GRASP and GRISP for Question Answering over Wikidata

Sebastian Walter[1,*], Hannah Bast[1]

[1]University of Freiburg, Georges-Köhler-Allee 51, 79110 Freiburg im Breisgau, Germany

Abstract

We describe our submissions to the WikiKGQA 2026 challenge on generating SPARQL queries for English and Spanish natural language questions over Wikidata. We use two methods: GRASP, an LLM agent that iteratively explores the knowledge graph and refines a query, and GRISP, a lightweight fine-tuned model that predicts a query skeleton and fills it with IRIs. We only entered the two *without-mentions* tracks, and we also report standalone Claude Code agent baselines to show what an off-the-shelf agentic LLM achieves without dedicated KGQA tooling. GRASP gets the best result of all submissions in both without-mentions tracks, matching the strongest Claude Code baseline at a fraction of its cost. GRISP is a much cheaper, fully local alternative, but its results are noticeably weaker.

Keywords

Knowledge Graph Question Answering, Large Language Model, Agent, SPARQL, Wikidata

1. Introduction

The WikiKGQA 2026 challenge asks systems to translate English and Spanish natural language questions into SPARQL queries against Wikidata [1]. We enter two knowledge-graph-agnostic systems that were developed independently of this challenge: GRASP [2], an LLM agent that explores the knowledge graph and iteratively drafts, tests, and refines a query, and GRISP [3], a lightweight fine-tuned model that generates a query skeleton and fills it with IRIs selected from the knowledge graph. GRASP sets itself up from the challenge-provided Wikidata dump without manual effort, and GRISP then reuses this setup. We also report standalone Claude Code [4] agent baselines as a reference point and discuss all results in Section 4. Our evaluation, materials, and code are available at grasp.cs.uni-freiburg.de/wiki-kgqa-evaluation, ad-publications.cs.uni-freiburg.de/grasp/wiki-kgqa and github.com/ad-freiburg/grasp, respectively.

2. The WikiKGQA 2026 Challenge

Given a natural language question, the task is to generate a *single* SPARQL query that answers it against the version of Wikidata fixed by the challenge. Answers must be produced either via the challenge's SPARQL endpoint or its provided Wikidata dump,[1] so that results are consistent with the evaluation version of the knowledge graph.

The challenge has four tracks, one for each of English and Spanish, split further into a without-mentions and a with-mentions variant, each with its own leaderboard. In the without-mentions track the system receives only the question, while in the with-mentions track the question additionally comes with entity and property annotations from the QAWiki metadata.

The data is derived from QAWiki [5], a community-edited collection of hand-crafted question-query pairs over Wikidata, with a full snapshot available on Zenodo.[2] Each of the four tracks provides

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

*Corresponding author.

swalter@cs.uni-freiburg.de (S. Walter); bast@cs.uni-freiburg.de (H. Bast)

https://ad.informatik.uni-freiburg.de/staff/walter (S. Walter); https://ad.informatik.uni-freiburg.de/staff/bast (H. Bast)

0009-0006-2613-3209 (S. Walter); 0000-0003-1213-6776 (H. Bast)

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

[1]21 billion triples, available at https://ag-sc.techfak.uni-bielefeld.de/download/wikikgqa/.

[2]https://doi.org/10.5281/zenodo.19474009

Questions

English: "Who has declined a Fields Medal?"

Spanish: "¿Quién ha rechazado una Medalla Fields?"

English Mentions

"Fields Medal" → wd:Q28835
"Medal" → wd:Q618779
"declined" → wd:P793 + wd:Q26252351
"Fields" → wd:Q315431

SPARQL

```
SELECT DISTINCT ?sbj WHERE {
  ?sbj p:P166 ?obj .
  ?obj ps:P166 wd:Q28835 .
  ?obj pq:P793 wd:Q26252351 .
}
```

Figure 1: A challenge sample: its English and Spanish question, the four English mention annotations, and the reference SPARQL query (answer: Grigori Perelman, wd:Q117346). The without-mentions track provides only the question; the with-mentions track additionally provides the annotations shown. The annotation for "declined" is a single mention combining an entity and a property, expressing that the award was declined (wd:P793 with value wd:Q26252351).

