

Answer-Convention Alignment for Agentic Text-to-SPARQL: AMA-KBQA at the WikiKGQA 2026 Challenge

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

¹Karlsruhe Institute of Technology (KIT), AIFB, Karlsruhe, Germany

Abstract

We describe our submission to the WikiKGQA 2026 challenge, which asks systems to return exact answer sets for natural-language questions over Wikidata, scored by Macro QALD F1. Our entry adapts AMA-KBQA, a multi-agent KBQA framework with a graph- and model-agnostic reasoning core without any model training. Because QALD F1 rewards reproducing the benchmark’s modelling conventions rather than merely a semantically correct answer, the adaptation centres on *answer-convention alignment*. The agent explores the knowledge graph through typed tools that insert its arguments into valid SPARQL templates, commits a single query whose executed bindings form the scored answer, and passes that commitment through per-question convention rules and a deterministic repair chain that steer the answer toward the gold form. The convention rules themselves are hand-derived from the training gold; the reusable contribution is the mechanism around them: commit-time repair, answer-sanity gating, and self-consistency voting over executed answer sets. For the without-mentions tracks, where no entity or property identifiers are supplied, we add live entity linking via `wbsearchentities` and a hybrid retrieval pipeline over Wikidata’s properties. On the English test sets AMA-KBQA places second by individual submission on the with-mentions track, and fifth by individual submission (third by team) on the without-mentions track, scoring **0.87** Macro QALD F1 with mentions and **0.79** without.

Keywords

KBQA, Text-to-SPARQL, Wikidata, Large Language Models, Model Context Protocol, QALD

Code <https://github.com/yannicd03/AMA-KBQA>

1. Introduction

The WikiKGQA 2026 challenge [1] poses question answering over Wikidata. Given a natural-language question in English or Spanish, a system must return the exact answer set over a fixed snapshot of Wikidata, a graph of roughly 115 million entities that is too large to embed or self-host without substantial infrastructure. Scoring is Macro QALD F1 [2] over four tracks, covering English and Spanish questions, each with or without pre-linked entity and property identifiers.

Two properties make this setting hard for LLM-based systems. First, models that write SPARQL freely are unreliable: even strong LLMs produce queries that fail to parse, hallucinate identifiers, or silently diverge from the schema [3]. Second, exact-set scoring is unforgiving of modelling choices. A query that is semantically defensible but follows a different convention than the gold query (labels instead of URIs, a list instead of a count, a lone winner instead of all tied entities) scores zero. Competitive F1 is therefore as much about *matching the benchmark’s answer conventions* as about finding the right facts.

Our submission builds on AMA-KBQA [4], a multi-agent KBQA framework in which a shared, training-free reasoning core drives a ReAct-style loop [5] over typed Model Context Protocol (MCP) [6] tools that wrap SPARQL requests. This system was previously instantiated for the KQAPro [7] and SciQA [8] benchmarks over self-hosted graphs. This paper reports what porting that framework to WikiKGQA required. Relative to [4], the challenge system is new in four respects: (i) a Wikidata tool layer that runs against live public endpoints, with no self-hosted triple store and no pre-built entity index;

Proceedings of the Wiki-Based Knowledge Graph Question Answering Challenge (WikiKGQA), co-located with the 25th International Semantic Web Conference (ISWC 2026), Bari, Italy, October 25-29, 2026

✉ 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).

(ii) an answer-convention alignment layer, comprising per-question injection of answer conventions and a commit-time repair chain, that treats the benchmark’s gold answer form as a first-class target; (iii) self-consistency voting gated by an answer-sanity guard; and (iv) live entity linking and a hybrid dense-sparse property index for the without-mentions tracks. The framework’s orchestrator, access-control, and lessons-learned components are not used in this submission. The reasoning core itself is reused unchanged, which is the portability claim this entry puts to the test. The challenge-specific work concentrates in the tool layer and in answer alignment, not in the agent itself. Section 3 first sketches the inherited framework so that this paper can be read on its own; Figure 1 marks each lifecycle stage as inherited, adapted, or new, and Table 1 classifies every component’s provenance, separating reusable mechanisms from heuristics fitted to the WikiKGQA training gold.

