Q121770302), "part" (Q13196193), "abstract entity" (Q7048977), and "collective entity" (Q99527517). (2) (...)	Support Sets: (1) entity (Q35120), (2) substance (Q378078), object (Q488383), continuant (Q103940464), location (Q115095765), unidentified entity (Q120725535), conceptual entity (Q121770302), part (Q13196193), abstract entity (Q7048977), collective entity (Q99527517),(...)
Iterations: 50 Tools: 90 Tokens: 251921 Time (s): 179.52 Nodes: 11 Edges: 10	
INCOMPLETE Query B3: Explore the ontology: Start from the root and list all direct subclasses and instances. Then for each subclass, list its direct subclasses.	
Answer: The root entity "entity" (Q35120) is an instance of the "BFO class" (Q124711104). (1) The root entity (Q35120) is also an instance of the "variable-order class" (Q23958852). (...)	Support Sets: (1) <entity (Q35120), instance of (P31), BFO class (Q124711104)>, (2) <Q35120, instance of (P31), variable-order class (Q23958852)>, (...)
Iterations: 50 Tools: 45 Tokens: 206342 Time (s): 134.64 Nodes: 4 Edges: 3	
INCOMPLETE Query A7: Which Italian cities are twinned with Japanese cities?	
Answer: Rome is twinned with Tokyo. (1) Turin is twinned with Nagoya. (2) Naples is twinned with Kagoshima. (3) Terni is twinned with Kobe. (...)	Support Sets: (1) <Rome (Q220), twinned administrative body (P190), Tokyo (Q1490)>, (2) <Turin (Q495), (P190), Nagoya (Q11751)>, (3) <Naples (Q2634), (P190), Kagoshima (Q15674)>(...)
Iterations: 18 Tools: 68 Tokens: 103346 Time (s): 62.94 Nodes: 216 Edges: 210	

5. Conclusions and Future Work

This work introduced RoboData, an Agentic System for LLM-based trustable KGBA, that simultaneously answers a natural language query and build a KG from a remote ontology, to support the generated claims with facts. RoboData retrieves and organizes relevant information from remote ontologies but also adapts its exploration strategy to overcome the limitations of LLM context windows and long-term memory. Experiments show that even when the system struggles with longer-term tasks requiring broader exploration of remote data, our approach mitigates them by leveraging the LLM’s metacognitive capabilities to plan corrective long-term strategies. The explicit metacognitive layer improved performance on complex, multi-hop queries and large-scale ontology exploration, yielding more complete and traceable answers. RoboData has promising potential applications in Robotics, allowing the unsupervised creation of semantic scene graphs to assist in goal-oriented reasoning and planning.

References

- [1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, D. Amodei, Scaling laws for neural language models, arXiv preprint arXiv:2001.08361 (2020).
- [2] L. Chen, J. Q. Davis, B. Hanin, P. Bailis, I. Stoica, M. A. Zaharia, J. Y. Zou, Are more llm calls all you need? towards the scaling properties of compound ai systems, *Advances in Neural Information Processing Systems* 37 (2024) 45767–45790.
- [3] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, Q. V. Le, Finetuned language models are zero-shot learners, arXiv preprint arXiv:2109.01652 (2021).
- [4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, *Advances in neural information processing systems* 33 (2020) 1877–1901.
- [5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al., Retrieval-augmented generation for knowledge-intensive nlp tasks, *Advances in neural information processing systems* 33 (2020) 9459–9474.
- [6] L. Liang, Z. Bo, Z. Gui, Z. Zhu, L. Zhong, P. Zhao, M. Sun, Z. Zhang, J. Zhou, W. Chen, et al., Kag: Boosting llms in professional domains via knowledge augmented generation, in: *Companion Proceedings of the ACM on Web Conference 2025*, 2025, pp. 334–343.
- [7] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Al-tenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774 (2023).
- [8] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: *International Conference on Learning Representations (ICLR)*, 2023.
- [9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, *Advances in neural information processing systems* 35 (2022) 24824–24837.
- [10] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, *Advances in neural information processing systems* 36 (2023) 11809–11822.
- [11] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, et al., Graph of thoughts: Solving elaborate problems with large language models, in: *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 2024, pp. 17682–17690.
- [12] J. Rowley, The wisdom hierarchy: representations of the dikw hierarchy, *Journal of information science* 33 (2007) 163–180.
- [13] M. Machado, J. M. Rodrigues, G. Lima, S. R. Fiorini, V. T. da Silva, Llm store: Leveraging large language models as sources of wikidata-structured knowledge, in: *International Semantic Web Conference*, 2024.
- [14] B. Zhang, H. Soh, Extract, define, canonicalize: An llm-based framework for knowledge graph construction, arXiv preprint arXiv:2404.03868 (2024).
- [15] S. Purohit, G. Chin, P. S. Mackey, J. A. Cottam, Graphaide: Advanced graph-assisted query and reasoning system, in: *2024 IEEE International Conference on Big Data (BigData)*,