317 training, 100 validation, and 75 test questions. The English and Spanish variants of a track are translations of one another sharing the same reference SPARQL query, which is what makes our Spanish results directly comparable to our English ones. The with-mentions and without-mentions variants pose the same questions on the training and validation splits, but their test splits are disjoint.

The official metric is macro QALD-F_1, computed on Codabench following the definition used by GERBIL QA [6], and this is what we report for the test set. On the validation splits we instead report the F_1 we compute ourselves, following the evaluation protocol of [2]: F_1 averaged per sample, but with a modification that permits additional columns in the predicted result compared to the ground truth without penalization, since this commonly happens with LLM-generated queries. Because we post-process every predicted query to a single answer column before submitting it (Section 4), the two protocols coincide on the runs we report. Solutions are submitted via Codabench.[3] The challenge states no restriction on external data or LLM usage.

Figure 1 shows a representative sample, with all information available for it across tracks: both variants share the question and its gold query, but the with-mentions track additionally supplies entity and property annotations. We motivate our focus on the without-mentions tracks based on such examples in Section 3.

3. Approach

We entered only the two without-mentions tracks, since giving the system just the question feels closer to how a user would actually pose one, rather than handing it a question already annotated with entities and properties. Inspecting the with-mentions training data, we also found the annotations to be heavily over-segmented, as illustrated by the example in Fig. 1: on the English train split, questions carry 4.6 mention annotations on average, and in 62.5% of them one mention string is nested inside another, as "Fields" and "Medal" are inside "Fields Medal". Only 56.7% of the annotated IRIs occur in the reference query, while the property the reference query is built around, wdt:P166 in Fig. 1, is not annotated at all. We do not read the surplus annotations as errors, since a system might well reach the same answer through wdt:P138 and wd:Q315431 instead of wd:Q28835, and there need not be a canonical set of IRIs to annotate. But a system that trusts the annotations still has to decide which of them to ignore, which is work the without-mentions setting does not create. The test split is annotated more sparingly than the train split: there, questions carry 3.0 mentions on average and only 5.3% of them contain a nested mention string.

The leaderboards suggest we were wrong to be wary. They cannot settle the question, since the with-mentions and without-mentions test sets cover disjoint questions, but on English the best with-mentions submission reaches 88.0 macro QALD-F_1 against the 85.0 of our best without-mentions run, and both teams that entered both English tracks score higher with mentions than without, 81.0 to 88.0 and 79.0 to 87.0. The Spanish with-mentions track has a single submission and is too thinly populated to say anything. Even imperfect annotations thus appear to help.

[3] https://www.codabench.org/competitions/15358

We also ran all our systems against our own local QLever [7] instance loaded with the challenge dump rather than against the challenge's hosted endpoint, which gives us an endpoint under our own control, with no contention from other users and with resource limits we choose ourselves; neither system relies on QLever-specific extensions.

3.1. GRASP

GRASP [2] generates a SPARQL query by letting an LLM interact with the knowledge graph autonomously, interleaving reasoning steps with calls to a fixed set of functions for searching and querying the knowledge graph, in a style similar to ReAct prompting. Functions include executing a SPARQL query, listing triples matching a pattern, and several search functions over entities, properties, and literals; the model picks whichever functions it needs and stops by calling a final *answer* function with the SPARQL query and a human-readable answer. All SPARQL functionality is backed by the local QLever instance described above, which both supplies the data for GRASP's search indices and executes every query GRASP issues at runtime.