2. Related Work

AMA-KBQA’s framework positioning, its relation to Interactive-KBQA’s multi-turn control loop over atomic graph APIs [9], to scratchpad-style working memory [10], and to trajectory-as-exemplar prompting [11], is discussed in the framework paper [4] and not repeated here. This section situates only the components the challenge submission adds. Neural text-to-SPARQL models translate questions into queries end to end [12, 13] but must be trained per dataset, and their prompted-LLM successors inherit a documented brittleness at free-form query writing [3]. Our tools sidestep both problems by constraining the model to fill parameterised, pre-validated templates, trading expressive freedom for executable, schema-valid queries. Self-consistency decoding [14] improves reasoning by sampling several chains and keeping the modal answer. We apply the idea at the level of executed answer sets and add a sanity gate, because a vote among queries can otherwise elect a well-supported but malformed result. Finally, our per-question convention injection is a narrow form of retrieval-augmented prompting [15] in which the retrieved context is benchmark-specific answer-convention rules rather than documents.

3. Approach

The inherited framework. AMA-KBQA [4] addresses the fact that KBQA systems are usually rebuilt for each knowledge base. It separates a graph-agnostic reasoning core from graph-specific data access, and its agent lifecycle runs in three phases. A deterministic *pre-agent hook* classifies the question, extracts entity mentions, and seeds the prompt with the result. A ReAct-style *main loop* alternates LLM reasoning with typed tool calls and records what each call verifies in the journal scratchpad described below. A *post-agent hook* synthesises the final answer from the journal. The core (BaseKBQAAgent) owns this lifecycle and enforces loop detection and tool budgets. It reaches a graph only through a fixed contract of typed MCP tools that interpolate validated arguments into pre-written SPARQL templates, so no schema or endpoint knowledge is compiled into the reasoning core. Supporting a new graph therefore means implementing that tool contract for the new endpoint. The LLM is likewise a swappable component, with tool calls parsed from a plain-text protocol for open models that lack native function calling, so the backbones we compare in Section 4 all drive the same tool loop. The framework was evaluated on KQAPro [7] and SciQA [8], reaching 84% and 73% accuracy with a 31B open backbone.

What the challenge changes in the lifecycle. The lifecycle is reused unchanged, but two of its slots are filled differently (Fig. 1). In the pre-agent hook, the framework’s LLM-based classification and extraction step is replaced by deterministic extraction of the Q/P identifiers supplied in the without-mentions tracks, joined by the conditional injection of answer-convention rules described below. In the post-agent hook, the synthesis target changes: where the framework’s earlier instantiations emit a natural-language answer, the challenge agent synthesises one final SPARQL query, which is executed so that its result set becomes the scored answer. The agent therefore never writes answer text itself, but composes the final query from what it verified while exploring the graph. Everything downstream of

