

AMA-KBQA: An Adaptable Multi-Agent Framework for Generalizable Knowledge Base Question Answering

Yannic Duan¹, Maximilian Klattig¹, Maurice Bardenhagen¹, Yuyang Li¹ and Tobias Käfer¹

¹Karlsruhe Institute of Technology (KIT), Institute of Applied Informatics and Formal Description Methods (AIFB), Karlsruhe, Germany

Abstract

We present **AMA-KBQA**, a multi-agent framework for retrieval-augmented generation (RAG) over knowledge graphs that decouples reasoning by large language models (LLMs) from graph retrieval through abstract agents, configuration-driven adapters, and Model Context Protocol (MCP) tool servers. The system requires *no model training*: onboarding a new knowledge graph reduces to writing SPARQL wrappers behind an abstract tool interface, and adopting a stronger LLM is a configuration change. A journal-based scratchpad keeps reasoning state token-efficient, and a Lessons-Learned loop distills high-quality traces into few-shot examples. Across KQAPro and SciQA, the same reasoning core generalises over a reified encyclopedic graph and a hierarchical scientific graph, reaching up to 86% accuracy.

Keywords

KBQA, SPARQL, Large Language Models, Model Context Protocol, Knowledge Graphs, RAG

Demo <https://purl.archive.org/ama-kbqa-demo> **Code+Video** <https://github.com/yannicd03/AMA-KBQA>

1. Introduction

Knowledge Base Question Answering (KBQA) is the task of answering natural-language questions over a structured knowledge base, typically a Resource Description Framework (RDF) graph queried via SPARQL, a query language for graph-based data. KBQA makes curated graph data accessible without requiring knowledge or construction of SPARQL queries; large language models (LLMs) have become the default instrument for this task. Yet LLMs alone suffer from knowledge cutoff, hallucination, and poor multi-hop reasoning over structured data [1]. Text-to-SPARQL semantic parsing [2] and retrieval-augmented generation (RAG) [3] ground the model in the graph, but bind the resulting system tightly to one graph’s schema and vocabulary, and state-of-the-art KBQA systems additionally depend on dataset-specific training, from fine-tuned query generators [4] to trained question classifiers [5]. Both forms of coupling are costly: porting to a new graph or adopting a stronger model means re-engineering or retraining the pipeline.

This rigidity is particularly painful because deployed knowledge graphs are not static. Industry-scale graphs evolve continuously as new domains are onboarded, schemas are refactored, and data sources are added, replaced, or deprecated [6]; rebuilding the pipeline for every such change is impractical. What is needed is a framework that treats both the knowledge graph (KG) and the language model as swappable components, adapted by configuration rather than re-implementation.

We introduce **AMA-KBQA**, a multi-agent framework built around exactly this requirement. It counters coupling to the graph with an abstract tool interface: the reasoning core only ever calls typed graph operations, which each graph implements as deterministic SPARQL wrappers that insert the tool’s parameters into a valid query template instead of letting the LLM write SPARQL freely. The backbone LLM itself is a configuration choice. And it requires no dataset-specific training: a question is answered by (i) structured *pre-processing* (question classification and entity extraction), (ii) *agentic exploration* via typed tool calls over SPARQL and vector search, and (iii) *grounded synthesis* over a scratchpad of verified facts rather than free LLM generation. On top of these three phases, a Lessons-Learned loop distills highly rated traces into few-shot examples, so the system improves over time without fine-tuning.

Posters, Demos, Blue Sky, and Tutorials at SEMANTICS 2026, Sep 2026, Ghent, Belgium

✉ yannic.duan@student.kit.edu (Y. Duan); maximilian.klattig@student.kit.edu (M. Klattig); maurice.bardenhagen@student.kit.edu (M. Bardenhagen); yuyang.li@kit.edu (Y. Li); tobias.kaefer@kit.edu (T. Käfer)