GRASP is entirely knowledge-graph-agnostic and requires no manual engineering to support a new knowledge graph: given only a SPARQL endpoint, it sets itself up through the following steps, each building on the partial setup produced by the previous ones:[4]

1. `grasp setup info` configures prefixes and a short description of the knowledge graph.
2. `grasp notes explore` in structural mode lets the model explore the knowledge graph around a handful of seed entities and properties, taking notes as it goes: short pieces of text later added to the model's context to improve performance.
3. `grasp setup indices` has the model find, for each of the entity, property, and literal indices, a SPARQL query that extracts the data to index and one that retrieves display metadata; `grasp data` then fetches the results of these queries and `grasp index` builds the indices from them.
4. `grasp setup shapes` has the model determine an instance pattern and a schema pattern: SPARQL graph patterns that identify a class's instances and its properties with their value types. From these patterns, `grasp index` builds the shape index storing ShEx-like shapes for classes, which the model can then search during inference. The index covers the 500 most common classes; shapes for other classes are computed on the fly.
5. `grasp notes explore` runs again, this time in functional mode, letting the model take notes on how all of its functions behave in practice. This step comes last, so that all search functions are available.

Two further, optional steps refine this setup with task-specific notes. For the *self-QA* configurations, `grasp notes generate-questions` has the model invent questions about the knowledge graph, which `grasp notes samples` then answers, taking notes along the way. For the *samples* configurations, we instead first build a few-shot example index from the WikiKGQA train set with `grasp examples`, retrievable later via the *search_example* function, and then run `grasp notes samples` on the WikiKGQA validation questions with that example index available. For each such question, the note-taking model is shown a GRASP trace for it alongside the reference question and query, and takes notes from that comparison. Since these notes are taken on the validation questions themselves, we report no validation F_1 for the samples configurations.

Setting a knowledge graph up this way is a one-time cost and is typically cheap; we give concrete figures for our runs in Section 4. For our challenge submissions, we ran this pipeline once on the official dump. The *Spanish setup* used in some of our Spanish-track runs is a partial rerun on top of it: the entity and property indices are rebuilt from extraction queries extended to also cover Spanish labels and aliases, and the functional notes are retaken in Spanish, while the knowledge graph description, the shape index, and the structural notes are reused from the English setup.

[4] Simplified, omitting flags and config arguments; see github.com/ad-freiburg/grasp for the exact commands.

3.2. GRISP

GRISP [3] is a much lighter-weight alternative to GRASP: instead of an LLM agent, it fine-tunes a small language model to generate SPARQL queries in two stages. First, beam search produces several candidate query *skeletons*, i.e. syntactically valid SPARQL queries with natural-language placeholders in place of IRIs. Second, GRISP resolves the placeholders of the best skeleton one at a time, left to right, by searching GRASP's entity and property indices for each placeholder's label, re-ranking the retrieved candidates with the same model, and backtracking to a previous choice if a candidate leads to a dead end or an invalid or empty result. At each step, GRISP also constrains the search to IRIs that are actually reachable given the query resolved so far, by executing the corresponding partial query against the knowledge graph.

We fine-tune Qwen2.5 [8] 7B and 14B models with LoRA for each challenge track, starting from checkpoints already pre-trained on the Wikidata Query Logs Dataset [9] (*WDQL*). For each language, we train one model on only that language's WikiKGQA train split, and a second, combined model (*all KGQA sources*) that additionally uses the other language's split, QALD-7 [10], QALD-10 [11], and SPARQL queries extracted from GRASP's note-taking traces and converted to question-query pairs via GRASP's SPARQL-to-Question task. These traces come from GRASP's exploration of Wikidata and its self-QA step only, never from the note-taking runs over the WikiKGQA validation questions, so no validation data enters GRISP's training set.

3.3. Claude Code Baselines

As a reference point, we also evaluate standalone Claude Code agents prompted to answer the questions with their off-the-shelf tooling. All runs use Claude Code's `/auto` mode. We tried different ways of using Claude Code for this: a single agent session answering all questions in turn, and an orchestrator that spawns one sub-agent per question. Some runs of each additionally use Claude Code's `/goal` feature, which sets a completion condition that a smaller model checks after each turn. While the condition does not hold, the session continues automatically instead of handing control back to the user. In all cases, we first prompted the agent to explore the Wikidata SPARQL endpoint and prepare itself for the task, before letting it answer the questions one by one.