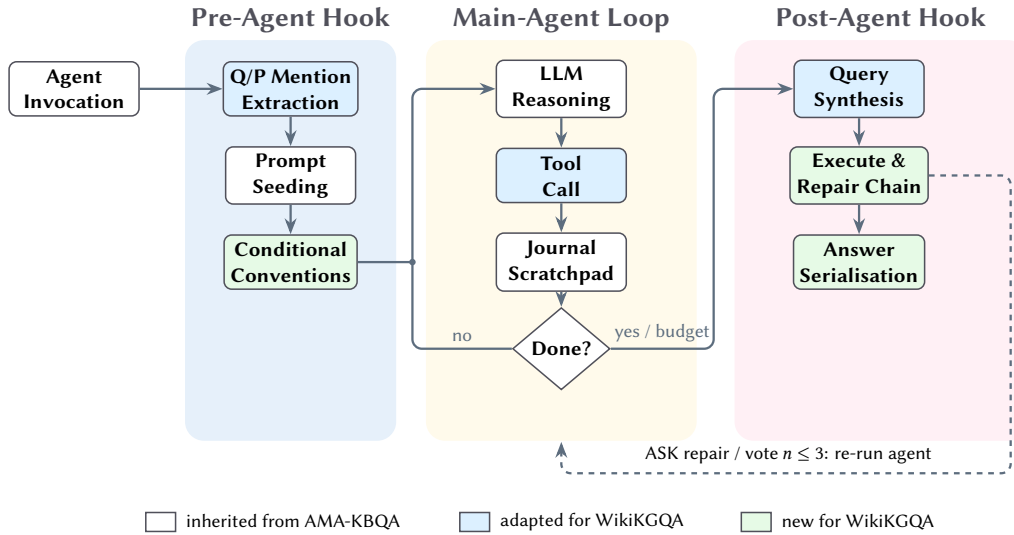


Figure 1: Agent lifecycle for WikiKGQA. The Pre-Agent Hook extracts any supplied Q/P identifiers from the question, seeds the prompt with them, and injects the answer conventions whose textual trigger the question matches. The Main-Agent Loop iterates LLM reasoning and typed tool calls, accumulating validated queries and verified facts in the journal scratchpad until the agent decides it is done or the tool budget forces synthesis. The Post-Agent Hook synthesises one final SPARQL query, executes it, and passes it through the deterministic commit-time repair chain (recovery from the journal, projection trim, class-closure expansion, sanity guard), then serialises the surviving bindings to the challenge answer format. Two paths re-run the agent rather than repairing the committed query in place (dashed): a yes/no question that commits a non-boolean triggers one ASK repair, and self-consistency voting draws up to n independent runs, stopping early once two agree. Each such re-run starts a fresh agent with a fresh tool budget. Fill colour marks provenance, matching the in-figure legend: white stages are inherited unchanged, blue stages are inherited slots filled with challenge-specific content, and green stages, like the dashed re-run path, exist only for WikiKGQA.

synthesis (execution, the commit-time repair chain, serialisation to the challenge format) is new for this submission.

Journal-based scratchpad. Agent state lives in a structured *journal* kept alongside the chat transcript, so that the state the synthesis step reads does not degrade as the transcript grows. It holds the question, its extracted target entities, visited nodes, discovered attribute values, verified facts, failed attempts, and the current plan. Exploration tools write to it automatically: every executed query records its full (untruncated) text, its projected variables, its exact result count, and up to its first 30 rows, so the journal accumulates validated candidate queries as a by-product of exploration. By default, synthesis reads only a compact journal summary rather than the full transcript, an A/B having found no benefit from the whole transcript (0.781 vs 0.807 F1, within run-to-run noise). Because the journal retains the last query that returned rows, a best validated query can be recovered even if the final synthesis call stalls.

SPARQL wrapper tools. Every interaction with the graph passes through typed MCP tools that validate their identifier arguments and deterministically interpolate them into fixed SPARQL templates before execution, rather than letting the model emit free-form SPARQL. Tools cover entity properties, incoming and outgoing relations, the relation between two entities, property metadata, statement qualifiers, and label lookup. The clearest example is `SelectExtreme`, which assembles a tie-safe superlative from typed parameters. It builds a `MAX/MIN` subquery plus a `FILTER(?v = ?m)` so that all entities tied for the extreme are returned, encoding an answer convention that a naive `ORDER BY . . . LIMIT 1` would violate. A single free-form `RunSPARQL` escape hatch remains for exploration, and it caps the rows it shows the model. Such a query is not the synthesis path, but it can still become the committed answer when the repair chain recovers it from the journal, in which case it is re-executed in full and

scored on its complete result set. A per-question budget of 20 tool calls (30 in the without-mentions tracks), surfaced in the system prompt and as a live counter on each tool result, is the binding stopping condition. At the budget the framework forces synthesis of the best validated query, which proved more reliable than a wall-clock timeout that could return no query at all.