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

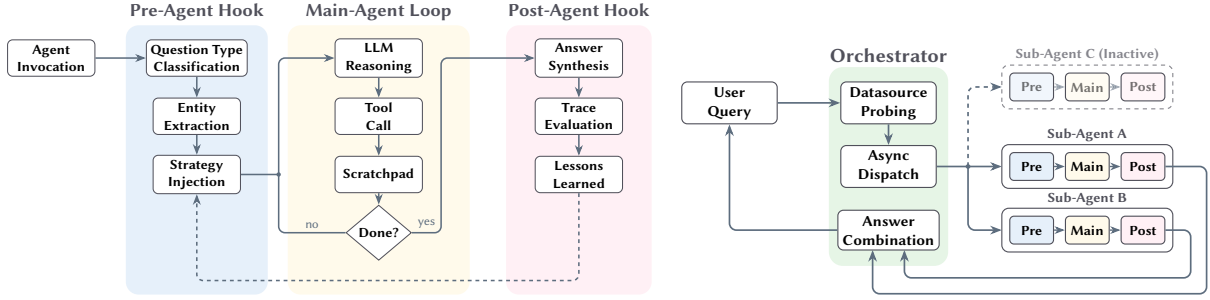


Figure 1: Left: agent lifecycle, with structured pre-processing, an iterative tool-calling main loop over a scratchpad, and post-hoc synthesis, trace evaluation, and lesson distillation. Right: the multi-agent orchestrator dispatches the question to the sub-agents matched by the routing probe (each an instance of the left lifecycle) and combines their answers; unmatched sub-agents stay inactive, and the current prototype commits to a single route per question.

2. Related Work

Text-to-SPARQL systems translate a question directly into a query: classical semantic parsers learn the mapping from question-answer pairs [2], neural approaches treat SPARQL as a target language for sequence-to-sequence models [4, 7], and recent work prompts LLMs to generate SPARQL directly, still with limited reliability [8]. All of these couple the system to one graph’s vocabulary and structure, so a schema change degrades the learned mapping. Interactive-KBQA [5] instead reframes KBQA as a multi-turn control loop over atomic graph APIs in the spirit of ReAct [9], paired with a trained BERT question classifier. We adopt this agentic stance and extend it for deployability: a configuration-driven adapter layer ports the same reasoning core to a new knowledge graph without rewrites or training, the typed tools are exposed via the Model Context Protocol (MCP) [10], an orchestrator federates questions across sub-agents, and a Lessons-Learned loop improves the system from its own traces, following Synapse’s trajectory-as-exemplar approach [11]. Each of these ingredients has precedent; our contribution lies in their composition: to our knowledge, no prior training-free system runs one unchanged reasoning core across structurally heterogeneous knowledge graphs and routes single questions across them.

3. Architecture

Federated multi-source design. AMA-KBQA centres on a shared reasoning core whose sub-agents are instances of the same template and differ only in their knowledge-graph adapter. All data access goes through typed MCP tools for entity discovery, graph navigation, and schema introspection, backed by a triple store and a vector store. Since each sub-agent is self-contained, an *orchestrator* can drive several of them (Fig. 1, right): it probes each data source’s vector store with the query’s top entities, commits to a route based on this evidence, dispatches the question to the matched sub-agent, and returns its grounded, source-attributed answer.

Agent lifecycle. Each sub-agent runs the same three-phase lifecycle (Fig. 1, left). A structured pre-agent hook classifies the question by type and extracts candidate entities, injecting a matching strategy document and type-specific few-shot examples into the system prompt. The main loop then iteratively calls typed tools and writes its findings to a structured scratchpad [12] of visited nodes, discovered values, and verified facts; deterministic comparison and verification tools offload numeric and boolean logic from the language model, and a loop detector injects recovery guidance when the agent stalls. Finally, a post-agent hook synthesises the answer using the scratchpad’s verified facts, and a judging agent scores the trace so that good runs become few-shot examples for future queries.

Abstract tool interface. The contract between the reasoning core and each graph is a two-level registry of atomic operations. A required tier of eleven operations every adapter must bind covers

Table 1

Evaluation excerpt: 500 questions sampled from the KQAPro validation split and the complete 100-question SciQA handcrafted split, per-question averages. A turn is one tool-calling iteration of the agent loop. At Gemma 4 31B’s \$0.12 / \$0.35 per 1M input / output tokens (OpenRouter, accessed 2026-06), a question costs under \$0.02.

Dataset	LLM	#Q	Accuracy	Turns	Time	Tokens
KQAPro	Gemma 4 31B	500	85.4%	9.4	46.3 s	70.3 k
KQAPro	MiniMax M2.7	500	86.2%	7.9	56.6 s	58.2 k
SciQA	Gemma 4 31B	100	73.0%	7.0	51.2 s	114.5 k
SciQA	MiniMax M2.7	100	66.0%	6.8	51.6 s	87.2 k

entity search, label and summary retrieval, forward and reverse traversal, comparison and counting, deterministic numeric verification, and a raw SPARQL escape hatch; an optional tier (aggregation, group-by, superlative selection) is bound exactly when a graph’s data model supports it, and coverage is validated by the adapter test suite. Each bound tool is a deterministic, endpoint-specific SPARQL template with validated argument slots, and the same operation compiles to a different query shape per graph: compound facts are reached through reified statements and their qualifiers on KQAPro, but through contribution and comparison structures on ORKG. The agent issues the same typed calls in either case; the modelling idiosyncrasies stay inside the templates. The same single boundary is also where access policy can later be enforced, injecting named-graph filters or per-caller predicates before a query ever reaches the triple store.