For some sub-agent orchestrator runs, we additionally gave the orchestrator information beyond the endpoint: for our strongest validation-split run, it saw all validation questions upfront, though never the gold SPARQL, before dispatching them to per-question sub-agents; for our test-split run, it instead had access to the train and validation set samples, matching GRASP's samples setting (Section 3.1).

4. Results

As backbone LLMs, we use OpenAI's GPT-5.4 [12] (including its mini variant), GPT-5.5 [13], Moonshot AI's Kimi K2.7 [14], and Anthropic's Claude Sonnet 4.6 [15] for GRASP, and Claude Sonnet 4.6 and Opus 4.8 [16] for the Claude Code baselines. All models are used with medium reasoning effort. Table 1 reports our results on the validation splits, Table 2 those on the held-out test splits.

Query Post-Processing. All methods, including the Claude baselines, predict full SPARQL queries, but the challenge metric requires a single-column answer list per question. We bridge this gap uniformly with a Claude Haiku 4.5 [17] agent that rewrites each query to project a single answer variable (passing ASK and already single-column queries through unchanged) and verifies the rewrite against the endpoint, retrying if it changes the result rows. This reproduces our own validation F_1 exactly, which confirms the rewrite is faithful.

GRASP Across backbones, GPT-5.4 performs best (86.8 English, 87.3 Spanish on validation), with GPT-5.5 a close second; Sonnet 4.6 trails both by several points, and the mini variant further still. GRASP's accuracy therefore depends substantially on the backbone. The backbones we compare were

Table 1
F_1-score (%) following the evaluation protocol of [2] for our GRASP, GRISP, and Claude Code baseline runs on the English and Spanish validation splits of the without-mentions tracks. The *Details* column provides run-specific information, such as average runtime per question (s/q), steps per question (S/q), or approximate cost of the full run in $. For GRISP, WikiKGQA fine-tuning uses only the corresponding track's train set in the respective language, whereas the fine-tuning on all KGQA sources includes QALD-7, QALD-10, as well as WikiKGQA train sets and GRASP trace-derived samples in both English and Spanish. The WDQL tag marks models pre-trained on the Wikidata Query Logs Dataset, which are either used directly or to initialize fine-tuning. GRASP's per-run costs exclude the one-time cost of building the English/Spanish setup, roughly $11/$10 using GPT-5.4, plus $8 for self-QA and $5 for samples. Costs for GRASP are estimated from the token usage reported in its run traces at published token prices; we did not track costs for the Claude Code runs on the validation splits.

Method	Description	F_1	Details
English without-mentions track			
GRASP	GPT-5.4	86.8	60.6 s/q, 9.6 S/q, $7
	GPT-5.5	86.3	47.4 s/q, 8.5 S/q, $6
	GPT-5.4 with self-QA	84.1	46.5 s/q, 9.6 S/q, $7
	Sonnet 4.6	80.7	37.2 s/q, 7.4 S/q, $7
	Sonnet 4.6 with self-QA	80.6	35.4 s/q, 6.8 S/q, $6
	GPT-5.4 mini	78.1	34.3 s/q, 8.0 S/q, $2
GRISP	Qwen2.5 14B WDQL fine-tuned on all KGQA sources	68.7	14.3 s/q
	Qwen2.5 14B WDQL fine-tuned on WikiKGQA	63.2	21.9 s/q
	Qwen2.5 14B WDQL	58.2	10.8 s/q
	Qwen2.5 7B WDQL fine-tuned on WikiKGQA	56.2	18.0 s/q
	Qwen2.5 14B fine-tuned on all KGQA sources	55.7	17.5 s/q
	Qwen2.5 7B WDQL	53.1	7.6 s/q
	Qwen2.5 14B fine-tuned on WikiKGQA	52.3	27.2 s/q
	Qwen2.5 7B fine-tuned on WikiKGQA	47.7	20.2 s/q
Claude Code	Opus 4.8 with Sonnet 4.6 sub-agents with questions upfront	86.6	auto mode, goal
	Opus 4.8 with Sonnet 4.6 sub-agents	80.8	auto mode
	Sonnet 4.6 with Sonnet 4.6 sub-agents	76.0	auto mode, goal
	Sonnet 4.6 in a single context	72.0	auto mode, goal
Spanish without-mentions track			
GRASP	GPT-5.4 with Spanish setup	87.3	59.7 s/q, 9.5 S/q, $7
	GPT-5.4	82.0	59.1 s/q, 9.5 S/q, $7
GRISP	Qwen2.5 14B WDQL fine-tuned on all KGQA sources	69.3	14.9 s/q
	Qwen2.5 14B WDQL fine-tuned on WikiKGQA	61.8	20.2 s/q
	Qwen2.5 7B WDQL fine-tuned on WikiKGQA	53.3	16.6 s/q
	Qwen2.5 14B fine-tuned on all KGQA sources	50.7	20.9 s/q
	Qwen2.5 7B fine-tuned on WikiKGQA	40.5	26.0 s/q