Wikidata query API. Queries execute against the Wikidata Query Service during development and against the challenge endpoint for submission. The client retries transient failures with linear backoff and bails out immediately on 4xx errors. A per-tool 30-second timeout returns a descriptive error message instead of hanging, which bounds cost and steers the model away from broad queries. Because the gold answers were computed against a frozen KG snapshot, every outgoing `NOW()` call is rewritten to a fixed reference instant, applied identically during exploration and at commit time so that age and date questions reproduce the evaluation’s clock.

Answer conventions. QALD F1 compares the executed answer set against one specific gold query, so a semantically correct query that follows a different modelling convention scores zero. We therefore encode the benchmark’s answer conventions explicitly: return entity URIs rather than labels, map yes/no questions to ASK, return all ties for superlatives, default “how many” to a COUNT scalar, resolve “country” to sovereign state, project exactly one variable, and so on. These are not a published specification. We derived them by hand from the training gold, so they are an empirical generalisation over the examples we saw rather than a guarantee about the test set, and Section 4 reports two questions on which the gold itself is inconsistent. Rules with a clean textual trigger are injected per question only when the question matches, because a held-out A/B (Section 4) found no benefit from injecting the full rule bundle on every question.

Commit-time repair chain. Prompt conventions are only advisory, so a chain of repairs then coerces the committed query toward the gold form. The chain exploits one property of this benchmark. Every gold answer is non-empty and drawn from a known value grammar, so a query that errors, returns no rows, or returns malformed URIs (blank nodes or statement nodes) is known to be wrong. If the committed query fails or yields an empty or malformed result, the chain swaps in a different validated query recovered from the journal. A projection trim reduces accidental multi-column SELECT queries to their first variable (94% of gold SELECTs are single-column). A class-closure expansion widens a bare `wdt:P31` to a transitive closure, adopting the wider query only when its answer set is a strict, still-valid superset. These stages are deterministic rewrites over queries the agent has already validated. One stage is not: an ASK repair re-runs the agent once, with an explicit instruction to emit ASK, when a yes/no question commits a non-boolean, and keeps the original result unless the re-run returns a boolean. For robustness we also apply self-consistency voting over the executed answer sets of all question types: the query is generated up to n times, generation stops early once two runs agree, and the modal answer set is committed. Our experiments use $n = 3$ as a cost compromise. We did not complete an A/B isolating the effect of n , so we make no accuracy claim for it; the one scored effect we can attribute to voting is the submission-2 regression discussed in Section 4. Voting also motivates the answer-sanity guard. A vote once committed a fifty-row statement-node result over a correct single answer, so the guard, which rejects blank-node and statement-node URIs, gates the adoption steps above rather than acting alone. We nonetheless keep voting in the final system: it protects against single-run failures such as a stalled synthesis call, and early stopping bounds its cost at two agreeing runs in the common case. Finally, any answer still empty after the main pass is re-run once with an enlarged tool budget, and the surviving bindings are serialised to the challenge format, reducing entity URIs to bare Q/P identifiers and passing literals through verbatim.

Entity and Property Linking for the without-mentions tracks. In the without-mentions tracks the input supplies no identifiers, so the agent must resolve entities and relations itself. We do not embed

Table 1

Component provenance. Inherited components are unchanged framework code; adapted components fill inherited extension slots with challenge content; components that are new for WikiKGQA are split into reusable mechanisms and heuristics whose content is derived from the training gold.