KG onboarding. A hierarchy of dataclasses declares everything the reasoning core needs to know about a graph: Uniform Resource Identifier (URI) prefixes, endpoint and named-graph settings, and vector-collection bindings. Onboarding a new KG follows four steps. (1) The RDF dump is loaded into the triple store under a dedicated named graph, which keeps graphs that share one store isolated; an existing external endpoint can be used instead. (2) Entity labels and relation names are indexed into two vector collections, powering fuzzy entry-point discovery and the routing probe. (3) BaseKGAdapter is subclassed to return the populated configuration, and the graph is exposed through an MCP server that binds the abstract operations to its SPARQL templates. (4) The new sub-agent, which declares only its adapter and MCP-server path, is registered with the orchestrator. Adapter and configuration are a few hundred lines of largely declarative code; the SPARQL templates behind the abstract tools dominate the effort, so onboarding cost scales with how idiosyncratic the graph’s modelling is rather than with the reasoning core, which stays untouched.

Evaluation excerpt. Table 1 reports results with an open-weights backbone: 85.4% on 500 questions sampled from the Wikidata-derived KQAPro benchmark [13] and, through its own adapter without core changes, 73.0% on the complete handcrafted split of the SciQA benchmark [14] over the Open Research Knowledge Graph (ORKG), by a system that sees no training data, which sets the terms of comparison. Swapping the backbone for MiniMax M2.7, a configuration change touching no code, moves KQAPro to 86.2% and SciQA to 66.0%, so the ranking of backbones is itself graph-dependent. Fitted systems score higher on both benchmarks: KQAPro parsers trained on its 94,376-example split reach 90–92%, while among systems that do not fine-tune the strongest published KQAPro result we are aware of is Interactive-KBQA at 71.89% on a 900-question sample [5]. The benchmark’s own authors report their parser falling to 77.86% once program structures are held out [13]. On SciQA, the Scholarly QALD-23 leaderboard’s >0.99 F1 systems retrieve near-duplicate questions and their gold queries from the 1,795-question training split (96% of the benchmark stems from eight templates); porting them to a new graph requires an equally dense annotated corpus, the cost our abstract tool interface replaces.

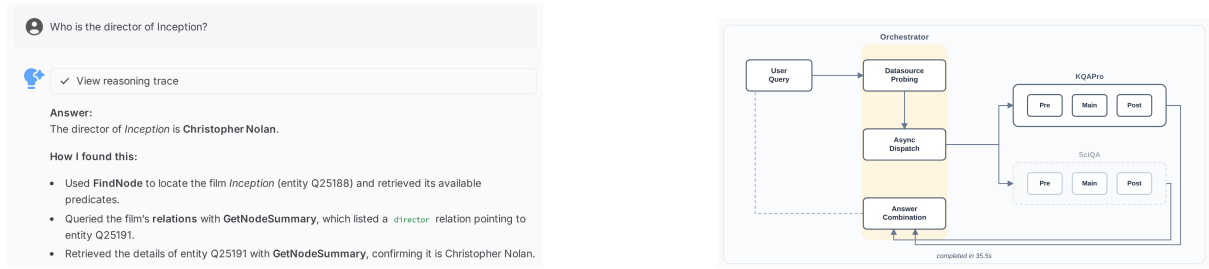


Figure 2: The demo answering a question via the Orchestrator. Left: the grounded answer cites the tool calls behind each step. Right: the live swarm view; the probe matched KQAPro, so only that sub-agent animates while SciQA stays inactive.

4. Demonstration

At our demo booth, attendees pick an agent (the Orchestrator by default, or a KQAPro/SciQA specialist with stateful multi-turn follow-ups) and a backbone LLM. While the agent runs, the UI animates the lifecycle of Fig. 1 (left) live, and in Orchestrator mode the swarm view of Fig. 2: stages light up as their trace spans open, and the routing probe’s per-source evidence feeds an explicit `select_agent` decision. Once the answer arrives (Fig. 2, left), a Trace tab exposes the full trace as a span tree, down to individual LLM messages, tool calls, and the raw SPARQL issued.