not released at the same time, so part of this spread reflects model generation rather than anything about GRASP. Kimi K2.7 also trails GPT-5.4, by 4 points on both test tracks (81.0 vs. 85.0 English, 79.0 vs. 83.0 Spanish), under otherwise identical setups. Comparing languages, the dedicated Spanish setup step raises validation F_1 by over 5 points on the Spanish track (82.0 to 87.3), yet on test, Spanish still trails English by 2 points (83.0 vs. 85.0). Neither optional note-taking step helps uniformly: self-QA costs GPT-5.4 2.7 points on validation (86.8 to 84.1) and leaves Sonnet 4.6 unchanged (80.7 to 80.6); samples add 5 points on the English test track (80.0 to 85.0) but nothing on Spanish (83.0 to 83.0), where the Spanish setup step on its own already reaches 83.0.

GRISP Both model size and pre-training matter: the 14B model consistently beats its 7B counterpart by 4.6 to 8.5 points, and starting from the WDQL-pre-trained checkpoint instead of base Qwen2.5 adds another 10+ points (52.3 to 63.2 for 14B fine-tuned on WikiKGQA). More fine-tuning data helps further:

Table 2
Official macro QALD-F_1 (%) for our GRASP, GRISP, and Claude Code runs on the English and Spanish test splits of the without-mentions tracks, as evaluated on Codabench. Gray rows are other teams' submissions; on the Spanish without-mentions track, no other team submitted a result. The *Details* column provides run-specific information, such as average runtime per question (s/q), steps per question (S/q), or approximate cost of the full run in $. Runs with samples have access to the corresponding track's train and validation set samples during setup and inference. As on the validation split (Table 1), GRASP's per-run costs exclude the one-time cost of building the setup. Kimi K2.7 ran on a university-hosted endpoint without a cost readout, so its *est.* figures are computed from the token counts in its run traces at OpenRouter's published token prices for Kimi K2.7 Code ($0.71, $0.15 and $3.50 per million uncached input, cached input and output tokens). Since that endpoint reports no cache hits, we assume the same 91% share of cached input as in the corresponding GPT-5.4 run.

Method	Description	QALD-F_1	Details
English without-mentions track			
GRASP	GPT-5.4 with samples	85.0	56.3 s/q, 13.7 S/q, $9
Claude Code	Opus 4.8 using Sonnet 4.6 sub-agents with samples	85.0	113 s/q, $26 sub-agents, $29 orchestrator
GRASP	Kimi K2.7 with samples	81.0	122.8 s/q, 16.3 S/q, $5 est.
Other	agentic-text2sparql-skills	81.0	from leaderboard
GRASP	GPT-5.4	80.0	56.1 s/q, 12.4 S/q, $8
Other	AMA-KBQA	79.0	from leaderboard
GRISP	Qwen2.5 14B WDQL fine-tuned on all KGQA sources	57.0	18.7 s/q
Spanish without-mentions track			
GRASP	GPT-5.4 with Spanish setup	83.0	68.4 s/q, 13.7 S/q, $10
GRASP	GPT-5.4 with Spanish setup and samples	83.0	66.9 s/q, 13.6 S/q, $10
GRASP	Kimi K2.7 with Spanish setup and samples	79.0	122.1 s/q, 13.7 S/q, $4 est.
GRISP	Qwen2.5 14B WDQL fine-tuned on all KGQA sources	56.0	24.3 s/q