Provenance	Components
Inherited, unchanged	Reasoning core and three-phase lifecycle; journal scratchpad; typed-tool contract (incl. <code>SelectExtreme</code> , <code>RunSPARQL</code>); loop detection and tool budgets; hybrid dense-sparse retrieval module; journal-recovery primitive (only its commit-time use is new)
Adapted: inherited slot, challenge content	Wikidata SPARQL templates behind the tool contract; deterministic Q/P mention extraction; SPARQL-emitting synthesis; property index over the filtered 3.2k-property collection
New: reusable mechanism	Repair-chain control flow (journal recovery on empty or malformed commits); answer-sanity gating; self-consistency voting over executed answer sets; <code>NOW()</code> pinning; live entity linking via <code>wbsearchentities</code> ; challenge-format serialisation
New: derived from training gold	Answer-convention rules and their textual triggers; single-column projection trim; class-closure escalation; ASK-repair trigger

Wikidata’s 115M entities. Instead, entity mentions are linked live through the `wbsearchentities` MediaWiki API, which returns a short candidate list of labels and descriptions that the agent disambiguates in context. Relations are harder, because lexical search over property labels misses indirect phrasings (searching “played” misses *cast member* and *performer*). We therefore build a small semantic index over Wikidata’s roughly 13.1k properties (as of July 2026). After dropping external-identifier properties (about three quarters of the total, never answer relations), the index holds roughly 3.2k properties, each represented by a dense embedding of its “label: description” and a BM25 sparse vector over its “label | aliases”, so that alias phrasings stay recoverable. Retrieval fuses the dense and sparse hits and returns ranked property candidates through a `SearchProperties` tool. Both linkers are enabled only for our without-mentions runs; the with-mentions runs use the supplied identifiers directly.

Provenance: inherited, adapted, and benchmark-derived. Table 1 classifies the components above by provenance. *Inherited* components are shared framework code used unchanged. *Adapted* components fill an inherited extension slot with Wikidata- or challenge-specific content, which is the framework’s intended onboarding path. Among the components that are *new* for this challenge, we further separate reusable mechanisms, whose control flow contains nothing WikiKGQA-specific, from heuristics whose content was mined from the WikiKGQA training gold. The distinction matters for reuse: the mechanisms in the third group would port to another exact-match KGQA benchmark as they stand, while every entry in the fourth group (the convention rules and their trigger phrases, the single-column projection trim, the class-closure escalation, the ASK-repair trigger) is an empirical fit to this benchmark’s training distribution and would need re-derivation elsewhere.

4. Results

We evaluated on the WikiKGQA test sets [16], 75 questions per track, scored by Macro QALD F1. We entered the English tracks only. All submissions use Qwen3.5 (Qwen3.5-397B-A17B), a 397B mixture-of-experts backbone with 17B active parameters, and the minimal answer-convention set. Every figure we report is a single run: the agent is stochastic and exploration trajectories diverge between runs, so we make no claim about deltas below the one-question floor, which is about 0.013 F1 on the 75-question test tracks and about 0.02 F1 on our 50-question development splits. We made three scored submissions per track, each a snapshot of the growing commit-time repair chain of Section 3. The initial submission predates the chain entirely. The second adds class-closure expansion, `NOW()` pinning, and

Table 2

English WikiKGQA test results (75 questions per track, Macro QALD F1). Submission 1 was made on 2026-07-03, submissions 2 and 3 on 2026-07-15; their contents are described in the text. “Best reported” is the strongest score submitted on each track by any team. Bold marks our best submission per track, not the best value in the row.

Track	Sub. 1	Sub. 2	Sub. 3	Best reported
With mentions	0.85	0.84	0.87	0.88
Without mentions	0.78	0.79	0.78	0.85

three-way self-consistency voting. The third adds the answer-sanity guard and the final repair package (ASK repair, projection trim, conditional convention injection, empty-answer escalation). Because each submission bundles several changes, the test-set deltas in Table 2 attribute score movement to bundles, not to individual components, and all of them sit near the one-question floor. Component-level evidence, where we have it, comes from the held-out development A/Bs reported in this section and in Section 3; for the components without a scored A/B (projection trim, ASK repair, the sanity guard, and the voting depth n) we make no per-component accuracy claim. Table 2 reports all three, alongside the best reported score on each track. On the with-mentions leaderboard AMA-KBQA places second by individual submission. On the without-mentions leaderboard it places fifth by individual submission and third by team, because two of the teams ahead of it entered several runs each. Our best scores trail the strongest reported result by 0.01 and 0.06 F1 respectively, without any training and without self-hosting Wikidata.