IEEE, 2024, pp. 3485–3493.

- [16] H. Ye, H. Gui, A. Zhang, T. Liu, W. Jia, Beyond isolation: Multi-agent synergy for improving knowledge graph construction, in: China Conference on Knowledge Graph and Semantic Computing, Springer, 2024, pp. 69–81.
- [17] R. Chen, W. Jiang, C. Qin, I. S. Rawal, C. Tan, D. Choi, B. Xiong, B. Ai, Llm-based multi-hop question answering with knowledge graph integration in evolving environments, arXiv preprint arXiv:2408.15903 (2024).
- [18] E. Biran, D. Gottesman, S. Yang, M. Geva, A. Globerson, Hopping too late: Exploring the limitations of large language models on multi-hop queries, arXiv preprint arXiv:2406.12775 (2024).
- [19] S. Liu, K. Halder, Z. Qi, W. Xiao, N. Pappas, P. M. Htut, N. A. John, Y. Benajiba, D. Roth, Towards long context hallucination detection, arXiv preprint arXiv:2504.19457 (2025).
- [20] Y. Hu, Z. Lei, Z. Zhang, B. Pan, C. Ling, L. Zhao, Grag: Graph retrieval-augmented generation. 2024, URL <https://arxiv.org/abs/2405.16506> (2024).
- [21] Y. Wen, Z. Wang, J. Sun, Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models, arXiv preprint arXiv:2308.09729 (2023).
- [22] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H.-Y. Shum, J. Guo, Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph, arXiv preprint arXiv:2307.07697 (2023).
- [23] S. Liu, S. J. Semnani, H. Friedman, J. Xu, I. D. Zhao, M. S. Lam, Spinach: Sparql-based information navigation for challenging real-world questions, arXiv preprint arXiv:2407.11417 (2024).
- [24] Y. Xu, S. He, J. Chen, Z. Wang, Y. Song, H. Tong, G. Liu, K. Liu, J. Zhao, Generate-on-graph: Treat llm as both agent and kg in incomplete knowledge graph question answering, arXiv preprint arXiv:2404.14741 (2024).
- [25] L. Sun, Z. Tao, Y. Li, H. Arakawa, Oda: Observation-driven agent for integrating llms and knowledge graphs, arXiv preprint arXiv:2404.07677 (2024).
- [26] 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).
- [27] X. Zhao, M. Blum, R. Yang, B. Yang, L. M. Carpinero, M. Pina-Navarro, T. Wang, X. Li, H. Li, Y. Fu, et al., Agentigraph: An interactive knowledge graph platform for llm-based chatbots utilizing private data, arXiv preprint arXiv:2410.11531 (2024).
- [28] M. Renze, E. Guven, Self-reflection in llm agents: Effects on problem-solving performance, arXiv preprint arXiv:2405.06682 (2024).
- [29] J. Toy, J. MacAdam, P. Tabor, Metacognition is all you need? using introspection in generative agents to improve goal-directed behavior, arXiv preprint arXiv:2401.10910 (2024).
- [30] G. Booch, F. Fabiano, L. Horesh, K. Kate, J. Lenchner, N. Linck, A. Loreggia, K. Murgesan, N. Mattei, F. Rossi, et al., Thinking fast and slow in ai, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, 2021, pp. 15042–15046.