training on all KGQA sources instead of only the track's WikiKGQA split adds another 5–8 points for the best configuration (63.2 to 68.7 English, 61.8 to 69.3 Spanish). Even the bare 14B WDQL checkpoint (58.2), without any fine-tuning at all, already outperforms several fine-tuned 7B configurations.

Claude Code The strongest off-the-shelf Claude agent (Opus 4.8 orchestrating Sonnet 4.6 sub-agents, shown all validation questions upfront) reaches an F_1 close to GRASP's best (86.6 vs. 86.8), while the remaining configurations trail well behind. On test (Table 2), the same orchestrator setup, but with the train and validation samples in place of the questions upfront (Section 3.3), costs about $26 in sub-agent calls plus $29 for orchestration at 113 s/q, compared to $9 for the corresponding GRASP run (GPT-5.4 with samples) at 56 s/q, or $9 plus $16 including GRASP's one-time setup cost. The $26 and the $29 are estimated from the session traces at published token prices, for a run spread over several days.

Across both systems, F_1 is consistently lower on the held-out test set than on the corresponding validation split, and every GRASP configuration also takes more steps per question there (9.6 on validation versus 12.4 and 13.7 on test for GPT-5.4 on English). The drop affects all our runs, most of which are zero-shot, while the rest are not heavily tuned on the validation split, so we do not attribute it to overfitting, but to the test set simply being harder.

5. Conclusion

We submitted GRASP and GRISP, two knowledge-graph-agnostic systems, to the without-mentions tracks of the WikiKGQA 2026 challenge. GRASP achieves the best result among all submissions in both without-mentions tracks, matching our strongest Claude Code baseline at a fraction of its cost, while GRISP trades some accuracy for much lower cost and fully local execution.

As future work, we plan to evaluate GRASP and GRISP on other knowledge graphs, relying on GRASP's ability to set itself up on a new knowledge graph without manual effort. We would also welcome future challenges that target efficiency, since our best-performing configurations are still fairly slow, and that use a knowledge graph unseen by systems beforehand, since Wikidata's prevalence in LLM training data likely washes out some of the difference a better method would otherwise make.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 499552394 – SFB 1597.

Declaration on Generative AI

All methods evaluated in this paper are themselves based on generative AI, i.e. large language models, so their use is inherent to the approaches described here. In addition, during the preparation of this paper the authors used AI to improve and polish the writing, and to rephrase existing paragraphs. In all cases, the authors carefully checked the AI-generated writing and take full responsibility for the content of this paper.

References

[1] D. Vrandecic, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85. URL: https://doi.org/10.1145/2629489. doi:10.1145/2629489.

[2] S. Walter, H. Bast, GRASP: Generic Reasoning And SPARQL Generation Across Knowledge Graphs, in: ISWC (1), volume 16140 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 271–289. doi:10.1007/978-3-032-09527-5_15.

[3] S. Walter, H. Bast, GRISP: Guided Recurrent IRI Selection over SPARQL Skeletons, CoRR abs/2604.21133 (2026). URL: https://doi.org/10.48550/arXiv.2604.21133. doi:10.48550/ARXIV.2604.21133. arXiv:2604.21133.

[4] Anthropic, Claude Code, https://www.anthropic.com/claude-code, 2025. URL: https://www.anthropic.com/claude-code.