Dev-to-test transfer and error analysis. The repairs added in submission 2 were gated on a paired 30-question held-out run, where they lifted Macro F1 from 0.717 to 0.834 (+0.117). On the test sets this compressed to roughly ± 0.01 (-0.01 with mentions, $+0.01$ without). The class-closure failures the repairs target are rarer in the test distribution than in our development split, so paired development gains on known failures should not be read as expected test deltas. The with-mentions regression in submission 2 nets across nine questions whose answer sets changed. Its dominant single contributor (about -0.013 macro F1) is one question on which a self-consistency vote committed a query that resolved to statement nodes rather than entities; serialisation reduced them to malformed identifiers, turning a previously correct single answer into fifty spurious rows. This is the failure that motivated the answer-sanity guard. The guard’s malformed-URI patterns had earlier been assessed as unreachable because no training gold answer contains such shapes. The system’s own output can still emit them, so sanity checks must validate against the reference value grammar, not only against shapes observed in gold data. The guard shipped in submission 3 together with the final repair package, whose pieces are grounded in training-gold scans rather than in scored A/Bs. Submission 3 more than recovered the regression with mentions (0.84 to 0.87, our best score and 0.01 behind the winner) while without mentions slipped back 0.79 to 0.78. One residual failure class lies beyond the repair chain entirely. On a contraindications question the agent deterministically projects RDF predicates as if they were entity answers, a genuine modelling gap in relation resolution rather than run-to-run noise.

Backbone comparison. The framework is model-agnostic, so we compared several backbones on a fixed 50-question development set (Table 3). The two weaker backbones were run at an earlier code state than the Qwen3.5 figure quoted here, so the table separates the backbones by a wide margin but does not isolate the model as the only difference. Qwen3.5 is markedly stronger at producing executable, convention-correct SPARQL than the smaller open models, which is why it backs every submission, and the two weaker backbones fail in distinct ways. Gemma 4 is reliable but shallow, committing an executable query on all 50 questions while losing on the deeper multi-hop cases, though four of its answer sets came back empty. MiniMax’s weakness is query specificity. Its queries frequently omit type anchors and join constraints, inflating answer sets and depressing precision. A question about movies directed by siblings returned 19,425 rows because its query only required each director to have a

Table 3

Backbone comparison on a 50-question development set (Macro QALD F1). The Gemma and MiniMax runs predate the Qwen3.5 run by two days and a number of framework changes, so the table is not a controlled sweep at one code state.

Backbone	Macro F1
Qwen3.5 (397B-A17B)	0.81
Gemma 4 (31B)	0.70
MiniMax M2.7 (229B)	0.57

sibling. It also returned empty answer sets on 8 of 50 questions and once committed prose reading “Final Answer: Yes” in place of a SPARQL ASK, echoing an earlier audit that found it prone to abandoning generation mid tool-call block. All backbones run through the same text-mode tool-call shim, so the comparison reflects reasoning quality rather than API compatibility.

Answer conventions and generalisation. On a held-out 50-question split, the minimal convention set (six always-on rules) scored 0.731 F1 against 0.683 for injecting the full rule bundle unconditionally. Only 11 of the 50 questions differ between the two arms (7 favouring minimal, 4 favouring full), and per-question inspection traced the largest full-arm losses to divergent exploration trajectories rather than to the extended rules themselves. We therefore read this as no measurable benefit from always-on injection rather than as evidence that it hurts, and default to the shorter prompt on cost and overfitting-risk grounds. The submitted system keeps the extended rules off by default and injects three of them only on questions whose narrow textual trigger matches, so most questions receive none. This held-out 0.731 is also our most honest estimate of generalisation. It sits below the 0.807 we measure on the split used for tuning, a gap that reflects mild overfitting of the hand-derived conventions to the training gold. The two splits are disjoint 50-question samples rather than the same questions under two configurations, so the comparison is indicative rather than paired. Some gold choices are genuinely unwinnable under any single policy, for instance two “how many” questions whose gold answers disagree on whether to return a count or a list. That the gold is not internally consistent on this point is also why we describe the target as a set of conventions rather than a specification. We accept a small, unavoidable F1 loss on such benchmark artifacts.