This trace-level transparency is also the demo’s central business use case: when answers feed decisions, every claim must be attributable to a source. Because synthesis relies on the facts the agent verified in the main loop to generate the final answer, any statement in it can be traced to the exact tool calls and triples supporting it, giving users a direct check that nothing was hallucinated. Per-question times in Table 1 are specialist-mode averages measured on a shared host against a shared community inference endpoint; repeated runs of the same configuration have differed by up to 40%, so timings are indicative, while accuracy reproduces within a few points. The public demo is slower still: it runs on a separate, smaller VM and defaults to Orchestrator mode, which adds a routing probe, a routing decision, and per-question agent start-up, and to a considerably larger backbone than Table 1’s Gemma 4 31B.

5. Conclusion

We showcased AMA-KBQA, a functional prototype of a training-free multi-agent system for KBQA over heterogeneous RDF graphs. A single reasoning core with per-graph MCP SPARQL-wrapper tools reaches competitive accuracy on KQAPro and ports to SciQA without any change to the core (Table 1), at practical runtimes and costs.

Limitations. Three limitations bound these claims. Both reference graphs are hosted inside a single triple store, which keeps the demo deployment simple but is unrealistic for production settings, where graphs sit behind separate, independently governed endpoints. We evaluate on two graphs only, one Wikidata-derived and one scholarly, so generalisation beyond these two modelling styles is untested. And the orchestrator routes a question to the matched sub-agent, but cannot yet dispatch to several at once or decompose a question whose answer spans several graphs. Accuracy on SciQA also still trails KQAPro’s, onboarding is not automated, and the wrapper boundary is designed for per-caller access control that we do not yet enforce.

Outlook. Addressing these, in particular cross-graph decomposition and automated onboarding, is our main direction towards a portable, model-agnostic KBQA ecosystem.

Acknowledgments

This work was funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project No. 459291153.

Declaration on Generative AI

The authors used Claude for grammar and spelling checks, paraphrasing, and implementation assistance; the LLMs inside AMA-KBQA are themselves the object of study. Generative AI was not used to conceive the research idea, make architectural decisions, formulate the evaluation, or draw conclusions. The authors reviewed and edited all code and content and take full responsibility for it.

References

- [1] G. Agrawal, T. Kumarage, Z. Alghamdi, H. Liu, Can knowledge graphs reduce hallucinations in LLMs?: A survey, in: Proc. of NAACL, 2024.
- [2] J. Berant, A. Chou, R. Frostig, P. Liang, Semantic parsing on Freebase from question-answer pairs, in: Proc. of EMNLP, 2013.
- [3] P. Lewis, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Proc. of NeurIPS, 2020.
- [4] M. R. A. H. Rony, et al., SGPT: A generative approach for SPARQL query generation from natural language questions, IEEE Access 10 (2022).
- [5] G. Xiong, J. Bao, W. Zhao, Interactive-KBQA: Multi-turn interactions for knowledge base question answering with large language models, in: Proc. of ACL, 2024.
- [6] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, J. Taylor, Industry-scale knowledge graphs: Lessons and challenges, Commun. ACM 62 (2019).
- [7] T. Soru, et al., SPARQL as a foreign language, in: Proc. of SEMANTiCS Posters & Demos, 2017.
- [8] L.-P. Meyer, J. Frey, F. Brei, N. Arndt, Assessing SPARQL capabilities of large language models, in: Proc. of NLP4KGC at SEMANTiCS, 2024.
- [9] S. Yao, et al., ReAct: Synergizing reasoning and acting in language models, in: Proc. of ICLR, 2023.
- [10] Anthropic, Model Context Protocol specification, 2024. URL: <https://modelcontextprotocol.io>.
- [11] L. Zheng, R. Wang, X. Wang, B. An, Synapse: Trajectory-as-exemplar prompting with memory for computer control, in: Proc. of ICLR, 2024.
- [12] M. Nye, et al., Show your work: Scratchpads for intermediate computation with language models, 2021. URL: <https://arxiv.org/abs/2112.00114>.
- [13] S. Cao, et al., KQA Pro: A dataset with explicit compositional programs for complex question answering over knowledge base, in: Proc. of ACL, 2022.
- [14] S. Auer, et al., The SciQA scientific question answering benchmark for scholarly knowledge, Scientific Reports 13 (2023).