[5] 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. URL: https://doi.org/10.5281/zenodo.19474009. doi:10.5281/zenodo.19474009.

[6] R. Usbeck, M. Röder, M. Hoffmann, F. Conrads, J. Huthmann, A. N. Ngomo, C. Demmler, C. Unger, Benchmarking question answering systems, Semantic Web 10 (2019) 293–304. URL: https://doi.org/10.3233/SW-180312. doi:10.3233/SW-180312.

[7] H. Bast, B. Buchhold, QLever: A Query Engine for Efficient SPARQL+Text Search, in: E. Lim, M. Winslett, M. Sanderson, A. W. Fu, J. Sun, J. S. Culpepper, E. Lo, J. C. Ho, D. Donato, R. Agrawal, Y. Zheng, C. Castillo, A. Sun, V. S. Tseng, C. Li (Eds.), Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017, ACM, 2017, pp. 647–656. URL: https://doi.org/10.1145/3132847.3132921. doi:10.1145/3132847.3132921.

[8] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, et al., Qwen2.5 technical report, arXiv preprint arXiv:2412.15115 (2024). URL: https://arxiv.org/abs/2412.15115.

[9] S. Walter, H. Bast, The Wikidata Query Logs Dataset, in: A. Moffat, F. Scholer, H. Bast, M. Najork, M. Zhang (Eds.), Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2026, Melbourne VIC Australia, July 20-24, 2026, ACM, 2026, pp. 3442–3447. URL: https://doi.org/10.1145/3805712.3808612. doi:10.1145/3805712.3808612.

[10] R. Usbeck, A. N. Ngomo, B. Haarmann, A. Krithara, M. Röder, G. Napolitano, 7th Open Challenge on Question Answering over Linked Data (QALD-7), in: M. Dragoni, M. Solanki, E. Blomqvist (Eds.), Semantic Web Challenges - 4th SemWebEval Challenge at ESWC 2017, Portoroz, Slovenia, May 28 - June 1, 2017, Revised Selected Papers, volume 769 of *Communications in Computer and Information Science*, Springer, 2017, pp. 59–69. URL: https://doi.org/10.1007/978-3-319-69146-6_6. doi:10.1007/978-3-319-69146-6_6.

[11] R. Usbeck, X. Yan, A. Perevalov, L. Jiang, J. Schulz, A. Kraft, C. Möller, J. Huang, J. Reineke, A. N. Ngomo, M. Saleem, A. Both, QALD-10 - The 10th challenge on question answering over linked data: Shifting from DBpedia to Wikidata as a KG for KGQA, Semantic Web 15 (2024) 2193–2207. URL: https://doi.org/10.3233/SW-233471. doi:10.3233/SW-233471.

[12] OpenAI, GPT-5.4 Thinking system card, https://openai.com/index/gpt-5-4-thinking-system-card/, 2026. URL: https://openai.com/index/gpt-5-4-thinking-system-card/.

[13] OpenAI, GPT-5.5 system card, https://openai.com/index/gpt-5-5-system-card/, 2026. URL: https://openai.com/index/gpt-5-5-system-card/.

[14] K. Team, et al., Kimi K2: Open agentic intelligence, 2025. URL: https://arxiv.org/abs/2507.20534. arXiv:2507.20534.

[15] Anthropic, System card: Claude Sonnet 4.6, https://www.anthropic.com/claude-sonnet-4-6-system-card, 2026. URL: https://www.anthropic.com/claude-sonnet-4-6-system-card.

[16] Anthropic, System card: Claude Opus 4.8, https://www-cdn.anthropic.com/0b4915911bb0d19eca5b5ee635c80fef830a37ea.pdf, 2026. URL: https://www-cdn.anthropic.com/0b4915911bb0d19eca5b5ee635c80fef830a37ea.pdf.

[17] Anthropic, System card: Claude Haiku 4.5, https://www.anthropic.com/claude-haiku-4-5-system-card, 2025. URL: https://www.anthropic.com/claude-haiku-4-5-system-card.