Efficiency. Each question runs a fresh agent and tool server with the journal reset, under a 20-call tool budget (30 in the without-mentions tracks), a 30-second per-query timeout, and a 600-second per-question wall-clock cap. Combined with commit-time recovery of the best validated query, this was sufficient for both final submissions to carry a non-empty, well-formed answer for all 75 questions.

5. Conclusion

We described the adaptation of the AMA-KBQA framework [4] to the WikiKGQA 2026 challenge, in which the reasoning core was reused unchanged and the challenge-specific work concentrated in the live-endpoint Wikidata tool layer and the answer-convention alignment layer. The main remaining gaps are the without-mentions setting, where entity and property linking rather than query construction appears to be the bottleneck (an attribution we could not isolate experimentally, since the two English tracks pose disjoint question sets), and the hand-derived nature of the answer conventions. Automating their discovery from a benchmark’s training gold would turn answer-convention alignment from bespoke effort into a reusable component for any exact-match KGQA evaluation.

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] S. Arzoo, D. Banerjee, P. Cimiano, A. Hogan, A. Moya Loustaunau, D. M. Schmidt, R. Usbeck, WikiKGQA: Wiki-based knowledge graph question answering challenge, 2026. URL: <https://wikikgqa.org/>, co-located with ISWC 2026.
- [2] R. Usbeck, M. Röder, et al., Benchmarking question answering systems, *Semantic Web* 10 (2019) 293–304.
- [3] L.-P. Meyer, J. Frey, F. Brei, N. Arndt, Assessing SPARQL capabilities of large language models, in: *Proc. of NLP4KGC at SEMANTiCS*, 2024.
- [4] Y. Duan, M. Klattig, M. Bardenhagen, Y. Li, T. Käfer, AMA-KBQA: An adaptable multi-agent framework for generalizable knowledge base question answering, in: *Proc. of SEMANTiCS Posters & Demos*, 2026. To appear.
- [5] S. Yao, et al., ReAct: Synergizing reasoning and acting in language models, in: *Proc. of ICLR*, 2023.
- [6] Anthropic, Model Context Protocol specification, 2024. URL: <https://modelcontextprotocol.io>.
- [7] S. Cao, et al., KQA Pro: A dataset with explicit compositional programs for complex question answering over knowledge base, in: *Proc. of ACL*, 2022.
- [8] S. Auer, et al., The SciQA scientific question answering benchmark for scholarly knowledge, *Scientific Reports* 13 (2023) 7240.
- [9] 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.
- [10] M. Nye, et al., Show your work: Scratchpads for intermediate computation with language models, 2021. URL: <https://arxiv.org/abs/2112.00114>.
- [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] T. Soru, et al., SPARQL as a foreign language, in: *Proc. of SEMANTiCS Posters & Demos*, 2017.
- [13] M. R. A. H. Rony, et al., SGPT: A generative approach for SPARQL query generation from natural language questions, *IEEE Access* 10 (2022).
- [14] X. Wang, et al., Self-consistency improves chain of thought reasoning in language models, in: *Proc. of ICLR*, 2023.
- [15] P. Lewis, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks, in: *Proc. of NeurIPS*, 2020.
- [16] S. Arzoo, A. Hogan, A. Moya Loustaunau, D. M. Schmidt, QAWiki v2026.2: Knowledge graph question answering (KGQA) / SPARQL query generation dataset for Wikidata, 2026. doi:10.5281/zenodo.19474009.